HCSS User's Reference Manual

Herschel Data Processing

Version 10.0, Document Number: HERSCHEL-HSC-DOC-0935
10 April 2017

HCSS User's Reference Manual: Herschel Data Processing

Table of Contents

List of Tables

Categorised view of commands

This chapter provides a categorised view of all built-in DP functions, tasks and objects.

Arrays and datasets

- Bool1d - A rectangular numeric boolean array of rank 1.
- Bool2d - A rectangular numeric boolean array of rank 2.
- Bool3d - A rectangular numeric boolean array of rank 3.
- Bool4d - A rectangular numeric boolean array of rank 4.
- Bool5d - A rectangular numeric boolean array of rank 5.
- Byte1d - A rectangular numeric byte array of rank 1.
- Byte2d - A rectangular numeric byte array of rank 2.
- Byte3d - A rectangular numeric byte array of rank 3.
- Byte4d - A rectangular numeric byte array of rank 4.
- Byte5d - A rectangular numeric byte array of rank 5.
- Complex1d - A rectangular numeric Complex array of rank 1.
- Complex2d - A rectangular numeric Complex array of rank 2.
- Complex3d - A rectangular numeric Complex array of rank 3.
- Complex4d - A rectangular numeric Complex array of rank 4.
- Complex5d - A rectangular numeric Complex array of rank 5.
- Double1d - A rectangular numeric double array of rank 1.
- Double2d - A rectangular numeric double array of rank 2.
- Double3d - A rectangular numeric double array of rank 3.
- Double4d - A rectangular numeric double array of rank 4.
- Double5d - A rectangular numeric double array of rank 5.
- Float1d - A rectangular numeric float array of rank 1.
- Float2d - A rectangular numeric float array of rank 2.
- Float3d - A rectangular numeric float array of rank 3.
- Float4d - A rectangular numeric float array of rank 4.
- Float5d - A rectangular numeric float array of rank 5.
- Int1d - A rectangular numeric int array of rank 1.
- Int2d - A rectangular numeric int array of rank 2.
- Int3d - A rectangular numeric int array of rank 3.
- Int4d - A rectangular numeric int array of rank 4.
- Int5d - A rectangular numeric int array of rank 5.
- Long1d - A rectangular numeric long array of rank 1.
- Long2d - A rectangular numeric long array of rank 2.

Long3d - A rectangular numeric long array of rank 3.

Long4d - A rectangular numeric long array of rank 4.

Long5d - A rectangular numeric long array of rank 5.

ObservationContext - An Observation Context is a container of Products applicable to a specific observation.

Polynomial - Returns the value of a polynomial, given its coefficients, at a series of positions.

Short1d - A rectangular numeric short array of rank 1.

Short2d - A rectangular numeric short array of rank 2.

Short3d - A rectangular numeric short array of rank 3.

Short4d - A rectangular numeric short array of rank 4.

Short5d - A rectangular numeric short array of rank 5.

String1d - A rectangular numeric String array of rank 1.

Display

OverPlotter - Overview of OverPlotter

TablePlotter - TablePlotter is a GUI tool to view and analyze instances of TableDataset.

Element selection

ALL - Determines whether all the elements of an array satisfy an expression.

AllPresent - Checks whether all the bits in the specified bitmask are present in the elements of the input array.

ANY - Determines whether any element of an array satisfies an expression.

AnyPresent - Checks whether any of the bits in the specified bitmask are present in the elements of the input array.

IS_ANY_NAN - Checks if the given array contains a NaN value.

IS_FINITE - Checks whether a number or the elements of an array are finite.

IS_INFINITE - Checks whether a number or the elements of an array are infinite.

IS_NAN - Checks whether a number or the elements of an array are NaN (Not a Number).

NAN_FILTER - Returns an array without any NaN (Not a Number) elements.

NotPresent - Checks whether none of the bits in the specified bitmask are present in the elements of the input array.

UNIQ_SORTED - Returns the unique elements of a sorted array.

UNIQ - Returns the unique elements of an array.

Manipulation

CONCATENATE - Concatenates an array of rank greater than one into an array of rank 1 of the same type.

convertAngles - Converts angles between numeric and textual representations.

convertUnits - Converts numeric data expressed in one unit, in terms of another unit.

FIX - For each element of an array, returns the largest integer less than or equal to the element.

Normalize - Normalizes sets of data.

PRODUCT - Returns the product of all the elements of an array.

REPEAT - Creates a new array which contains the specified repetitions of the original array.

RESHAPE - Creates a new array of the same type as the input array but with a different shape.

REVERSE - Reverses the order of the elements of a one-dimensional array.

Rotate - Rotates a Numeric two- or three-dimensional array.

ROUND - Returns the input number or array rounded to the nearest integer or decimal position.

SHIFT - Shifts the elements of an array along the specified dimension.

SORT - Sorts the specified array into ascending natural order.

sortTable - Returns a new table with rows sorted.

UNWRAP - Unwraps the radian phase angles in an array.

updateDataset - Updates a dataset in a context hierarchy so that the changes are preserved in the context tree.

updateMetadata - Updates metadata in a context hierarchy so that the changes are preserved in the context tree.

Reduction

Condense - Condenses an array along a given dimension by applying a function to groups of array items.

MAX - Returns the value of the largest element in the input array.

MIN - Returns the value of the smallest element in the input array.

StatWithNaN - Removes NaN variables from input arrays to other numeric functions.

SUM - Returns the sum of all the elements of an array.

Astronomical utilities

Aberration - Obtain astrometric and geometric positions from apparent ones.

astrometryFix - This task changes the astrometry in the input data to make it consistent with some other data.

Planck - Computes the Planck function.

pointHistoryDisplay - Task for displaying graphs of pointing information.

RadialV - Calculates S/C velocity projection in the direction of the pointing (radial velocity).

radialV - The RadialV Task calculates S/C radial velocity projection in the direction of the pointing.

Sso - Tools for SSO centred coordinate systems.

Photometry

annularSkyAperturePhotometry - This is a task that performs aperture photometry of target, enclosed

AnnularSkyAperturePhotometryProduct - This is a class to deal with the results of aperture photometry with a circular target aperture and an annular sky aperture.

fixedSkyAperturePhotometry - This is a task for aperture photometry for a circular target aperture.

FixedSkyAperturePhotometryProduct - This is a class to deal with the results of aperture photometry with a circular target aperture and a fixed value for the sky intensity.

rectangularSkyAperturePhotometry - This is a task for aperture photometry for a rectangular target

RectangularSkyAperturePhotometryProduct - This is a class to deal with the results of aperture photometry with a circular target aperture and an rectangular sky aperture.

SourceFittingProduct - A class to handle the result of the SourceFittingTask.

Source extraction

PrfGaussian - A representation of a Gaussian point response function as an Image.

PrfImage - A representation of a point response function based on an image.

sourceExtractorDaophot - This task extracts point sources from an HCSS SimpleImage image using DAOPHOT.

sourceExtractorSimultaneous - This task measures point source photometry of multiple sources simultaneously using linear inversion method.

sourceExtractorSussextractor - This task extracts point sources from an HCSS SimpleImage image using SUSSEXtractor.

sourceFitting - This is a task to fit a two-dimensional gaussian to a source in a

Data access

AttribQuery - Attribute Query formulates a query on the attributes of a Product.

CachedPool - Implementation of a PAL ProductPool that adds caching functionality to a remote ProductPool.

exportObservation - Exports observations from a pool to an HSA hierarchical directory structure.

FullQuery - A data mining query formulates a query the full interface of a Product.

HttpClientFactory - Factory class to create HttpClientPool instances.

ListContext - Groups products (or other Contexts that in turn group products) in a list-like structure.

localStoreCopier - Creates a copy of a LocalStore (pool).

MapContext - Groups products (or other Contexts that in turn group products) in a map-like structure.

MetaQuery - Meta data query formulates a query on the meta data of a Product.

PoolManager - The PoolManager provides the means to reference Product Pools without

ProductRef - A ProductRef provides a reference to a product that is held in a product storage or to a product in memory.

ProductStorage - The ProductStorage is a storage mechanism to provide read, write and

saveProduct - The product is saved to the pool you pass.

saveProduct - The product is saved to the pool you pass.

Data cubes

importCube - The ImportCube task for Cubes.

importSpectralCube - The ImportSpectralCube task for SpectralSimpleCubes.

nearestNeighbourProjection - Creates a spectral cube by converting the pixels specified by a Wcs into world coordinates and assigning the nearest spectral data to each pixel.

SimpleCube - A Product describing cubes.

SpectralSimpleCube - SpectralSimpleCube is a SimpleCube implementation where the 3rd axis represents a spectral axis.

Wcs - A class to create a Wcs.

Analysis

computePVMap - This task computes a Position-Velocity (PV) map, which is a cross-section of a spectral cube along a given slit.

computeVelocityMap - VelocityPosMapComputeTask computes several velocity related values from a processed cube.

cropCube - Extracts a sub-cube from the provided Cube.

extractRegionSpectrum - Extracts a spectrum from either the whole image, a single pixel, or an ellipsoid, rectangular or slit region.

flagSaturatedPixelsCube - A Task to flag saturated pixels in a cube.

integrateSpectralMap - Integrates a spectral cube over one or several ranges in order to produce a series of integrated maps.

removeBaselineFromCube - Simple baseline subtraction task.

History

historyExtract - This task extracts a specific parameter value from a task in the history of a product.

Images

Flag - A class to describe Flags for images (SimpleImage and SimpleCube).

imageSaver - This is a task which creates a grey colors JPEG file from an image.

importImage - This is a task to import an image from a file.

importRgbImage - This is a task to import an RGB image from a file.

RgbSimpleImage - A Product describing three colour images.

SimpleImage - A Product describing images.

Wcs - A class to create a Wcs.

Analysis

autoCorrelation - This is a task to calculate the linear Pearson correlation coefficients of an image.

automaticContour - This is a task that generates the contours for a given number of contour levels, distribution and extreme values.

boxCarSmoothing - This is a task to smooth/filter an image using a convolution with a box car.

circleHistogram - This is a task to make a histogram of a region of interest on an image which is bounded by a circle.

clamp - This is a task which restricts the range of intensity given low and high values in an image.

convertImageUnit - This is a task to convert the unit of an image from one surface brightness unit to another one.

createRgbImage - This is a task to combine three images to an RGB-image.

crop - This is a task to crop an image to a rectangular region.

crossCorrelation - This is a task to calculate the linear Pearson correlation coefficient of two images.

cutLevels - This is a task to determine the cut levels of an image.

flagSaturatedPixels - This is a task to flag saturated pixels in an image.

gaussianSmoothing - This is a task to smooth/filter an image using a convolution with a gaussian.

imageAbs - This is a task that takes the absolute value of the intensity values of an image.

imageAdd - This task is able to add two images or a scalar to an image, using pixel positions or Wcs values.

imageCeil - This is a task that ceils the intensity values of an image.

imageContourSaver - This is a task to save image contours as a ds9 region file.

imageConvolution - This is a task to convolve an image with a given convolution kernel of the same dimensions.

imageDivide - This is a task to divide two images pixel-by-pixel (ref = 0) or based on their Wcs (ref = 1), or to divide all intensity values in an image by a scalar.

imageExp10 - This is a task that allows to change the intensity values of an image according to a exp10 scaling.

imageExp - This is a task that allows to change the intensity values of an image according to a exp scaling.

imageExpN - This is a task that allows to change the intensity values of an image according to a expN scaling.

imageFloor - This is a task that floors the intensity values of an image.

imageHistogram - This is a task to make a histogram of an image as a whole.

imageLog10 - This is a task that allows to change the intensity values of an image according to a log10 scaling.

imageLog - This is a task that allows to change the intensity values of an image according to a log scaling.

imageLogN - This is a task that allows to change the intensity values of an image according to a logN scaling.

imageModulo - This is a task that allows to take the modulo of an image, either with another image or with a scalar.

imageMultiply - This is a task to multiply two images or an image and a scalar.

imagePower - This is a task that changes the intensity values of an image according to a power scaling.

imageRound - This is a task that rounds the intensity values of an image.

imageSqrt - This is a task that changes the intensity values of an image according to a sqrt scaling.

imageSquare - This is a task that changes the intensity values of an image according to a square scaling.

imageSubtract - This is a task to subtract two images pixel-by-pixel (ref = 0) or based on their Wcs (ref = 1), or to subtract a scalar from all intensity values in an image.

IntensityCalculator - This is a class to apply sky estimtion algorithms.

meanSmoothing - This is a task to smooth/filter an image by applying a mean filter.

medianSmoothing - This is a task to smooth/filter an image by applying a median filter.

mosaic - This is a task to make mosaics, by

nSigmaClip - This is a task to flag image values for which |value| is greater than n·σ.

profile - This is a task to make intensity plots along a straight line.

regrid - This is a task to regrid an image.

rotate - This is a task to rotate an image.

scale - This is a task to scale an image, and adapts its Wcs.

tiledImage - This is a task to convert an image to a TiledImage.

translate - This is a task to translate an image, and adapts its Wcs.

transpose - This is a task to transpose an image, and adapts its Wcs.

Display

AnnotationToolbox - A toolbox to draw annotations.

CircleHistogramProduct - A class to deal with the results of a circle histogram.

Contour - A class to deal with Contours.

contour - This task generates the contours for one given contour value.

ContourLevel - A class to deal with ContourLevels.

Display - A class to display images.

ellipseHistogram - This is a task to make a histogram of a region of interest on image which is bounded by an ellipse.

EllipseHistogramProduct - A class to deal with the results of an ellipse histogram.

ImageAxis - Axes for image display.

ImageHistogramProduct - A class to deal with the results of an image histogram.

manualContour - This is a task that generates the contours for a given list of contour values.

Polygon - A polygon shape.

polygonHistogram - This is a task to make a histogram of a region of interest on am image which is bounded by a polygon.

PolygonHistogramProduct - A class to deal with the results of a polygon histogram.

PolygonIntersect - This is a class to calculate the overlap between polygons.

PositionList - Position list.

Profile - A class to deal with the results of profile plotting.

rectangleHistogram - This is a task to make a histogram of a region of interest on an image which is bounded by a rectangle.

RectangleHistogramProduct - A class to deal with the results of a rectangle histogram.

SkyMaskToolbox - SkyMaskToolbox construct a toolbox where the user can annotate an image and create a SkyMask from the annotations.

Input-output

asciiTableWriter - Task for writing TableDatasets into ASCII (text) files.

FitsArchive - A class for reading and writing FITS files.

fitsReader - Task to read and import arbitrary products in FITS files into HIPE.

getObservation - Task to download, browse, or import an Observation.

HduHeaders - A class for working with FITS HDUs. HDU data is loaded only when requested.

imageSaver - This is a task which creates a grey colors JPEG file from an image.

importCube - The ImportCube task for Cubes.

importImage - This is a task to import an image from a file.

importRgbImage - This is a task to import an RGB image from a file.

importSpectralCube - The ImportSpectralCube task for SpectralSimpleCubes.

saveObservation - Save an observation into a local pool.

SerialArchive - A class for writing and reading serialised objects.

simpleFitsReader - Task to read HCSS products (and simple external FITS).

simpleFitsWriter - Saves a product in a FITS file.

Mathematics

Fitting

AbstractBasicModel - AbstractBasicModel implements the common parts of Models and in particular those of simple or basic Models.

AbstractFitter - Fitter for linear models.

AbstractModel - AbstractModel implements the common parts of (compound) Models which can be handled by the Fitter class and its descendants.

AmoebaFitter - Simulated annealing simplex finding minimum.

ArctanModel - ArcTangus Model.

ArrayAssistant - ArrayAssistant contains 2 methods to assist with more dimensional fitting.

BinomialModel - General binomial model of arbitrary degree.

CauchyErrorDistribution - CauchyErrorDistribution aka Lorentz distribution.

ChebyshevPolynomialModel - Chebyshev polynomial model of arbitrary degree.

ComboModel - ComboModel combines a number of copies of the same basic model.

ConjugateGradientFitter - Non-linear fitter using the conjugate gradient method.

ConstantModel - ConstantModel is an extension of NullModel. It does not have any parameters.

ContainerModel - ContainerModel is a AbstractModel that contains another AbstractModel.

EngineList - EngineList is a ArrayList of Engines generated by a Enginer.

ExpModel - Exponential Model.

ExponentialPrior - Exponential prior distribution.

FitterFunction - Specialization of RealFunction that fits some given data.

Fitter - Fitter for linear models.

FreeShapeModel - Free Shape Model.

Gauss2DModel - Rotationally symmetric two dimensional Gaussian Model.

Gauss2DRotModel - Rotated asymmetric 2 dimensional Gaussian Model.

GaussErrorDistribution - GaussErrorDistribution models the fitting error using a Gauss distribution.

GaussModel - Gaussian Model.

HarmonicDynamicModel - Harmonic oscillator Model of adaptable Order.

HarmonicModel - Harmonic oscillator Model.

JeffreysPrior - Jeffreys prior distribution, for scale-like parameters.

Kernel2dModel - Two dimensional Kernel Model.

Kernel - A Kernel is a non-negative real-valued integrable function.

KernelModel - Kernel Model, a Model build around an Kernel.

KernelModel - Kernel Model, a Model build around an Kernel.

LaplaceErrorDistribution - LaplaceErrorDistribution.

LaplacePrior - Laplace prior distribution.

LevenbergMarquardtFitter - Non-linear fitter using the Levenberg-Marquardt method.

LorentzModel - Lorentzian Model.

modelFit - Task shell around the package herschel.ia.numeric.toolbox.fit.

MonteCarloError - Fitter for linear models.

MpFitter - Non-linear fitter using MINPACK minimization.

NestedSampler - NestedSampler can be used to fit data to a model.

NoiseScale - NoiseScale contains a measure of the noise.

NullModel - NullModel represents models without parameters.

PadeModel - General Pade model of arbitrary degrees in numerator and denominator.

PoissonErrorDistribution - PoissonErrorDistribution models the fitting error using a Poisson distribution.

PolynomialDynamicModel - General polynomial model of an adaptable degree.

PolynomialModel - General polynomial model of arbitrary degree.

PolySurfaceModel - General polynomial surface model of arbitrary degree.

PowerLawModel - General powerlaw model of arbitrary degree.

PowerModel - General power model of arbitrary degree.

PriorList - PriorList is a ArrayList of AbstractPrior elements.

RobustClipWeight - Clip weighting scheme. Also known as sigma-clipping.

RobustCosineWeight - Cosine weighting scheme.

RobustMedianWeight - Median weighting scheme; also known as Huber.

RobustShell - RobustShell is a shell around another fitter to robustify the fit of the inner fitter.

RobustTukeyWeight - Tukey's weighting scheme.

SampleList - SampleList is a ArrayList of Samples generated by a Sampler.

SideBandGED - SideBandGED is a special case (with a sideband) of the GaussErrorDistribution.

SideBandRatioGED - SideBandRatioGED is a special case of the GaussErrorDistribution that considers the sideband ratio.

SincGaussModel - Sinc-Gauss Model.

SincModel - Sinc Model.

SineAmpModel - Find amplitudes/phases for sinusoidal of a given frequency.

SineModel - Sinusoidal Model.

SingularValueDecompositionFitter - SingularValueDecompositionFitter implements a linear fitter based on Singular Value Decomposition (SVD).

SkewGaussModel - A class that defines a skew gaussian model.

sourceFit - Task to easily fit Source models (Gauss, Lorentz, Voigt etcetera).

SplinesModel - General splines model of arbitrary order and with arbitrary knot settings.

SurfaceSplinesModel - General surface splines model of arbitrary order and with arbitrary knot settings.

UniformPrior - Uniform prior distribution.

VoigtModel - Voigt Model.

General functions

ABS - Returns the absolute value of a number or a numeric array. For complex numbers, ABS(x) returns the modulus.

EXP10 - Returns the base 10 exponential function applied to a number or array.

EXP - Returns the natural exponential function applied to a number or array.

ExpN - Returns the base n exponential function applied to a number or array.

LOG10 - Returns the base 10 logarithm of a number or array.

LogFactorial - To get a log factorial of a value.

LOG - Returns the natural logarithm of a number or array.

LogN - Returns the base n logarithm of a number or array.

Pow - Raises a number or the elements of an array to the specified power.

SIGNUM - Returns the sign function of a number or array.

SQRT - Returns the square root of a number or array.

SQUARE - Returns the square of a number or array.

Histograms

BinCentres - Returns an array of histogram bin centres for your data, based on bin size.

Histogram - Returns histogram column sizes for an array, using a user-specified bin size.

Integration

DERIV - Given a set of knots, this function performs numerical differentiation using 3-point, Lagrangian interpolation.

Integrator - Interface for all integrators.

IntTabulated - Integrate tabular data similar to IDL's INT_TABULATED.

Interpolation

Bilinear - The Bilinear class is an interpolation algorithm used to compute the value between regular grid points in an 2D array.

CubicSplineInterpolator - Given a set of knots (x,y), this function computes values at arbitrary positions by use of a cubic spline.

LinearInterpolator - This class linearly interpolates unknown values based on the two closest known knots (x,y data).

NearestNeighborInterpolator - This function interpolates values based on a set of known knots (x,y data).

PreviousInterpolator - Creates an linear interpolation function from a set of knots (x,y), that can be applied to numeric arrays of rank 1.

Rebin - The Rebin class resizes a vector or array to dimensions given by the parameters Di.

Regrid - The Regrid class shrinks or expands the size of an array by an arbitrary amount.

Masks

FixedMask - FixedMask represents the more traditional mask form, where a mask is defined at a fixed bit offset position.

PackedMask - PackedMask creates and stores compressed mask array data which can be set, unset and checked.

Matrices

CholeskyDecomposition - CholeskyDecomposition implements the Cholesky method for decomposing a matrix.

DETERMINANT - Yields the determinant of a square matrix.

EigenvalueDecomposition - EigenvalueDecomposition

INVERSE - Returns the inverse of a square matrix.

MATRIXMULTIPLY - Performs matrix multiplication of numeric or logical matrices.

MATRIXSOLVE - Solves systems of linear equations of type A x = b.

QRDecomposition - QRDecomposition

SingularValueDecomposition - SingularValueDecompositon

TRANSPOSE - Transposes a matrix.

Nearest integer

CEIL - Returns the largest integer greater than or equal to the input.

FLOOR - Returns the largest integer less than or equal to the input.

ROUND - Returns the input number or array rounded to the nearest integer or decimal position.

Random numbers

MoreRandom - More Random utilities for use with the rest of classes of this package.

RandomGauss - RandomGauss generates or populates an array with normally N(0,1) distributed pseudo random numbers.

RandomPoisson - RandomPoisson generates Poisson random numbers sequence with a given mean.

RandomUniform - RandomUniform generates or populates an array with uniformly-distributed pseudo-random

Signal processing

BoxCarFilter - Creates a box car filter, that can be applied to numeric arrays of rank 1 and 2.

Convolution - This class implements the convolution of a one- or two-dimensional array with a one-dimensional convolution kernel.

dFT2d - This is a task to calculate the Discrete Fourier Transform and the power spectrum of an image.

fFT2d - This is a task to calculate the Fast Fourier Transform and the power spectrum of an image.

FFT_AUTO - Gives the Fast Fourier Transform, automatically choosing what should generally be the fastest implementation.

FFT_PACK_EVEN - Gives the Discrete Cosine Transform (DCT) of real data.

FFT_PACK_ODD - Gives the Discrete Sine Transform (DST) of real data.

FFT_PACK - Gives the Fast Fourier Transform of complex data using an FFT_PACK algorithm.

FFT - Gives the Fast Fourier Transform.

GaussianFilter - Creates a Gaussian filter, that can be applied to numeric arrays of rank 1 and 2.

HAMMING - Hamming function for multiplying with input arrays.

HANNING - Hanning function for multiplying with input arrays.

inverseDFT2d - This is a task to calculate the inverse Discrete Fourier Transform and the power spectrum of an image.

inverseFFT2d - This is a task to calculate the inverse Fast Fourier Transform and the power spectrum of an image.

PowerSpectrum - A PowerSpectrum is a normalised Fourier Transform of the time series.

RealDoubleFFT - Gives the Fast Fourier Transform of real data using an FFT_PACK algorithm.

Statistics

ChiSquared - ChiSquared computes the chi^2 and significance of the null hypothesis that an

Correlate - Returns the linear Pearson correlation coefficient of the input arrays.

CorrelateMatrix - Returns the linear Pearson correlation coefficients of the input m x n matrix.

Covariance - Yields the covariance between two random variables/vectors x and y with finite second moments

CovarianceMatrix - Yields the covariance matrix of the input M x N matrix

ERFC - Returns the complementary error function of a number or array.

ERF - Returns the error function of a number or array.

GAMMALN - Returns the natural logarithm of the Gamma function.

GammaP - Returns the incomplete Gamma function P(a,x).

GammaQ - Returns the complement of the incomplete Gamma function Q(a,x).

GEOMEAN - Yields the geometric mean value of the elements in the input array.

KURTOSIS - Returns the kurtosis excess of an array.

MEAN - Returns the mean of an array.

MedianAbsoluteDeviation - Returns the median absolute deviation of an array.

MEDIAN - Returns the median of an array.

MODE - Yields mode(s), or the most common element(s), in an array of rank 1 to 5.

QRMS - Returns the quadratic root mean square of a set of values.

RMS - Returns the root mean square of a set of values.

Sigclip - Finds and removes outliers in an array.

SKEWNESS - Returns the skewness of an array.

StatWithNaN - Removes NaN variables from input arrays to other numeric functions.

STDDEV - Returns the standard deviation of an array.

VARIANCE - Returns the variance of an array.

WeightedMean - Returns the quadratically weighted mean of an array.

Trigonometry

ARCCOS - Returns the inverse cosine of a number or array.

ARCCOSH - Returns the inverse hyperbolic cosine of a number or array.

ARCSIN - Returns the inverse sine of a number or array.

ARCSINH - Returns the inverse hyperbolic sine of a number or array.

ARCTANCONTFRAC - Returns the inverse tangent of a number or array using Euler's continued fraction approximation.

ARCTAN - Returns the inverse tangent of a number or array.

ARCTANH - Returns the inverse hyperbolic tangent of a number or array.

COS - Returns the cosine of a number or array.

COSH - Returns the hyperbolic cosine of a number or array.

SINC - Returns the sinc of a number or array.

SIN - Returns the sine of a number or array.

SINH - Returns the hyperbolic sine of a number or array.

TAN - Returns the tangent of a number or array.

TANH - Returns the hyperbolic tangent of a number or array.

UNWRAP - Unwraps the radian phase angles in an array.

Session utilities

bg - Execute Jython commands in the background.

clear - Remove variables from the Jython session.

compress - Compresses files and directories.

decompress - Decompresses archives.

help - Display help page associated with an object.

openFile - Opens a file in a viewer.

openTask - Opens a task in its viewer.

openVariable - Opens a variable in a viewer.

restore - Restore variables from a file to a jython session.

save - Save variables from the jython session to a file.

Spectra

openSE - Opens a Spectrum Explorer in the Editor Area.

SimpleSpectrum - SimpleSpectrum is a simple wrapper to transform a Spectrum1d into a Product.

SpectralLineList - SpectralLineList is a simple Product which implements the Herschel LineList definition.

SpectralSimpleCube - SpectralSimpleCube is a SimpleCube implementation where the 3rd axis represents a spectral axis.

Spectrum1d - Spectrum1d is an extension of AbstractSpectrumDataset which in turn is an extension of (Strict)TableDataset.

Spectrum2d - Spectrum2d is an extension of AbstractSpectrumDataset which in turn is an extension of (Strict)TableDataset.

SpectrumContainerBox - Defines the requirements for an object that contains a number of SpectrumContainers.

SpectrumContainer - Interface used to access spectrum information sitting in diverse data structures.

Analysis

accumulate - Accumulates (averages) spectra to a common wave scale grid and returns a Spectrum1d.

add - Task for adding a scalar to the flux data of spectra or for pairwise adding spectra.

avg - Task for averaging spectra.

convertWavescale - Task for transforming the wavescale between frequency (w), velocity (v), wavelength (l) and wavenumber (k).

divide - Task for dividing the flux data of spectra by a scalar or for pairwise dividing spectra.

exportSpectrumToAscii - Task for exporting spectrum data to ASCII.

extract - Task for extracting a wavescale range (or ranges) from a spectrum or set of spectra.

filterSpectrum - A Task which apply a filter on a spectrum (Double1d).

FitFringeData - Class that wraps Double1d's into a SpectralSegment.

fitFringe - Task that fits and removes fringes from spectra.

flagPixels - Task for flagging pixels in spectra.

fold - Task for folding frequency-switched spectra.

importSpectrumFromAscii - Task for importing spectrum data from an ASCII file into a HCSS spectrum data structure.

multiply - Task for multiplying the flux data of spectra by a scalar or for pairwise multiplying spectra.

pairAvg - Task for pairwise averaging spectra.

replace - Task for inserting and/or replacing the ranges of given spectra by the ranges spanned by other spectra.

resample - Task for resampling the flux values of spectra with respect to a modified wavescale grid.

select - Task for selecting individual spectra and pack them in a new so called spectrum container.

smoothBaseline - A task that generates a smooth and clipped version of the observation flux.

smooth - Task for smoothing spectra using a suitable smoothing filter (kernel).

statistics - Task to compute statistical characteristics for the spectrum data.

stitch - Task for stitching the overlapping parts of spectra.

subtract - Task for subtracting a scalar from the flux data of spectra or for pairwise subtraction of spectra.

Toolboxes

Pointing

angles2RotationMatrix - Function to go from Euler angles to a 3x3 rotation matrix.

axisAngle2RotationMatrix - Calculate rotation matrix about an arbitrary axis.

calc_errcov - Calculates the (rotation angle) error covariance matrix.

calcAttitude - Creates a new pointing product with reconstructed filterQuat.

calcGyroAttitude - Constructs a table containing the reconstructed ACA-frame attitude.

calcStrAttitude - Constructs a table containing the corrected attitude measurements made by the (operational) star tracker.

DistortionMaps - Class providing access to the Herschel Distortion Maps via HSA or local file.

est_attitude_new - A new version of the function to estimate the star tracker attitude.

est_attitude - Estimate the star tracker attitude, eliminating bad stars if necessary.

qMethod - Implementation of Davenport's q-method.

quat_to_att - Converts an attitude quaternion into an attitude matrix.

raDecRoll2xyz - Function to go from difference in ra/dec/roll to spacecraft xyz axis angular offsets.

StrCalibration - Class providing time-dependent access to the Herschel Star Tracker (STR) properties.

strExtractVelocityFromTeleCommHistory - Function to extract the spacecraft velocity used for on-board Star tracker aberration correction from the telecommand history.

xyz2raDecRoll - Function to go from difference in spacecraft x,y,z to ra/dec/roll.

binstruct

PacketSequence - A container for telemetry and telecommand source packets.

How to use this manual

The *User's Reference Manual* contains information about all the main tasks and classes that you can use within your scripts. There are four version of this manual: the *HCSS* version describes the functions provided by the core Herschel software, and is always shipped with HIPE; the three other versions describe functions provided by HIFI, PACS and SPIRE software. You may or may not have these additional manuals, depending on the software you installed.

This manual includes the following sections:

- Categorised view of commands. All the functions described in the manual, organised by category.
- Dictionary. A list of definitions of words and acronyms you are likely to encounter in documentation on the Herschel satellite.
- Section 1.1. This section lists all the available commands in alphabetical order. Each command has a description, usage instructions and examples.

II.1. Related documentation

The aim of this manual is to provide reference information for all the user-relevant aspects of the HCSS system. Despite our efforts, you may find that some routines are missing, or have incomplete or inaccurate descriptions. In this case you are encouraged to consult the *Javadoc* developer's reference documentation. You can access the Javadoc by clicking on *HCSS Developer's Reference Manual (API)* in the table of contents of the HIPE Help System.

Guidance on how to use the Javadoc is provided in the *Scripting Guide*: Section 8.1 in *Scripting Guide*.

Some Javadoc pages may have links to more in-depth developer documentation. Be aware that these are *not* fully fledged help documents and are most useful to system developers or advanced users only.

Inspecting the source code

If you are an advanced user versed in Java and Jython, you might want to have a look at the source code of a task or class. You can include source code in your HIPE installation in the following ways:

- If you are installing a developer build via the Continuous Installation System, the `--src=yes` option will install source code. This is enabled by default if you use the `--developer` option.

  With the `--unpack=yes` option, the source code will be unpacked into a `src` subdirectory in your HIPE installation. With the `--unpack=no` option, source files for each module will be kept as ZIP files in the repository (usually under `$HOME/.hcss.d/repository` in UNIX systems and under `%HOMEPATH%/.hcss.d/repository` in Windows).

- If you are using an installer, select the checkbox next to the question *Would you like to have the source code installed?* This is only available if you choose the *Advanced* installation. The source code will be in the `src` subdirectory of your HIPE installation.

Dictionary

This section contains a list of definitions of words and acronyms you are likely to encounter in documentation on the Herschel satellite.

Dictionary

AAS	[Acronym] Altitude Anomaly Sensors
ABCL	[Acronym] As-Built Configuration List
ACA	[Acronym] Altitude Control Axis
ACC	[Acronym] Attitude Control Computer
ACM	[Acronym] Attitude Control and Measurement
ACMS	[Acronym] Attitude Control and Measurement System
ACS	[Acronym] Auto-Correlation Spectrometer
activateMasks	[Pipeline] PACS task to activate or deactivate specified masks before running a pipeline task
A/D	[Acronym] Analogue to Digital
AD	[Acronym] Applicable Document
ADC	[Acronym] Analogue to Digital Converter
addUTC	[Pipeline] PACS Level 0 task to convert from onboard time to UTC
ADP	[Acronym] Acceptance Data Package
AIT	[Acronym] Assembly, Integration and Test
AIV	[Acronym] Assembly, Integration, and Verification
AM	[Acronym] Avionics Model
AMA	[Acronym] Absolute Measurement Accuracy
AME	[Acronym] Attitude Measurement Error
AO	[Acronym] Announcement of Opportunity
AOCS	[Acronym] Attitude and Orbit Control System
AOR	[Acronym] Astronomical Observation Request
AOS	[Acronym] Acousto Optic Spectrometer
AOT	[Acronym] Astronomical Observing Template
APD	[Acronym] Absolute Pointing Drift
APE	[Acronym] Absolute Pointing Error
API	[Generic HIPE; Acronym] Application Programming Interface. It is an interface that a software program or library imple-

	ments in order to allow other software to interact with it. In the API documentation, details on how to use Java and Jython tasks and classes in HIPE are provided. There is an API section in most entries of the *User's Reference Manual*, listing task properties and class methods you can use from your scripts. The *Developer's Reference Manual* contains the API of the whole HCSS software, but it can be difficult to understand if you have no programming experience.
APID	[Acronym] Application Programmer IDentifier. The APID number is how the pipeline identifies spectrometers
AR	[Acronym] Acceptance Review
ARE	[Acronym] Absolute Rate Error
associateSkyPosition	[Pipeline] SPIRE photometry Level 0.5 task to add the pointing product to the timelines. Common to jiggle and scan map pipelines
argument	[I/O; Data types; Other] Or parameter, are the variables you send to a task when you call that task
attribute	[I/O; Data types; Other] Is a name for a specific field in a class. A field that denotes a particular characteristic of the class, much like a property of a class. For example, an important attribute of an observation context is the observation ID
auxiliary	[I/O; Data types; Other] Additional products containing information necessary to process raw Herschel data, but which the general user will never need to look at
BACUS	[Acronym] Bolometer Array Calibration Unit for SPIRE
BBID	[Generic HIPE; Acronym] Building block ID = bbtype*65536 + bbnumbers
bbnumber	[Generic HIPE] Building block number
BE	[Acronym] Back End
BEM	[Acronym] Back End Module (LFI)
BER	[Acronym] Bit Error Rate
block	[Generic HIPE] A limited stream of data, for example part of an observation, that are logically related to each other. See for instance *calibration block*.
BlockTable	[I/O; Data types; Other] PACS. A table showing the organisation of your observation data (your pipeline-processed or being processed product) into logical blocks, for example by raster position, wavelength range, nod position and so on.
boolean	[Generic HIPE] Can be True/False (1/0)
bps	[Acronym] bits per second
BW	[Acronym] Bandwidth
CA	[Acronym] Calibration Analysis (software)

CachedPool	[I/O; Data types; Other] A type of pool (see *pool*), containing cached products and meta data retrieved from remote pools.
calcBsmAngles	[Pipeline] SPIRE photometry Level 0.5 task to convert BSM telemetry in a Y angle and Z angle timeline. Common to jiggle and scan map pipelines
calcBsmFlags	[Pipeline] SPIRE photometry Level 0.5 task to convert BSM telemetry in a Y angle and Z angle timeline. Jiggle pipeline
calibration block	[Generic HIPE] The segment of your observation corresponding to when the internal calibration sources were observed.
calibration product	[Generic HIPE] A product from the calibration store
calibration store	[Generic HIPE] The store of calibration data, necessary to process your data and which comes with HIPE and with your HSA data
calibration tree	[Generic HIPE] The calibration information, held in the calibration store, necessary to reduce Herschel data. It comes with HIPE and with your HSA data and is necessary for pipeline processing
calstore	[Generic HIPE] Shorthand for the store of calibration data, which comes with HIPE and with your HSA data.
CAP	[Acronym] calibration analysis procedure (a script)
CASSIS	[I/O; Data types; Other] Spectral fitting software, also available as a HIPE plugin (http://cassis.cesr.fr).
CDD	[Acronym] Configuration Data Document
CEU	[Acronym] Cryo Electronics Unit
CheckDataStructure	[Pipeline] HIFI Level 0.5 task that checks that the datasets included in the HifiTimelineProduct have a unique bbnumber
CheckFreqGrid	[Pipeline] HIFI Level 0.5 task that provides information about the frequency ranges observed in the HTP and forms groups of datasets with common LO settings
CheckPhases	[Pipeline] HIFI Level 0.5 task that checks that the Chopper, LO frequency and buffer follow specific patterns prescribed by the observing modes
CLASS	[Acronym] software package for continuum and spectral line analysis.
class	[I/O; Data types; Other] The Java or Jython class of an object defines the type of object it is. In object-oriented programming a class is a construct that is used as a blueprint to create objects, so all objects of a particular class will have the same organisation and definition.
cleanPlateauFrames	[Pipeline] PACS Level 0 task to flag data points at the beginning and end of each chopper movement. Photometry. End of Level 0 pipeline processing
COA	[Acronym] Common Optics Assembly

Co-I	[Acronym] Co-Investigator
COM	[Acronym] Centre of Mass
console	[Generic HIPE] A HIPE view where you can write commands that are executed immediately.
constructor	[I/O; Data types; Other] In object-oriented programming a constructor in a class is a special block of statements that are called when an object is created.
Context	[I/O: Data types; Other] A Context is a special type of product that contains references to other products. Think of it as a bag of other products. This enables a means of building complex data structures. There are two ways in which these products can be organised in a Context: as a List, where they are accessed by index, or as a Map, where they are accessed as key—value pairs, similar to a Python dictionary.
convertChopper2Angle	[Pipeline] PACS Level 0 task to convert chopper instrument positions to sky angle
convertSignal2StandardCap	[Pipeline] PACS Level 0.5 task to convert all data down to the values they would have had if taken at the lowest capacitance setting. Subsequent tasks require this. Spectroscopy
ConvertFrequencyTask	[Pipeline] HIFI Level 1 pipeline task. Converts the IF frequency scale to the sideband frequencies defined by f_usb = f_LO + f_IF and f_lsb = f_LO - f_IF. At the same time, the units are changed from MHz to GHz. Can also be used to transform back to the IF or a velocity scale
COP	[Acronym] Commissioning Operations Plan or Commissioning phase (CoP)
corrBolTimeResponse	[Pipeline] SPIRE photometry Level 0.5 task to do the bolometer time response correction. Common to jiggle and scan map pipelines
CP	[Acronym] Calibration Pointing
CQM	[Acronym] Cryogenic Qualification Model
CRE	[Acronym] Cryogenic Readout Electronics (in some documents: Cold Readout Electronics)
createSpirePointing	[Pipeline] SPIRE photometry Level 0.5 task to run the Pointing Task to get the instrument pointing timeline. Common to jiggle and scan map pipelines
CRS	[Acronym] Coarse Rate Sensors
CS	[Acronym] (internal) calibration source
CSDT	[Acronym] Common Software Development Team
CST	[Acronym] Community Support Tools
Cube	[I/O; Data types; Other] a class of product at Level 1 to 2, of 3 dimensions, usually RA and Dec and time/wavelength or frequency

CUS	[Acronym] Common Uplink System
cutPhotDetTimelines	[Pipeline] SPIRE photometry scan map Level 0.5 task to cut the timeline back to scan line range. End of Level 0.5 pipeline
CVV	[Acronym] Cryostat Vacuum Vessel
D/A	[Acronym] Digital to Analogue.
DAC	[Acronym] Digital to Analogue Converter
DACS	[Acronym] Digital Autocorrelator Spectrometer
DAE	[Acronym] Data Acquisition Electronics (LFI)
DAPSAS	[Acronym] Data Processing and Science Analysis Software
DBMS	[Acronym] Database Management System
dBmW/MHz	[Acronym] Noise power level in dB relative to 1 mW in a 1 MHz bandwidth
DbPool	[I/O; Data types; Other] A type of pool (see pool) for accessing products from a remote database (e.g. the Versant database)
DBS	[Acronym] dual beam switch
DBU	[Acronym] Digital Bus Unit, the digital interface provided by the spacecraft
DC	[Acronym] Direct Current.
DC/DC	[Acronym] Direct Current voltage converter
DDS	[Acronym] Data Distribution System
deactivateMasks	[Pipeline] PACS task to deactivate specified masks before running a pipeline task
DEC	[Acronym] Detector Control
DECMEC	[Acronym] Detector and mechanism controller.
deglitchTimeline	[Pipeline] SPIRE photometry Level 0.5 task to do the deglitching for scan maps. Common to jiggle and scan map pipelines
demodulate	[Pipeline] SPIRE photometry Level 0.5 task to perform demodulation on jiggle pipeline data
denodding	[Pipeline] SPIRE photometry Level 0.5 task run to remove nodding from jiggle data. Jiggle pipeline
detectCalibrationBlock	[Pipeline] PACS Level 0 task to identify the calibration blocks and fill the CALSOURCE entry in the status table
Developer's Reference Manual	[Generic HIPE] The documentation describing the API (Application Programming Interface) of the HCSS software. Being aimed at developers, it can be difficult to understand if you have no programming experience. Also known as Javadoc.
DHS	[Acronym] Data Handling System
DHSS	[Acronym] Data Handling Subsystem

dictionary	[I/O; Data types; Other] A type of Jython data array made of key and value pairs.
dither	[I/O; Data types; Other] Moving sky position a small amount and taking another exposure
DMA	[Acronym] Direct Memory Access
dmchead	[Pipeline] PACS. The Dec Mec (detector mechanism) header. It contains the position and status of the instrument sampled at a high frequency. Values therein are in detector units.
DML	[Acronym] Declared Material List
DMS	[Acronym] Document Management System
DoAntennaTemp	[Pipeline] HIFI Level 1 task to translates to the antenna temperature scale
DoChannelWeights	[Pipeline] HIFI Level 0.5 task to compute the (channel-dependent) weights and fills them into the datasets of type 'science'
DoCleanUp	[Pipeline] HIFI Level 1 task to remove data that is no longer needed after running the Level1 pipeline
DOF	[Acronym] Degree of Freedom
DoFluxHotCold	[Pipeline] HIFI Level 0.5 task that applies the bandpass spectra to the science spectra for the intensity calibration. End of Level 1 pipeline
DoFreqGrid	[Pipeline] HIFI Level 1 task to compute a frequency scale which will be used for resampling
DoHkCheck	[Pipeline] HIFI Level 0.5 task that checks the House Keeping contained in a HifiCalibrationDataset
doHrsCorrSP	[Pipeline] HIFI Level 0 HRS task which corrects HRS spectra from IF non-linearity errors
doHrsCutBandEdges	[Pipeline] HIFI Level 0 HRS task which cuts the edges of the HRS sub-bands, according to the bandpass of the filter. End of HRS pipeline
doHrsFFT	[Pipeline] HIFI Level 0 HRS task which applies a FFT processing on the correlation functions of HRS
doHrsFreq	[Pipeline] HIFI Level 0 HRS task which computes the frequency of the sub-bands
doHrsNorm	[Pipeline] HIFI Level 0 HRS task which normalises the raw correlation functions of HRS
doHrsOffsetPow	[Pipeline] HIFI Level 0 HRS task which computes the offset and power of HRS
doHrsPowCorr	[Pipeline] HIFI Level 0 HRS task which corrects the non-Pipeline.linearity error of the power of HRS.
doHrsQDCFast	[Pipeline] HIFI Level 0 HRS task which corrects roughly the quantisation distortion of the correlation functions

doHrsQDCFull	[Pipeline] HIFI Level 0 HRS task which corrects the quantisation distortion of the correlation functions
doHrsSmooth	[Pipeline] HIFI Level 0 HRS task which applies a Hanning smoothing on the spectra
doHrsSubbands	[Pipeline] HIFI Level 0 HRS task which extracts the HRS subbands from the HRS readout
doHrsSymm	[Pipeline] HIFI Level 0 HRS task which symmetries the correlation function of HRS to prepare the FFT
doHrsWindow	[Pipeline] HIFI Level 0 HRS task which applies a Hanning or Hamming windowing to the correlation functions of HRS
DoOffSubtract	[Pipeline] HIFI Level 0.5 task to compute the differences between ON and OFF positions
DoPointingTask	[Pipeline] HIFI Level 0.5 task that applies the Pointing Product to the HTP to give position of the spectrum
DoRadialVelocity	[Pipeline] HIFI Level 1 task to correct the local oscillator frequency for the radial velocity of the spacecraft
DoRefSubtract	[Pipeline] HIFI Level 0.5 task that is used for the reference subtraction
DoSidebandGain	[Pipeline] HIFI Level 1 task that applies the sideband gain coefficients to the flux values
DoSpurs	[Pipeline] HIFI Level 1 task that removes spurious signals in the scans
DoSubStitch	[Pipeline] HIFI Level 1 task to adjust the subbands relative to each other, possibly removing the overlap
DoWbsBadPixels	[Pipeline] HIFI Level 0 WBS task that applies the masking pixel list
DoWbsDark	[Pipeline] HIFI Level 0 WBS task to subtract the dark values for all scans in a HifiSpectrumDataset
DoWbsFreq	[Pipeline] HIFI Level 0 WBS task that creates a frequency table for each scan
DoWbsNonLin	[Pipeline] HIFI Level 0 WBS task to perform nonlinearity correction for even and odd WBS pixels
DoWbsScanCount	[Pipeline] HIFI Level 0 WBS task to normalise the integration time of each scan to 10 milliseconds
DoWbsSubbands	[Pipeline] HIFI Level 0 WBS task that separates the output from CCDs into the four subbands for theWBS for all HifiSpectrumDatasets contained in the HifiTimelineProduct
DoWbsZero	[Pipeline] HIFI Level 0 WBS task that subtracts appropriate zero spectra for each time step
DP	[Acronym] Data Processing. Often used to indicate the data processing language used in HIPE, a combination of Jython and HIPE-specific commands and data types.

DPL	[Acronym] Declared Process List
DPOP	[Acronym] Daily Prime Operational Phase
DPU	[Acronym] Data Processing Unit
DRCU	[Acronym] Detector Readout and Control Unit
DRM	[Acronym] Developer's Reference Manual
DS	[Acronym] Digital Serial
DSB	[Acronym] Double Side Band
DSP	[Acronym] Digital Signal Processor
DTCP	[Acronym] Daily Telecommunications Period
EDAC	[Acronym] Error Detection And Correction
elecCrossCorrection	[Pipeline] SPIRE photometry Level 0.5 task to do the electrical crosstalk correction. Common to jiggle and scan map pipelines
editor	[Generic HIPE] A HIPE view in which you can load, write and run scripts. The graphical interfaces of tasks also open in the Editor view.
EOM	[Acronym] End of Mission
EOP	[Acronym] Early Orbit Phase
EP	[Acronym] Entrance Pupil
EPLM	[Acronym] Extended Payload Module
ESAC	[Acronym] European Space Astronomy Centre
ESD	[Acronym] Electrostatic Discharge
ESOC	[Acronym] European Space Operations Centre
ESTEC	[Acronym] European Space Technology and Research Centre
ESTRACK	[Acronym] European Space Tracking Station Network
EU	[Acronym] Electrical Unit
FCU	[Acronym] Focal Plane Control Unit (HIFI)
f/D	[Acronym] Ratio of focal length to Diameter
FEM	[Acronym] Front End Module (LFI)
FEU	[Acronym] Front End Unit (LFI)
FFT	[Acronym] Fast Fourier Transform
FH	[Acronym] Feed Horn (LFI)
filterSlew	[Pipeline] PACS Level 1 photometry task to remove the data taken during satellite slews
findBlocks	[Pipeline] PACS Level 0 task to organise your data into "blocks" according to an internal logical. Result is put in the BlockTable of your product

fitRamps	[Pipeline] PACS Level 0 task to fit the slope of the averaged or raw ramps, turning a Ramps product into a Frames product. Spectroscopy
flag	[Generic HIPE] A flag in the context of the pipeline is a layer in a data product where data points, or whole pixels, have been identified as bad or potentially bad. Also referred to as mask. See also *quality flag*.
flagChopMoveFrames	[Pipeline] PACS Level 0 task to mask individual data points that were taken while the chopper was moving. Spectroscopy
flagGratMove	[Pipeline] PACS Level 0 task to mask individual data point that were taken while the grating was moving. Spectroscopy
FM	[Acronym] Flight Model
FoV	[Acronym] Field of View
FP	[Acronym] Fabry-Perot
FPA	[Acronym] Focal Plane Assembly
FPU	[Acronym] Focal Plane Unit
FRD	[Acronym] Formatted Raw Data
Frames	[I/O; Data types; Other] Data Frames (or frames) is a generic word used for the datasets that comes from Herschel. Frames is also PACS class of data product that goes through the pipeline
FS	[Acronym] Flight Spare Model
FSS	[Acronym] Fine Sun Sensor
FTB	[Acronym] Focal Plane JFET and/or RF filter box (SPIRE)
FTP	[Acronym] File Transfer Protocol
FTS	[Acronym] Fourier Transform Spectrometer
FWHM	[Acronym] Full Width Half Maximum
GaAs	[Acronym] Gallium Arsenide Semiconductor
GeGa	[Acronym] Germanium Gallium Detector
getCalTree	[Pipeline] PACS task to get the calibration tree, necessary to do any pipeline processing
getObservation	[I/O; Data types; Other] Task to download one or more observations from the Herschel Science Archive, or to locate and load an observation from a local pool on your hard disk.
glitch	[Generic HIPE] A hiccup in the signal, due e.g. to a cosmic ray. Unlike for optical data, these glitches may affect the response of the immediately subsequent readouts
GO	[Acronym] Geostationary Orbit or Guest Observer
G/S	[Acronym] Ground Station
G/T	[Acronym] Gain to Temperature Ratio.

GT	[Acronym] Guaranteed Time
GTO	[Acronym] Geostationary Transfer Orbit or Guaranteed Time Observer
GUI	[Acronym] Graphical User Interface
GYR	[Acronym] Gyroscope
HAIO	[Acronym] Herschel Archive Interoperability
HBES	[Acronym] HIFI Back-End Sub-system
HBES-1	[Acronym] HBES IF processor
HBES-2	[Acronym] HBES high resolution spectrometer
HBES-3	[Acronym] HBES wide band spectrometer
HCNE	[Acronym] Herschel Confusion Noise Estimator
HCSS	[Acronym] Herschel Common Science System
HDB	[Acronym] HSA Data Base
HDD	[Acronym] HSA Data Distribution
HDP	[I/O; Data types; Other] Herschel Data Processing. Often used to indicate the data processing language used in HIPE, a combination of Jython and HIPE-specific commands and data types.
HDU	[Acronym] Header Data Unit (FITS)
HEB	[Acronym] Hot Electron Bolometer
HEMT	[Acronym] High Electron Mobility Transistor
HEO	[Acronym] Highly Eccentric Orbit
HET	[Acronym] Heterodyne instrument of the FIRST model payload
HFCU	[Acronym] HIFI Focal Plane Control Unit
HFPU	[Acronym] HIFI Focal Plane Unit
HGA	[Acronym] High-Gain Antenna
HICU	[Acronym] HIFI Instrument Control Unit
HIFI	[Acronym] Heterodyne Instrument for the Far Infrared
HifiPipelineTask	[Pipeline] Task to run all (or any) of the pipeline algorithms for all (or any) of the HIFI spectrometers
HIPE	[Acronym] Herschel Interactive Processing Environment
HK	[Acronym] House Keeping (data)
H/K	[Acronym] House Keeping (data)
HLCU	[Acronym] HIFI Local Oscillator Control Unit

HLOU	[Acronym] HIFI Local Oscillator Unit
HLS	[Acronym] High-Level Software (LFI)
HOB	[Acronym] Herschel Optical Bench
HOTAC	[Acronym] Herschel Observing Time Allocation Committee
housekeeping	[Generic HIPE] Data about what the satellite was doing and what its state was during your observation. Normally you will never need to look at the housekeeping data.
HPBW	[Acronym] Half-Power Beam Width
HPMCS	[Acronym] Herschel/Planck Mission Control System
HRI	[Acronym] High-Resolution IF processor (HIFI)
HRP	[Acronym] High Resolution IF Processor
HRS	[Acronym] High-Resolution Spectrometer (HIFI)
HSA	[Acronym] Herschel Science Archive
HsaReadPool	[I/O; Data types; Other] A type of pool (see pool), to access the Herschel Science Archive
HSC	[Acronym] Herschel Science Centre
HSIA	[Acronym] Hardware/Software Interaction Analysis
HSK	[Acronym] House Keeping
HTP	[Acronym] HifiTimelineProduct, a MapContext containing HifiSpectrumDatasets
HttpClientPool	[I/O; Data types; Other] A type of pool for accessing data on remote servers. See also *pool*.
HUI	[Acronym] HSA User Interface
H/W	[Acronym] Hardware
IA	[Acronym] Interactive Analysis
ICC	[Acronym] Instrument Control Centre
ID	[Acronym] Identification
I/F	[Acronym] Interface. Also IF: Intermediate Frequence
IF	[Acronym] Intermediate Frequency
IFP	[Acronym] Intermediate Frequency Processor
IFS	[Acronym] Integral field spectroscopy/spectrograph
IFSS	[Acronym] Intermediate Frequency and Spectrometer System
IFU	[Acronym] Integral Field Unit
ILT	[Acronym] Instrument Level Test
InP	[Acronym] Indium Phosphide semiconductor material

I/O	[Acronym] Input/Output
IPAC	[Acronym] (NASA/JPL) Infrared Processing and Analysis Center
IST	[Acronym] Instrument System Test
Javadoc	[Generic HIPE] A technology to generate developer documentation from *tags* embedded in Java source code. By extension, the developer documentation itself. In HIPE, also known as the *Developer's Reference Manual* (DRM). This documentation could be difficult to understand if you have no programming experience.
jiggleAverage	[Pipeline] SPIRE photometry Level 0.5 task average on the jiggle position for a number (1-4) of DemodPointingProducts. Jiggle pipeline
Label	[Generic HIPE] A bitword recorded with every PACS frame, indicating where we are in the synchronised mechanism movement sequence (chopper plateau, grating movement, and so on).
LAN	[Acronym] Local Area Network
Level 0	[I/O; Data types; Other] Herschel data can have up to 5 levels, depending on the amount of (mainly pipeline) processing that has been performed. There is a common convention for all Herschel instruments. Level 0 has had no pipeline processing done, and the data are organised in a time sequence (and perhaps other parameters)
Level 0.5	[I/O; Data types; Other] Herschel data can have up to 5 levels, depending on the amount of (mainly pipeline) processing that has been performed. There is a common convention for all Herschel instruments. Level 0.5 has had all basic pipeline processing done so the product can be inspected
Level 1	[I/O; Data types; Other] Herschel data can have up to 5 levels, depending on the amount of (mainly pipeline) processing that has been performed. There is a common convention for all Herschel instruments. Level 1 has had all instrument-correction type pipeline processing done and the data converted to physical units; they may now need to be inspected by the astronomer
Level 2	[I/O; Data types; Other] Herschel data can have up to 5 levels, depending on the amount of (mainly pipeline) processing that has been performed. There is a common convention for all Herschel instruments. Level 2 data should be of science quality
Level 2.5	[I/O; Data types; Other] Herschel data can have up to 5 levels, depending on the amount of (mainly pipeline) processing that has been performed. For PACS, level 2.5 products are photometric maps (`SimpleImage`) produced with MadMap, combining the scan and cross-scan AORs. For SPIRE, level 2.5 products are combined maps of any overlapping areas from the same proposal using scan mode AOTs. In contrast to PACS, the "naive" mapper is used rather than MadMap.
Level 3	[I/O; Data types; Other] Herschel data can have up to 5 levels, depending on the amount of (mainly pipeline) processing

	that has been performed. There is a common convention for all Herschel instruments. Level 3 data include publishable science products with level 2 data products as input. Possibly combined with theoretical models, other observations, laboratory data, catalogues, etc. Formats should be Virtual Observatory compatible.
LGA	[Acronym] Low-Gain Antenna
list	[I/O; Data types; Other] A type of Jython array.
list context	[I/O; Data types; Other] see *ListContext*.
ListContext	[I/O; Data types; Other] A type of Context which is a list of individual products (some of which can themselves be lists or other Contexts); see also *Context*.
LOBT	[Acronym] Local On-Board Time
local pool	[I/O; Data types; Other] The local pool (by default in *home/*`.hcss/lstore`) is a database in the form of a directory structure which contains data products in the form of FITS files. These files are organised in subdirectories of the local pool. See also *storage*, *pool*, *PAL*.
LocalPool	[I/O; Data types; Other] PACS. A way to define where you want your pool to be so you can save a product to it
local store	[I/O; Data types; Other] Name previously used to indicate a *local pool*.
LOS	[Acronym] Line of Sight
LOU	[Acronym] Local Oscillator Unit
lpfResponseCorrection	[Pipeline] SPIRE photometry Level 0.5 task to do the electrical Low Pass Filter response correction. Common to jiggle and scan map pipelines
LSB	[Acronym] Lower Side Band
LSU	[Acronym] Local Oscillator Synthesiser Unit (HIFI)
LTA	[Acronym] Long Term Archive
LWU	[Acronym] Local Oscillator Waveguide Unit (HIFI)
M3	[Acronym] Third Mirror in the optical train starting with the telescope primary
makeTodArray	[Pipeline] PACS Level 1 photometry task to build a time-ordered data stream for input into MadMap and add to the meta header for the output skymap
MapContext	[I/O; Data types; Other] A MapContext is a Context for products grouped into containers with access to each by a key (similar to a python dictionary). See also Context
mask	[Generic HIPE] A layer in a data product where data points, or whole pixels, have been identified as (potentially) bad. Not the same as quality flags

MCC	[Acronym] Mission Control Centre
M-CDR	[Acronym] Mission Level CDR
MCM	[Acronym] Multi Chip Module
MEM	[Acronym] Maximum Entropy Method
Meta data	[I/O; Data types; Other] Information about the data held in a data structure, from array datasets to observation contexts (entire observations). This information is held in the Meta header, which is conceptually the same as a FITS header.
Meta header	[I/O; Data types; Other] Part of a data structure (array dataset, table dataset, product, product context) where the *Meta data* are held.
method	[I/O; Data types; Other] In object-oriented programming a method is a group of (software) instructions that is given a unique name and can be called up at any point by simply quoting the name. In other languages a method is called a function, subroutine or procedure. Example: > myTask.getSomething(); the getSomething() is a method for myTask and it returns an answer that depends on what is (or is not) in the ()
MIP	[Acronym] Mission Implementation Plan
MkFluxHotCold	[Pipeline] HIFI Level 0.5 task to compute the bandpass and receiver temperature for the intensity calibration of the spectra
MkFreqGrid	[Pipeline] HIFI Level 1 pipeline task. At the end of the generic pipeline, the flux of all the resulting scans are resampled to a common frequency grid which this tasks computes
MkOffSmooth	[Pipeline] HIFI Level 0.5 task to calibrate a baseline by processing the measurements of an OFF position in sky
MkSidebandGain	[Pipeline] HIFI Level 1 task that prepares a helper object that provides the means to compute the IF- and LO- Pipeline.frequency dependent sideband gains ratios
MkSpur	[Pipeline] HIFI Level 0 WBS task to identify spurs in WBS HifiTimelineProducts. End of WBS pipeline
MkWbsBadPixels	[Pipeline] HIFI Level 0 WBS task that checks for saturated pixels
MkWbsFluxAtten	[Pipeline] HIFI Level 0 WBS task that analyses the attenuator settings
MkWbsFreq	[Pipeline] HIFI Level 0 WBS task that derives frequency scale from comb spectra
MkWbsZero	[Pipeline] HIFI Level 0 WBS task that checks the zeros and compute an interpolation function to represent zeros for each time
MLI	[Acronym] Multi-Layer Insulation
MOC	[Acronym] Mission Operations Centre
MoI	[Acronym] Moment of Inertia

MOIS	[Acronym] Mission Operations Information System
NAIF	[Acronym] Navigation Ancillary Information Facility
naiveApppMapper	[Pipeline] Level 1 SPIRE basic mapper for jiggle maps. End of Level 1 pipeline
naiveScanMapper	[Pipeline] Level 1 SPIRE task for basic map creation for scan maps. Scan map pipeline
namespace	[Generic HIPE] An abstract container grouping related items, for example HIPE tasks and classes. In Java, namespaces are rperesented by *packages*. Namespaces help to avoid name clashes: for example, Java classes with the same name can exist in different packages.
NaN	[Acronym] Not a Number
navigator	[Generic HIPE] A HIPE view in which you can navigate through the directories of your hard disk and open some file types into HIPE.
NDIU	[Acronym] Network Data Interface Unit
NEFD	[Acronym] Noise Equivalent Flux Density
NEP	[Acronym] Noise Equivalent Power
NFS	[Acronym] Network File System
NHSC	[Acronym] NASA Herschel Science Centre
nodAverage	[Pipeline] SPIRE photometry Level 0.5 task averaging denodded jiggle data, producing AveragedPhotPointingProduct. Jiggle pipeline
OBCP	[Acronym] On Board Control Procedure
OBDH	[Acronym] On-board Data Handling (System)
OBS	[Acronym] On-board Software
observation context	[I/O; Data types; Other] A container for all the data of a Herschel observation. It contains at least the Level 0 data, and all levels above that have been pipeline-processed by the HSA. It is a wrapper for everything you need to process and understand your observation. The observation context is a type of map context. The observation context can contain data of Levels 0 to 2 and usually includes other contexts, such as the auxiliary and calibration context.
ObservationContext	[I/O; Data types; Other] A container for all the data of a Herschel observation. It contains at least the Level 0 data, and all levels above that have been pipeline-processed by the HSA. It is a wrapper for everything you need to process and understand your observation. The observation context is a type of map context. The observation context can contain data of Levels 0 to 2 and usually includes other contexts, such as the auxiliary and calibration context.
obsid	[I/O; Data types; Other] Observation identifier. A number that uniquely identifies a Herschel observation.

OBSM	[Acronym] On-board Software Monitoring System
OBSW	[Acronym] On-board Software
OBT	[Acronym] On-Board Time
OD	[Acronym] Operational Day
ODB	[Acronym] Operational Database
ODD	[Acronym] On-demand Processed Data
ODP	[Acronym] On-demand Processing
ODR	[Acronym] On-demand Processing Report
OFD	[Acronym] On the Fly processed Data or Operations Facilities Document
OFP	[Acronym] On the Fly Processing
OFR	[Acronym] On the Fly Processing Report
OGSE	[Acronym] Optical Ground Support Equipment
OIRD	[Acronym] Operations Interface Requirements Document
operational day	[I/O; Data types; Other] the Herschel day unit
OTAC	[Acronym] Observing Time Allocation Committee
OTF	[Acronym] On-Target Flag
OTF	[Acronym] On the Fly
OU	[Acronym] Optics Unit
outline	[Generic HIPE] A HIPE view where you can see an outline of any selected variable in the *Variables* view.
package	[Generic HIPE] A package is a container for organising tasks, functions and tools that have been written for you to use in HIPE. A package must be *imported* into HIPE for these features to be available. The most common packages are imported automatically when HIPE starts.
packet sequence	[Generic HIPE] A sequence of packets of raw Herschel data. Telemetry files are made of packet sequences.
PACS	[Acronym] Photodetector Array Camera and Spectrometer
PAL	[Acronym] Product Access Layer
PDB	[Acronym] Proposal Data Base
PDE	[Acronym] Pointing Drift Error
PER	[Acronym] Proposal Evaluation Report
perspective	[Generic HIPE] A group of HIPE views organised in a specific way inside the HIPE main window. Each perspective groups

	views needed for a high-level task, for example data analysisor data retrieval from the Herschel Science Archive. See also *view*.
photAddInstantPointing	[Pipeline] PACS Level 0 task to do the first step of the astrometric calibration, associating raster point counter, nod counter or sky line scan counter to the individual frames within an observation. Photometry
photAssignRaDec	[Pipeline] PACS Level 0.5 task to assign RA and Dec to every pixel in the image. Photometry
photAvgDith	[Pipeline] PACS Level 0.5 task to mean-combine the chops, after photDiffChop has been run. Photometry
photAvgNod	[Pipeline] PACS Level 0.5 photometry task to average nod signals per nod cycle
photAvgPlateau	[Pipeline] PACS Level 0.5 task to average all signals on a chopper plateau (chopper not moving time). Photometry
photCombineNod	[Pipeline] PACS Level 0.5 task to combine repeated nod cycles. Photometry
photConvDigit2Volts	[Pipeline] PACS Level 0 task to convert units to Volts. Photometry
photCorrectCrosstalk	[Pipeline] PACS Level 0 task to correct for electronic crosstalk. Photometry
photDiffCal	[Pipeline] PACS Level 0 task to subtract the off and on calibration block data and average the results. Various statistics are calculated. Photometry
photDiffChop	[Pipeline] PACS Level 0.5 task to subtract the off- from the corresponding on-.chops. Photometry
photDiffNod	[Pipeline] PACS Level 0.5 task to subtract the A and B nods, per cycle. Photometry
photDriftCorrection	[Pipeline] PACS Level 0.5 task to apply a correction for drift in the flatfield over the observation. Photometry. End of Level 0.5 pipeline processing
photFlagBadPixels	[Pipeline] PACS Level 0 task to mask pixels that have been pre-identified by the PACS team as bad. Photometry
photFlagSaturation	[Pipeline] PACS Level 0 task to mask individual data points for being saturated. Photometry
photFluxConversion	[Pipeline] SPIRE photometry Level 0.5 task to do the flux conversion. Common to jiggle and scan map pipelines
photGetPointings4PointSource	[Pipeline] PACS Level 0.5 task to get the pointings for the virtual aperture; part of the astrometric calibration. Photometry
photHighPassFilter	[Pipeline] PACS Level 1 photometry task to remove the 1/f noise
photMakeDithPos	[Pipeline] PACS Level 0.5 task to check whether a dither pattern exists, and if so to identify the positions. Photometry

photMakeRasPosCount	[Pipeline] PACS Level 0.5 task to add raster position to the status table. Photometry
photMMTDeglitching	[Pipeline] PACS Level 0 task to detect, mask and remove the effects of cosmic rays (glitches). Photometry
photOptCrossCorrection	[Pipeline] SPIRE photometry Level 0.5 task to do the optical crosstalk correction. Common to jiggle and scan map pipelines
photProject	[Pipeline] PACS Level 1 photometry task to create a map out of images, by adding the images. End of Level 1 pipeline
photProjectPointSource	[Pipeline] PACS Level 1 photometry task to create an image covering the total footprint of a point source of an observation. End of Level 1 pipeline processing
photRespFlatFieldCorrection	[Pipeline] PACS Level 0.5 task to apply the flatfield correction and convert signal to a flux density. Photometry
photShiftDith	[Pipeline] PACS Level 1 photometry task to shift the dither offsets of an observation and then add the images
PHS	[Acronym] Proposal Handling Subsystem
PI	[Acronym] Principal Investigator
P/L	[Acronym] Payload
PLM	[Acronym] Payload Module
POF	[Acronym] Planned Observations File
pointing product	[Pipeline] The product that contains the spacecraft pointing over the entire observation.
pool	[I/O; Data types; Other] A pool is a Herschel database on your disc, on an external disc, or online. By default pools are subdirectories off of your local store ([your home directory]/.hcss/lstore). Data therein are in FITS format. In a pool you will usually place data that are related in some way. There are several types of pools, the general user will be most used to ProductPools. See also local store, storage, PAL
POS	[Acronym] Planned Observation Sequence
product	[I/O; Data types; Other] Herschel data are carried about in structures called products, which are like software onions: they are made up of layers, each of which has its own description and history of the contents, and on each of which specific actions can be taken. Peel the layers to get at the actual data, be that one number or arrays of numbers. Generally Herschel data products include meta data keywords, tables wth actual data and the history of the generation of the product
product access layer	[I/O; Data types; Other] PAL. This is a manager to allow you to set up, explore and get data from storage areas. The PAL is also a perspective of HIPE
ProductRef	[I/O; Data types; Other] A reference to a product. Not the actual product but a "link" to it. You can have more than one product "stored" in a ProductRef

product reference	[I/O; Data types; Other] A reference to a product. Not the actual product but a "link" to it. You can have more than one product "stored" in a ProductRef
ProductPool	[I/O; Data types; Other] A type of pool (see pool), containing products
product storage	[I/O; Data types; Other] See storage, because product storage is a "storage" of products. ProductStorage is the name of the java class you use to define a storage
ProductStorage	[I/O; Data types; Other] See storage, because product storage is a "storage" of products, and ProductStorage is the name of the task you can use to define the storage
product viewer	[Generic HIPE] Viewers for looking at products. You can access them from the command line or by clicking on a product in the variables panel/editor panel
PSF	[Acronym] Planning Skeleton File or Point Spread Function
PT	[Acronym] Product Tree
PV	[Acronym] Performance Verification
PVOP	[Acronym] Performance Verification Operations Plan
QA	[Acronym] Quality Assurance
QAR	[Acronym] Quality Assessment Report
QCP	[Acronym] Quality Control Pipeline
QLA	[Acronym] Quick Look Analysis
QM	[Acronym] Qualification Model
QPD	[Acronym] Quality Processed Data
QTR	[Acronym] Qualification Test Review
quality flag	[Generic HIPE] An indication of how well a pipeline task ran
Ramps	[I/O; Data types; Other] A PACS class of product used in the pipeline of Level 0 to 0.5. It has 4 dimensions
raster	[I/O; Data types; Other] Moving sky position by small increments over a larger field to observe a larger area in more dense detail
RCS	[Acronym] Reaction Control System
reference	[I/O; Data types; Other] A pointer to something. For example, when you create an array in HIPE "sey = [4,5,6]", then "sey" is a pointer to [4,5,6]. "sey" is not actually [4,5,6], it just references it
RF	[Acronym] Radio Frequency
RFI	[Acronym] Radio Frequency Interference
RFW	[Acronym] Request for Waiver

RMS	[Acronym] root-mean squared
RPE	[Acronym] Relative Pointing Error
rsrfCal	[Pipeline] PACS Level 0.5 task to correct a spectroscopy Frames product for the relative spectral response function. Absolute correct is done with specRespCal.
RSS	[Acronym] root-sum squared
runMadMap	[Pipeline] PACS Level 1 photometry task to run the java MadMap module. This module uses the maximum-likelihood technique to build a map from an input time-ordered data stream. End of Level 1 pipeline processing
RWA	[Acronym] Reaction Wheel Assembly
SA	[Acronym] Solar Array or Scientific Analysis
SAA	[Acronym] Solar Aspect Angle
SAMP	[Generic HIPE] The technology used by HIPE to interact with the Virtual Observatory (VO).
SAS	[Acronym] Sun Aquisition Sensors
saveObservation	[I/O; Data types; Other] PACS task to save your ObservationContext to a pool
S/C	[Acronym] Spacecraft
secondDeglitching	[Pipeline] SPIRE photometry Level 0.5 task to do a second level degliticing for jiggle pipeline
SED	[Acronym] Spectral Energy Distribution
SIN	[Acronym] Straylight Induced Noise
S/N	[Acronym] Signal to Noise Ratio
SOVT	[Acronym] System Operation Validation Test
spaxel	[Generic HIPE] A spatial pixel, i.e. the spatial unit of a cube
SPC	[Acronym] Science Programme Committee
specAddInstantPointing	[Pipeline] PACS Level 0 task to add the RA and Dec to the central spaxel. Spectroscopy
specAddNod	[Pipeline] PACS Level 0.5 task to add together corresponding nods. Spectroscopy
specAssignRaDec	[Pipeline] PACS Level 0 task to take the RA and Dec that have been assigned to the central spaxel (specAddInstantPointing) and calculate the pointing for all other spaxels. Spectroscopy
specConvDigits2VoltsPerSecFrames	[Pipeline] PACS Level 0 task to convert the units to Volts/s. For a spectroscopy Frames product
specConvDigits2VoltsRamps	[Pipeline] PACS Level 0 task to convert the units to Volts. For a spectroscopy Ramps product

specCorrectHerschelVelocity	[Pipeline] PACS Level 0 task to correct the wavelength grid for the velocity of Herschel during your observation. Spectroscopy
specCorrectSignalNonLinearities	[Pipeline] PACS Level 0.5 task to correct the Frames data points for the fact that the true slope of the raw ramps (that the frames came from) are not linear. Spectroscopy
specDiffChop	[Pipeline] PACS Level 0.5 task to subtract the off chop data from the corresponding on chop data. Spectroscopy
specDiffCs	[Pipeline] PACS Level 0.5 task to calculate the dark and response from the calibration block, for each pixel. Spectroscopy
specExtendStatus	[Pipeline] PACS Level 0 task to add information to the Status table, based on the previous pipeline processing that you have done. Spectroscopy
specFitSignalDrift	[Pipeline] PACS Level 0.5 task to calculate the drift in the response of the pixels during an observation. Spectroscopy
specFlagBadPixelsFrames	[Pipeline] PACS Level 0 task to add a mask to the bad pixels identified by the PACS team. Spectroscopy
specFlagGlitchFramesQTest	[Pipeline] PACS Level 0.5 task to add a mask to individual data points that are glitches. Spectroscopy
specFlagOutliers	[Pipeline] PACS Level 1 task to flag individual data points in a PacsCube for being outliers (most likely glitches). Spectroscopy
specFlagSaturationFrames	[Pipeline] PACS Level 0 task to mask individual data points for saturation. Works on a Frames product. Spectroscopy
specFlagSaturationRamps	[Pipeline] PACS Level 0 task to mask individual data points for saturation. Works on a Ramps product. Spectroscopy
specFrames2PacsCube	[Pipeline] PACS task to convert a spectroscopy Frames product to a PacsCube product. End of Level 0.5 pipeline processing
specProject	[Pipeline] PACS task to take the RebinnedCube and project each spaxel and each data point onto a regular RA/Dec grid on the sky. Spectroscopy and end of Level 2 pipeline processing
specRespCal	[Pipeline] PACS Level 0.5 task to correct the data to an absolute flux scale. Spectroscopy
Spectrum1d	[I/O; Data types; Other] Is a class of product that is for holding 1-dimensional spectral data
Spectrum2d	[I/O; Data types; Other] Is a class of product that is for holding 2-dimensional spectral data
spectrum container	[I/O; Data types; Other] SpectrumContainer is a class of product. A container of spectra
SpectrumContainer	[I/O; Data types; Other] Is a class of product. A container of spectra
specWaveRebin	[Pipeline] PACS Level 1 task to convert the 5x5 PacsCube with several spectra in each spaxel to a 5x5 RebinnedCube with one spectrum in each spaxel. Spectroscopy. End of Level 1 pipeline processing

SPF	[Acronym] Single Point Failure
SPG	[Acronym] Software/Software Product Generation
SPIRE	[Acronym] Spectral and Photometric Imaging Receiver
SpireListContext	[Pipeline] Level 1 SPIRE task to create a SpireListContext to be used as input for map making. Scan map pipeline
SPR	[Acronym] Software Problem Report
SPU	[Acronym] Signal Processing Unit
SREM	[Acronym] Standard Radiation Environment Monitor
SRPE	[Acronym] Spatial Relative Pointing Error
S/S	[Acronym] Subsystem
SSB	[Acronym] Single Side Band
SSO	[Acronym] Solar System Object
SSR	[Acronym] Solid State Recorders
Status	[I/O; Data types; Other] PACS. Information about the instrument status, held in tabular format and added to as you pipeline process your observation
storage	[I/O; Data types; Other] Consider a storage a database storing data in an structure that Herschel observations require. A storage is the variable defined in HIPE pointing to where the data is on disc, rather than the name of the directory on disc that actually contains the data. See also pool, local store, PAL
STR	[Acronym] Star Trackers
SUM	[Acronym] Satellite User Manual
Summary Table	[I/O; Data types; Other] HIFI. a table for each spectrometer at each level in the ObservationContext, detailing the components of the observation
SVM	[Acronym] Service Module
SVT	[Acronym] System Validation Test
S/W	[Acronym] Software
TA	[Acronym] Trend Analysis (software) or Telescope Assembly
TableDataset	[I/O; Data types; Other] A dataset held in a of tabular structure. Is also a class of product
task	[Generic HIPE] A self-contained HIPE script that does something to the input data. Also a panel of HIPE in which tasks are listed
TC	[Acronym] Telecommand
temperatureDriftCorrection	[Pipeline] SPIRE photometry Level 0.5 task to do the temperature drift correction. Common to jiggle and scan map pipelines

TM	[Acronym] Telemetry. A TM file contains telemetry data, and has a tm suffix
ToO	[Acronym] Target of Opportunity
toolbox	[Generic HIPE] A set of routines for use in data processing and which you can use if writing your own scripts in HIPE
tuple	[I/O; Data types; Other] A type of data array (jython)
UM	[Acronym] User Manual
URD	[Acronym] User Requirements Document
URM	[Acronym] User Reference Manual
URN	[Acronym] Uniform Resource Name (a string identifier to uniquely identify any product stored in any pool)
URR	[Acronym] User Requirements Review
USB	[Acronym] Upper Side Band
useRemoveBaseline	[Pipeline] Level 1 SPIRE task to set a boolean Flag to switch on and off the baseline removal. Scan map pipeline
variable	[Generic HIPE] A variable is a label representing some data. Such data can be a number, a string of text, a Herschel product or even an entire observation. Each variable has a *type* describing the kind of data it refers to. If you write `myNumber = 12` in the *Console* view of HIPE and press **Enter**, you create a variable whose name is `myNumber`, whose type is `Integer` and whose value is 12. A command like `myObs = getObservation (obsid=123456)` creates a variable whose name is `myObs`, whose type is `ObservationContext` and whose value is the actual data of the observation. Variables in your HIPE session are listed in the Variables view.
VC	[Acronym] Virtual Channel
view	[Generic HIPE] The individual panels you see in the HIPE GUI are "views". These each allow you to view and thus search through something, or allow you to type commands into HIPE. The main toolbar lists all views you can have
VO	[Acronym] Virtual Observatory
waveCalc	[Pipeline] PACS Level 0 task to calculate the wavelength grid from the grating positions for an observation. Spectroscopy.
wavelengthGrid	[Pipeline] PACS Level 1 task to calculate the wavelength grid of a PacsCube. Some optimisation possible. Spectroscopy
WBC	[Acronym] Wide-Band Spectrometer Control (HIFI))
WBI	[Acronym] Wide-Band Spectrometer IF processor (HIFI)
WBO	[Acronym] Wide-Band Spectrometer AOS (HIFI)

WBS	[Acronym] Wide-Band Spectrometer
WBSPU	[Acronym] Wide Bandwidth Spectrometer Processing Unit
WCS	[Acronym] world coordinate system
WFE	[Acronym] WaveFront Error
WG	[Acronym] Wave Guide
workbench	[Generic HIPE] The main GUI of HIPE in where you will work. Full workbench has more panels that the workbench, but all panels can also be accessed from the main toolbar
XML	[Acronym] Extensible Markup Language

Chapter 1. DP Commands

1.1. Introduction

This chapter is a listing of built-in HIPE functions, task and objects, collectively referred to as commands.

How to use this chapter

A first section gives a general overview what a command is used for and how to use it. This includes a general description and some examples. The "API summary" section provides a quick overview of the available constructors, methods and properties of a command. It is intended as quick reference for users that just need to remember a certain functionality. The "API details" section provides more detailed information about the constructors, methods and properties. Furthermore it provides links to other references and information about the command's history.

Overview

A table gives the full name of the command, indicates if it is written in Java or Jython and if it is just an ordinary command or if it follows the HCSS Task specifications. The import statement in this table can be copied to a Jython console.

Description

General description of the command.

Examples

Most commands include one or more examples that demonstrates how the command is used. Most of the examples are just one or two lines of DP code that can be entered at the Jython prompt. Others are code fragments or routines designed to serve as an examples for your own programs.

Limitations and Miscellaneous

The "Limitations" and "Miscellaneous" sections describes the boundaries of applicability and other specialties of the command.

API Summary

The API Summary provides a quick summary of the commands constructor, method, and properties. Note that most of the arguments are positional parameters that must be supplied in the order indicated by the command's syntax. However arguments can be often supplied by specifying their names. If its the case then they can provided in any order.

Constructor

Constructors are used to create a command.

Method

Methods are functions of a command that can be executed.

Properties

Properties are fields of a command that can be read or written by a command. INPUT properties contain values that are required by a command, OUTPUT properties contain the result of a command, INOUT provide both functionalities at the same time.

See also

The "See also" section provides links to related commands, to other manuals, or to external resources in the Internet.

History

The "History" section describes the changes occurred to the command.

1.2. Aberration

Full Name:	herschel.ia.obs.auxiliary.pointing.Aberration
Alias:	Aberration
Type:	Java Class -
Import:	from herschel.ia.obs.auxiliary.pointing import Aberration
Category	Astronomical utilities

Description

Obtain astrometric and geometric positions from apparent ones.

Definitions

- The **geometric** position is where the target was when the light reached the observer. This is what you get from SSO ephemeris data if you apply no corrections.
- The **astrometric** position is where the target was when the light left the body. It corrects for the amount the target has moved in the time it takes the light to reach you. Light-travel time correction corrects the geometric positions to astrometric ones. This is where the body would appear to be if the observer were at rest.
- The **apparent** position is the where the target appears to be (where you have to point the telescope). Aberration correction corrects the astrometric positions to apparent ones. This is due to motion of the observer.

Inputs details

- **PointingArray** : Pointings as provided from a PointingProduct retrieved from the archive contain **ACA** coordinates. These are apparent positions displaced by a SIAM offset (due to the relative position of the receptors wrt the X-axis of the SC). To obtain astrometric and geometric coordinates, the PointingArrays used as input to these functions require that you previously decorrect the SIAM offsets to obtain proper apparent coordinates.
- **FineTime[]** : The FineTime arrays should provide TAI.

Examples

Example 1: Detailed example obtaining one astrometric coordinate from an observation

```
# obs contains an observation
t2t = obs.auxiliary.timeCorrelation
ppIterator = pointingProd.iterator(pointingProd.getStartTime(),
 pointingProd.getEndTime())
item = ppIterator.next() #item.isOnTarget() should be true
# Siam data
siamDir = Configuration.getProperty("hcss.pointing.siamdir")
siamFilePath= siamDir + str(pointingProd.getSiamId()) +".SIAM"
siam = SiamProduct(siamFilePath);
#Recovering apparent pointing
aperture = item.getPointingId().getAperture()
qT_SIAM =
 siam.getSiamSet(aperture).getMatrix3().copy().mTranspose().toQuaternion()
q_aper = item.getFiltered().getQuaternion().multiply(qT_SIAM)
p = Pointing(Attitude(q_aper))
#Build the the TAIs (ts)
ts = [t2t.toFineTime(item.getObt())]
# Build the pointing array (ps)
ps = PointingArray(1)
ps.addPointing(0, p)
# Get the astrometric pointing(s)
```

Example 1: Detailed example obtaining one astrometric coordinate from an observation

```
pa = Aberration.correct(ps, ts, ephem)
```

Example 2: Detailed example obtaining one geometric coordinate from an SSO observation

```
naifid = Integer.parseInt(obs.getMeta()["naifId"].value)
ppIterator = pointingProd.iterator(pointingProd.getStartTime(),
 pointingProd.getEndTime())
item = ppIterator.next() #item.isOnTarget() should be true
# Siam data
siamDir = Configuration.getProperty("hcss.pointing.siamdir")
siamFilePath= siamDir + str(pointingProd.getSiamId()) +".SIAM"
siam = SiamProduct(siamFilePath);
#Recovering apparent pointing
aperture = item.getPointingId().getAperture()
qT_SIAM =
 siam.getSiamSet(aperture).getMatrix3().copy().mTranspose().toQuaternion()
q_aper = item.getFiltered().getQuaternion().multiply(qT_SIAM)
p = Pointing(Attitude(q_aper))
#Build the the TAIs (ts)
ts = [t2t.toFineTime(item.getObt())]
# Build the pointing array (ps)
ps = PointingArray(1)
ps.addPointing(0, p)
# Get the geometric pointing(s)
pa = Aberration.correctAndLTT(ps, ts, naifid, horizons)
print pa.getPointing(0)
```

API Summary

Methods
PointingArray **correct** (PointingArray ps, FineTime[] ts) Obtain astrometric positions: naive aberration decorrection.
PointingArray **correct** (PointingArray ps, FineTime[] ts, OrbitEphemerisProduct e) Obtain astrometric positions using an Earth model.
PointingArray **correct** (PointingArray ps, FineTime[] ts, Ephem e) Obtain astrometric positions: MPS aberration decorrection.
PointingArray **correctAndLTT** (PointingArray ps, FineTime[] ts, int naifid, basicHorizons h) Obtain geometric positions of SSOs: MPS aberration and LTT decorrections for SSOs.

Limitations

- To obtain an Horizons object you need an Ephemeris object and a directory with SSO files.
- the functions are list-array based so that the calculations are done in a tight Java loop
- When using PointingProduct's pointings, the PointingArrays used as input to these functions require that you previously decorrect the SIAM offsets to obtain proper apparent coordinates.
- The FineTime arrays provided as input should contain TAI.
- For further information consult the following classes in the HCSS Developer's Reference Manual (Javadoc API):
  - herschel.ia.obs.auxiliary.orbitephem.OrbitEphemerisProduct
  - herschel.share.fltdyn.ephem.Ephem

- herschel.share.fltdyn.ephem.Ephemerides
- herschel.share.fltdyn.ephem.horizons.BasicHorizons
- herschel.share.fltdyn.math.Attitude
- herschel.share.fltdyn.math.Direction
- herschel.share.fltdyn.math.Quaternion
- herschel.share.fltdyn.math.Vector3
- herschel.share.fltdyn.time.FineTime
- herschel.share.fltdyn.time.Mjd2000TimeFormat
- herschel.share.fltdyn.time.TimeScale

API Details

Methods

`PointingArray correct (PointingArray ps, FineTime[] ts)`

Obtain astrometric positions: naive aberration decorrection.

This method provides a first-approximation correction for aberration, using an Earth orbit model and neglecting the S/C movement. While very simplistic, it has been checked that the max. difference w.r.t. full correction is 1 arcsec and the average difference is 0.65 arcsec measured over the whole sky for a two-year period. **Use only if you have no access to products.** If you have a processed observation, it is better to use:

- For the astrometric position: Use **PointingArray correct(PointingArray ps, FineTime[] ts, Ephem e)**
- For the geometric position of SSOs: Use **PointingArray correctAndLTT(PointingArray ps, FineTime[] ts, int naifid, BasicHorizons h)**

Arguments

`PointingArray` **`ps`** `[INPUT, MANDATORY, default=no default value]`

Pointings with apparent positions.

`FineTime[]` **`ts`** `[INPUT, MANDATORY, default=no default value]`

Array with sequence of TAI (each ts[i] corresponds to a ps[i]).

Return

PointingArray

pointings to the astrometric positions

`PointingArray correct (PointingArray ps, FineTime[] ts, OrbitEphemerisProduct e)`

Obtain astrometric positions using an Earth model.

The angular error wrt MPS data (arcsec) is: MAX = 0.000349, AVG = 0.000132 If you have a processed observation, it is better to use:

- For the astrometric position: Use **PointingArray correct(PointingArray ps, FineTime[] ts, Ephem e)**

PointingArray correct (PointingArray ps, FineTime[] ts, OrbitEphemerisProduct e)

- For the geometric position of SSOs: Use **PointingArray correctAndLTT (PointingArray ps, FineTime[] ts, int naifid, BasicHorizons h)**

Arguments

`PointingArray` **`ps`** `[INPUT, MANDATORY, default=no default value]`

Pointings to be aberration-decorrected.

`FineTime[]` **`ts`** `[INPUT, MANDATORY, default=no default value]`

Array with sequence of TAI (each ts[i] corresponds to a ps[i]).

`OrbitEphemerisProduct` **`e`** `[INPUT, MANDATORY, default=no default value]`

Orbit ephemeris product with S/C data for ts (via interpolation).

Return

PointingArray

pointings to the astrometric positions, throws IllegalArgumentException if OrbitEphemerisProduct has no data for a ts

PointingArray correct (PointingArray ps, FineTime[] ts, Ephem e)

Obtain astrometric positions: MPS aberration decorrection.

Note: For both SSO and non-SSO. Requires S/C Ephemerides.

This is as good as mission planning data (from SPICE STELAB)

Arguments

`PointingArray` **`ps`** `[INPUT, MANDATORY, default=no default value]`

Pointings to be aberration-decorrected.

`FineTime[]` **`ts`** `[INPUT, MANDATORY, default=no default value]`

Array with sequence of TAI (each ts[i] corresponds to a ps[i]).

`Ephem` **`e`** `[INPUT, MANDATORY, default=no default value]`

Data to get the S/C velocity.

Return

PointingArray

pointings to the astrometric positions

PointingArray correctAndLTT (PointingArray ps, FineTime[] ts, int naifid, basicHorizons h)

Obtain geometric positions of SSOs: MPS aberration and LTT decorrections for SSOs.

Note: This is only for SSO objects and requires Horizons.

This is almost as good as mission planning data (from SPICE STELAB)

Arguments

`PointingArray` **`ps`** `[INPUT, MANDATORY, default=no default value]`

Pointings to be aberration-decorrected.

`FineTime[]` **`ts`** `[INPUT, MANDATORY, default=no default value]`

Array with sequence of TAI (each ts[i] corresponds to a ps[i]).

`int` **`naifid`** `[INPUT, MANDATORY, default=no default value]`

PointingArray **correctAndLTT (PointingArray ps, FineTime[] ts, int naifid, basicHorizons h)**

NaifId of the observed body.

`basicHorizons` **`h`** `[INPUT, MANDATORY, default=no default value]`

Horizons data to decorrect the pointing.

Return

PointingArray

pointings to the geometric positions

See also

- Developers Manual: herschel.ia.obs.auxiliary.pointing.Aberration

History

- `2011-01-18 - JDS`: First publication with jtags
- `2011-01-24 - JDS`: Clarifications about astrometric and geometric coordinates (no jparams yet)
- `2011-09-08 - JDS`: Extensive rewriting of documentation
- `2012-02-28 - JDS`: Examples use TimeCorr.toFineTime

1.3. ABS

Full Name:	herschel.ia.numeric.toolbox.basic.Abs
Alias:	ABS
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Abs
Category	Mathematics/General functions

Description

Returns the absolute value of a number or a numeric array. For complex numbers, ABS(x) returns the modulus.

Example

Example 1: Applying ABS on numbers and arrays

```
x = Int1d( [-1,0,1] )
print ABS(-123), ABS(x) # 123 [1,0,1]
```

API Summary

Jython Syntax

```
&lt;y&gt;=ABS(&lt;x&gt;)
```

Properties

Number or array **x** [INPUT, MANDATORY, default=no default value]

Number or array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Number or array x [INPUT, MANDATORY, default=no default value]

The number or array of which to compute the absolute value, or modulus for complex numbers.

Number or array y [OUTPUT, MANDATORY, default=no default value]

The absolute value of the input, or the modulus for complex numbers.

See also

- Developers Manual: herschel.ia.numeric.toolbox.basic.Abs

1.4. AbstractBasicModel

Full Name:	herschel.ia.numeric.toolbox.fit.AbstractBasicModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import AbstractBasicModel
Category	Mathematics/Fitting

Description

AbstractBasicModel implements the common parts of Models and in particular those of simple or basic Models.

A model consists of a concrete instantiation of the ModelFunction interface, together with its parameters. Optionally also prior ranges, fitindices and limits can be stored.

Each model contains a PriorList, an ArrayList of AbstractPriors. In principle each parameter of a model has its own prior. The prior contains information about the parameters which is known beforehand, like the limits between which the parameter is to be found. In most simple cases the priors are UniformPriors with optional limits. The full use of priors is reserved for Bayesian calculations as in NestedSampler.

AbstractBasicModel also implements a numerical derivation of partial to be used when partial is not given in the model definition itself.

An introduction to the use of fit package can be found here.

At least once have a look at the background on the organization and structure of the package. The fit/demo directory contains worked examples on almost all aspects of of the package.

See also

- PriorList
- UniformPrior
- NestedSampler
- Developers Manual: herschel.ia.numeric.toolbox.fit.AbstractBasicModel

1.5. AbstractFitter

Full Name:	herschel.ia.numeric.toolbox.fit.AbstractFitter
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import AbstractFitter
Category	Mathematics/Fitting

Description

Fitter for linear models.

The Fitter class is to be used in conjunction with *Model classes.

The Fitter class and its descendants fit data to a model. Fitter itself is the variant for linear models, ie. models linear in its parameters.

For both linear and nonlinear models it holds that once the optimal estimate of the parameters is found, a variety of calculations is exactly the same: standard deviations, noise scale, evidence and model errors. They all derive more or less from the inverse Hessian matrix (aka the covariance matrix). All these calculations are in this Fitter class. Other Fitter classes relegate their calculation in these issues to this one.

An introduction to the use of fit package can be found here.

At least once have a look at the background on the organization and structure of the package. The fit/demo directory contains worked examples on almost all aspects of of the package.

Example

Example 1: Fitter (for models which are linear in its parameters.)

```
# assume x and y are Double1d data arrays:
x = Double1d.range(100)
y = Double1d( Int1d.range(100).divide( 4 ) )         # digitization noise
poly = PolynomialModel( 1 )            # line
fitter = Fitter( x, poly )
param = fitter.fit( y )
stdev = fitter.getStandardDeviation()    # stdevs on the parameters
chisq = fitter.getChiSquared()
scale = fitter.getScale()            # noise scale
yfit  = fitter.getResult()            # fitted values
yfit  = poly( x )                # fitted values (same as previous)
yband = fitter.monteCarloError()     # 1 sigma confidence region
```

API Summary

Methods
getChiSquared Returns the Chi-Squared goodness of fit.
Double1d **getStandardDeviation** Returns the standard deviations pertaining to the parameters.
Double1d **monteCarloError** (NumericData input) Calculates 1σ-confidence regions on the model given some inputs.
getResult

Methods
Returns the fitted values (yfit).
getScale
Return the noise scale: sqrt(chisq/DOF).

Limitations

1. The Fitter does **not** work with limits.

2. The calculation of the evidence is an Gaussian approximation which is only exact for linear models with a fixed scale.

Miscellaneous

In case of problems look at the trouble shooting section.

API Details

Methods

getChiSquared
Returns the Chi-Squared goodness of fit.

***Double1d* getStandardDeviation**
Returns the standard deviations pertaining to the parameters. $\sigma_i = \sigma * \sqrt{(C_{i,i})}$. where C is the Covariance matrix, the inverse of the Hessian Matrix Standard deviation are calculated for the fitted parameters only. Note that the stdev will decrease with sqrt(N) of the number of datapoints while the noise scale, σ, does not. **Return** **Double1d** the standard deviations. **Error** `IllegalStateException` when chisq <= 0

***Double1d* monteCarloError (NumericData input)**
Calculates 1σ-confidence regions on the model given some inputs. From the full covariance matrix (= inverse of the Hessian) random samples are drawn, which are added to the parameters. With this new set of parameters the model is calculated. This procedure is done by default, 25 times. The standard deviation of the models is returned as the error bar. **Argument** `NumericData` **`input`** `[INPUT, OPTIONAL, default=null.]` Input over which to calculate the error bars. **Return** **Double1d**

Double1d **monteCarloError (NumericData input)**
the confidence vector (same length as input)

getResult
Returns the fitted values (yfit).

getScale
Return the noise scale: sqrt(chisq/DOF).

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.AbstractFitter

1.6. AbstractModel

Full Name:	herschel.ia.numeric.toolbox.fit.AbstractModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import AbstractModel
Category	Mathematics/Fitting

Description

AbstractModel implements the common parts of (compound) Models which can be handled by the Fitter class and its descendants.

A model consists of one or more instantiations of (basic) models which are concatenated in a chain of models using addModel. Almost all methods defined in AbstractBasicModel are overwritten here, recursively calling the AbstractBasicModel's methods.

An AbstractModel contains a NoiseScale, a measure of the noise.

As the AbstractModel is the last in line of the general, unspecific models it contains a number of Jython hooks and some methods to test the validity of partial derivatives against numerically calculated derivatives.

AbstractModel also implements a numerical derivation of partial to be used when partial is not given in the model definition itself. This same numerical derivation of partial is used in testPartial to indeed test whether the partial has been implemented properly.

An introduction to the use of fit package can be found here.

At least once have a look at the background on the organization and structure of the package. The fit/demo directory contains worked examples on almost all aspects of of the package.

Example

Example 1: AbstractModel

```
x = Double1d.range( 10 )
poly = PolynomialModel( 2 )                 # quadratic model
poly.setParameters( Double1d([3,2,1]) )     # set the parameters for the model
y = poly( x )                              # evaluate the model at x
p0 = poly[0]                               # 3: the first parameter
gauss = GaussModel()                       # gaussian model
gauss += PolynomialModel( 0 )              # gaussian on a constant background
print gauss.getNumberOfParameters()         # 4
lolim = Double1d([0,-10,0,-5])             # lower limits for the parameters
hilim = Double1d([10,10,2, 5])             # high limits for parameters
gauss.setLimits( lolim, hilim )             # set limits. Does not work with all
 Fitters
index = Int1d([0,3])
value = Double1d([4.5,0])
gauss.keepFixed( index, value )             # keep index parameters fixed at
 value
```

API Summary

Methods

`isolateModel (int k)`

Return a (isolated) copy of the k-th model in the chain.

Methods
addModel (AbstractModel model) Make a compound model by concatinating/adding model to this.
subtractModel (AbstractModel model) Make a compound model by concatinating/subtracting a model from this.
multiplyModel (AbstractModel model) Make a compound model by concatinating/multiplying a model with this
divideModel (AbstractModel model) Make a compound model by concatinating/dividing by a model.
keepFixed (Int1d fixed, Double1d values) Keeps parameters fixed at the provided values.
getParameters Returns parameters of the (compound) model.
getStandardDeviations Returns (present) standard deviations for the parameters of the (Compound) model.
setLimits (Double1d lolimits, Double1d hilimits) Sets the limits for the parameters of the compound model.

API Details

Methods

isolateModel (int k)

Return a (isolated) copy of the k-th model in the chain.

Argument

`int` **`k`** `[INPUT, MANDATORY, default=0]`

- Zero (0) value is for head.

addModel (AbstractModel model)

Make a compound model by concatinating/adding model to this.

The final result is the sum of the individual results.

The compound model is implemented as a chain of {@link AbstractBasicModel}s. Each of these basic models contain the attributes (parameters, limits etc.) and when needed these attributes are taken from there, or stored there.

The operation (addition in this case) is always with the total result of the existing chain. For the use of "brackets" in a chain use ContainerModel.

Argument

`AbstractModel` **`model`** `[INPUT, MANDATORY, default=no default value]`

The model to be added to this instance.

subtractModel (AbstractModel model)

Make a compound model by concatinating/subtracting a model from this.

The final result is the difference of the models.

Argument

subtractModel (AbstractModel model)

`AbstractModel` **`model`** `[INPUT, MANDATORY, default=no default value]`

The model to be subtracted from this instance.

multiplyModel (AbstractModel model)

Make a compound model by concatinating/multiplying a model with this

The final result is the product of the models.

Argument

`AbstractModel` **`model`** `[INPUT, MANDATORY, default=no default value]`

The model to be multiplied by this instance.

divideModel (AbstractModel model)

Make a compound model by concatinating/dividing by a model.

The final result is the division of the models.

Argument

`AbstractModel` **`model`** `[INPUT, MANDATORY, default=no default value]`

The model to be to be divided by this instance.

keepFixed (Int1d fixed, Double1d values)

Keeps parameters fixed at the provided values.

1. The model will act exactly as if it were a model with less parameters, although slightly less efficient.

2. Repeated calls start from scratch.

3. Reset with `keepFixed( null );` or with `keepFixed( Int1d( 0 ) );`

Arguments

`Int1d` **`fixed`** `[INPUT, MANDATORY, default=no default value]`

A list of parameter indices to be kept fixed. (any order)

`Double1d` **`values`** `[INPUT, OPTIONAL, default=no default value]`

The values at which the parameters should be kept. length of values <= length of fixed; the remainder is left as is. They should be in the proper order (that of the model).

Error

`IndexOutOfBoundsException`

when a fixed index is out of range.

getParameters

Returns parameters of the (compound) model.

getStandardDeviations

Returns (present) standard deviations for the parameters of the (Compound) model.

setLimits (Double1d lolimits, Double1d hilimits)

Sets the limits for the parameters of the compound model.

1. It is valid to insert for either parameter a (Double1d)null value indicating no lower/upper limits.

setLimits (Double1d lolimits, Double1d hilimits)

2. When a lowerlimit >= upperlimit no limits are enforced.

It only works in *Fitter classes which can support it. Eg. Fitter itself cannot handle limits as it would turn the problem into a **non**-linear one. Note: Limits will also work when the model is used in a Minimizer. In that case the limits apply to the minimization range.

Arguments

`Double1d` **`lolimits`** `[INPUT, MANDATORY, default=no default value]`

The lower limits.

`Double1d` **`hilimits`** `[INPUT, MANDATORY, default=no default value]`

The upper limits.

See also

- Fitter
- AbstractBasicModel
- NoiseScale
- Developers Manual: herschel.ia.numeric.toolbox.fit.AbstractModel

1.7. accumulate

Full Name:	herschel.ia.toolbox.spectrum.AccumulateSpectrumTask
Alias:	accumulate
Type:	Java Task -
Import:	from herschel.ia.toolbox.spectrum import AccumulateSpectrumTask
Category	Spectra/Analysis

Description

Accumulates (averages) spectra to a common wave scale grid and returns a `Spectrum1d`.

In contrast to the average task it checks whether the spectra align at the wave scale (within a given tolerance specified by the `wavescaleTolerance` parameter) and automatically does a resampling if not. Furthermore, the shapes need not be the same. Note that by default the wavescaleTolerance is 0. As a result, resampling is applied even in case the spectra are mis-aligned only at the last significant digit. As a result of resampling, spurious NaN's may occur at the edge of the result spectra. As a remedy to avoid these when processing spectra with almost perfectly aligning wave scales is to set a small wavescaleTolerance.

The task finds the overlapping ranges and if necessary resamples to the overlapping regions. The user can specify a new grid the original spectra should be resampled to before performing the averaging or, if resampling is necessary, the task computes a grid that envelopes all the input spectra with a bin width defined by the coarsest (average) bin width found for of all the input spectra.

The user can also specify a range for the result spectrum (using the `range` parameter).

Further checks involve

- the unit the intensities / flux values are expressed in: Here no automatic transformation to a common scale is available - all the intensities of the spectra to be accumulated need to be the same.
- the pointing information associated with the individual spectra: The spectra need to have consistent pointing information up to within the tolerance specified (see the parameter `pointingTolerance`). If NaN pointing information is found in the spectra, the pointing check is skipped. Furthermore, if setting `pointingTolerance=Double.NaN`, the pointing check is skipped. In this way, spectra can be processed without pointing information included (such as the result of this task).

Similar to the average task a `variant` parameter can be set to specify ho to deal with flags (mask) or weights (error). In contrast to e.g. the average task further attributes, such as integration time are not checked nor processed.

Finally, selections can be specified the same way as with the average task (by using the `selection` and `filter_meta` parameters).

The result is provided in form of a `Spectrum1d`. In addition to columns with the wave, flux and possibly flag and weight data it contains a column with the number of spectra involved for a given pixel (`counts`).

Example

Example 1: In HIPE:

```
global spectra, htp, htpH, htpV, htpHU, htpHL # defined elsewhere
# single spectrum container (dataset) 'ds', restructing the processing to
 suitable sub-selection of point spectra
# grid determined from data
result = accumulate(ds=spectra, unit="MHz")
```

Example 1: In HIPE:

```
# grid specified by user
grid = Double1d.range(2000)+600.0
result = accumulate(ds=spectra, grid = grid, unit="MHz")
# many spectrum container , eg from a map context such as a HifiTimelineProduct
 'htp'
# - restricting to spectrum containers that fulfill condition on meta data
# note that the origin nor the type of these spectrum containers need to be the
 same
dsArray = [htp[4], htp[5], htp[6], htp[7], htp[8], htp[11]]
result = accumulate(ds=dsArray, grid = grid, filter_meta={"sds_type":
["science"]})
# map contexts such as timeline products, 'htpH' or 'htpV'
result = accumulate(ds=htpH, grid = grid, filter_meta={"sds_type":["science"]},
 selection={"bbtype":[6031]})
result = accumulate(ds=[htpH,htpV], grid = grid, filter_meta={"sds_type":
["science"]}, selection={"bbtype":[6031]})
# giving more tolerance to deviations between wavescales - so that less
 resampling is done
result = accumulate(ds=[htpHU,htpHL], wavescaleTolerance=0.001)
# giving more tolerance to deviations of the pointings
#- so that, potentially, more spectra are accumulated (averaged).
result = accumulate(ds=[htpHU,htpHL], pointingTolerance=1.0) # number specified
 in degrees
# directly transforming to grid specified in mm wavelength
result = accumulate(ds=[htpHU,htpHL], unit="mm", pointingTolerance=0.001)
# specify a range to restrict the output spectrum to
result = accumulate(ds=spectra, range=(1000.0,1500.0), unit="GHz")
result = accumulate(ds=spectra, range=( 800,1200.0), grid = grid, unit="MHz")
```

API Summary

Jython Syntax
See examples below.

Properties
Object **ds** [INOUT, MANDATORY, default=no default value]
String **variant** [INOUT, OPTIONAL, default="flux"]
Double1d **grid** [INOUT, OPTIONAL, default=no default value]
PyTuple **range** [INPUT, OPTIONAL, default=no default value.]
Object **selection** [INPUT, OPTIONAL, default=None.]
PyDictionary **filter meta** [INPUT, OPTIONAL, default=No default value.]
UNIT **unit** [INOUT, OPTIONAL, default=no default value]
Integer **flagToIgnore** [INOUT, OPTIONAL, default=no default value]
String **resampler** [INOUT, OPTIONAL, default="euler"]
Double **wavescaleTolerance** [INOUT, OPTIONAL, default=0.0]
Double **pointingTolerance** [INOUT, OPTIONAL, default=0.001]

API details

Properties

Object ds [INOUT, MANDATORY, default=no default value]

The input spectrum data: a SpectrumContainer, and array of such, or a HifiTimelineProduct or an array of such or a mixture of all.

String variant [INOUT, OPTIONAL, default="flux"]

The variant that tells how to do the averaging:

- straight average of all the data
- straight average but omitting flagged data
- weighted average of all data
- weighted average but omitting flagged data.

See also 'flagValue' for further information.

Double1d grid [INOUT, OPTIONAL, default=no default value]

Specify what wave scale to target at. If no grid is specified a grid will be determined by the task which basically will envelope all the wave scales found in the data and picks the coarsest width.

PyTuple range [INPUT, OPTIONAL, default=no default value.]

Specify the range to restrict the result spectrum to - in units as specified by the unit parameter. This parameter is specified as a PyTuple, e.g. `(530.0, 610.0)`.

Object selection [INPUT, OPTIONAL, default=None.]

Specification of what point spectra the average should be restricted to. Different ways to specify these selections are possible:

- Specify a list of indices (in jython) of the point spectra for which the average should be taken. For cubes, you can alternatively also pass a list of spaxel coordinates (a list of integer tuples `(row,col)`).
- Use a dictionary (in jython) to specify a selection model (lookup for attributes, filter attributes to be included in ranges or intervals).
- Pass a string representation of a jython list or a jython dictionary. The string will be parsed and the processing delegated to the two cases above.
- Pass any java instance that implements the SelectionModel interface (for the advanced user).

See the examples below or the `SelectSpectrum`-task for how to specify selections.

PyDictionary filter_meta [INPUT, OPTIONAL, default=No default value.]

Restrict the operation on spectrum containers with meta data that match given values.

UNIT unit [INOUT, OPTIONAL, default=no default value]

The unit the target grid and the range is expressed. In case no unit parameter is specified the unit to express interpret the grid or the range in is obtained from the 'first' input spectra. If the input spectra are expressed in different units we recommend to explicitly set the unit parameter.

Integer flagToIgnore [INOUT, OPTIONAL, default=no default value]

The integer value representing the mask for flags to be ignored.

String resampler [INOUT, OPTIONAL, default="euler"]

Resampling scheme to be adopted. Available are "euler", "trapezoidal", "gaussian"

Double wavescaleTolerance [INOUT, OPTIONAL, default=0.0]

The tolerance (given in unit of the output grid) used to check whether resampling is necessary or not.

Double pointingTolerance [INOUT, OPTIONAL, default=0.001]

Pointing tolerance used to checked whether the pointing (longitude and latitude expressed in degrees) sufficiently align. If setting the `pointingTolerance` to NaN, the pointing check is skipped.

See also

- Developers Manual: herschel.ia.toolbox.spectrum.AccumulateSpectrumTask

History

- `2011-10-17 - melchior`: initial version.
- `2012-01-13 - melchior`: added possibility to specify range, some bugs of the initial version fixed.

1.8. add

Full Name:	herschel.ia.toolbox.spectrum.AddSpectrumTask
Alias:	add
Type:	Java Task -
Import:	from herschel.ia.toolbox.spectrum import AddSpectrumTask
Category	Spectra/Analysis

Description

Task for adding a scalar to the flux data of spectra or for pairwise adding spectra.

For the scalar mode, 'ds' and 'param' need to be specified - the input spectra and the scalar to add; for the pair-wise mode, 'ds1' and 'ds2' need to be set - two spectrum containers. In this pair-wise mode, the first spectrum in the first container is added to the first spectrum in the second container, and so on. In case the size of the two containers is different, the result will contain a number of point spectra equal to the minimum size. Hence, the remaining spectra in the larger container are ignored. The behavior is different in case one of the datasets only contains a single spectrum: Here, the task always refers to single spectrum while iterating over all the spectra in the other container.

In the pairwise mode, the individual spectra are processed on a per frequency or wavelength bin (or whatever the wavescale unit is) basis. Hence, there is no check of whether the frequencies (or wavelengths) assigned to these bins are well aligned across the two spectra. If this is not the case, the spectra should first be resampled to a common wavescale grid.

The input data with the spectra to be processed needs to be an object that implements `SpectrumContainer` (e.g. `Spectrum1d, Spectrum2d, SpectralSimpleCube`). `SpectrumContainer`'s provides a uniform way to access spectrum data organized in quite different structures.

For a successful processing in the pair-wise mode, the data specified with `ds1` and `ds2` should be consistent, i.e. they should have the same wavescale range and spectral sampling. Furthermore, the segments found in all the spectra in the (two) containers should be consistent (same number of segments, same lengths, same segment indices - e.g. check with `ds.getSegmentIndices()`). Otherwise an exception is thrown.

Different selection schemes are available for selecting the point spectra or the segments to be processed. The most simple scheme is to specify lists of point spectrum indices (`selection=[1,3,4,2]`). In this case, the operation (scalar- or pair-operation) is just taken over the point spectra with corresponding indices. Instead of specifying a list of indices (which requires the knowledge of these) other recipes to specify a selection are available. Typical recipes test attribute values to match certain values (e.g. `selection={"bbtype":[6031, 6032]}`) or to be included in a given interval (e.g. `selection={"LoFrequency":(999.98,1000.02)}`). These recipes we refer to as selection models. The different options to specify selections can be combined. Here, an AND logic is adopted. See the parameter descriptions and the examples below for further detail. Alternatively, look in the `SelectSpectrum`-task.

Similarly, you can specify what segments to consider. In the most simple case, you can just specify the indices of segments to be considered (for the scalar operation `segments=[1,3,4]` and for the pair operation `segments1=[1,3,4], segments2=[3,6,5]`). In more advanced situations you can use a `SegmentSelection` object. Note that the pairs of segments to be processed need to be consistent - otherwise the processing will fail.

You can specify whether the orginal datasets should be overwritten with the 'overwrite' flag. By default no data is overwritten. In the pair-wise mode, it is not always possible to overwrite the data - in particular if non-trivial segment selections are specified.

Examples

Example 1: for pairwise add:

```
global spectra1, spectra2 # defined elsewhere
add = AddSpectrumTask()
spectraOut = add(ds1=spectra1, ds2=spectra2)
```

Example 2: for scalar add:

```
global spectra # defined elsewhere
add = AddSpectrumTask()
spectraOut = add(ds=spectra, param=2.1)
# restrict the add to a suitable selection of spectra and return the processed
# just select by index:
spectraOut = add(ds=spectra, selection=[0,1,2,3], param=2.1)
# select by looking up suitable attributes attached to the spectra to be
 selected (here: "bbtype" matched):
spectraOut = add(ds=spectra, selection={"bbtype":[6613]}, param=2.1)
# select by looking up suitable attributes attached to the spectra to be
 selected (here: "Chopper" in given ranges):
spectraOut = add(ds=spectra, selection={"Chopper":(4.,7.), "bbtype":[6613]},
 param=2.1)
# select by looking up suitable attributes attached to the spectra to be
 selected
# (here: "LoFrequency" in given interval):
spectraOut = add(ds=spectra, selection={"LoFrequency":(500., 600.)}, param=2.1)
# the same as above - this time by by modifying the input spectra:
spectra = add(ds=spectra, selection=[0,1,2,3], param=2.1, overwrite=True)
spectra = add(ds=spectra, selection={"bbtype":[6031]}, param=2.1,
 overwrite=True)
spectra = add(ds=spectra, selection={"Chopper":([-4.4,5.9],0.2), "bbtype":
[6031]}, param=2.1, overwrite=True)
spectra = add(ds=spectra, selection={"LoFrequency":(4000.0,5000.0)}, param=2.1,
 overwrite=True)
```

Example 3: for pairwise add:

```
global spectra1, spectra2 # defined elsewhere
add = AddSpectrumTask()
# pairwise add ds2 to ds1
spectraOut = add(ds1=spectra1, ds2=spectra2)
# restrict the pairwise add to suitable selection of spectra and return the
 processed
# just select by index:
spectraOut = add(ds1=spectra1, ds2=spectra2, selection=[0,1,2,3])
# select by looking up suitable attributes attached to the spectra to be
 selected (here: "bbtype" matched):
spectraOut = add(ds1=spectra1, ds2=spectra2, selection={"bbtype":[6613, 6031]})
# select by looking up suitable attributes attached to the spectra to be
 selected (here: "Chopper" in given ranges):
spectraOut = add(ds1=spectra1, ds2=spectra2, selection={"Chopper":(4.,7.),
 "bbtype":[6613]})
# select by looking up suitable attributes attached to the spectra to be
 selected
# (here: "LoFrequency" in given interval):
spectraOut = add(ds1=spectra1, ds2=spectra2, selection={"LoFrequency":
(550.0,570.0)})
```

API Summary

Properties
SpectrumContainer **ds** [INPUT, OPTIONAL, default=no default value.]

Properties
Double **param** [INPUT, OPTIONAL, default=no default value.]
Object **selection** [INPUT, OPTIONAL, default=None.]
PyDictionary|Map<String,Set<Object>> **lookup_selection** [INPUT, OPTIONAL, default=no default value.]
PyList **index_selection** [INPUT, OPTIONAL, default=No default value.]
Boolean **overwrite** [INPUT, OPTIONAL, default=False.]
SpectrumContainer **ds1** [INPUT, OPTIONAL, default=no default value.]
SpectrumContainer **ds2** [INPUT, OPTIONAL, default=no default value.]
Object **segments** [INPUT, OPTIONAL, default=no default value.]
Object **segments1** [INPUT, OPTIONAL, default=no default value.]
Object **segments2** [INPUT, OPTIONAL, default=no default value.]
SpectrumContainer **result** [OUTPUT, OPTIONAL, default=no default value]

API details

Properties

SpectrumContainer ds [INPUT, OPTIONAL, default=no default value.]

Input container to be processed by the task in case the task is used as a scalar operation. Examples of `SpectrumContainer` are `Spectrum1d, Spectrum2d, SpectralSimpleCube`.

Double param [INPUT, OPTIONAL, default=no default value.]

The scalar parameter to be considered when the task is used as scalar operation.

Object selection [INPUT, OPTIONAL, default=None.]

Specify what point spectra the operation should be applied to. Different ways to specify these selections are possible:

- Specify a list of indices (in jython) of the point spectra for which the add should be applied.
- Use a dictionary (in jython) to specify a selection model (lookup for attributes, filter attributes to be included in ranges or intervals).
- Pass a string representation of a jython list or a jython dictionary. The string will be parsed and the prcessing delegated to the two cases above.
- Pass any java instance that implements the SelectionModel interface (for the advanced user).

See the examples below or the SelectSpectrumTask-task for how to specify selections.

PyDictionary|Map<String,Set<Object>> lookup_selection [INPUT, OPTIONAL, default=no default value.]

Specify a PyDictionary with keys given by the attribute name to look up and a list of admissible values. Behind the scenes, a DiscreteValueSelectionModel is instantiated. This parameter is actually obsolete but is kept for historical reasons (use `selection`).

PyList index_selection [INPUT, OPTIONAL, default=No default value.]

Specify a PyList with the indices of the point spectra to be considered. This parameter is actually obsolete but is kept for historical reasons.

Boolean overwrite [INPUT, OPTIONAL, default=False.]

Specify whether the input data container can be reused - the values found therein is overwritten.

SpectrumContainer ds1 [INPUT, OPTIONAL, default=no default value.]

First input container for pair-wise operations.

SpectrumContainer ds2 [INPUT, OPTIONAL, default=no default value.]

Second input container for pair-wise operations.

Object segments [INPUT, OPTIONAL, default=no default value.]

Specify what segments the operation should be applied to. There are two options available:

- Either pass an instance of a SegmentSelection which gives the information on what segments for each point spectrum included in the container.
- Specify a PyList of segment indices.

Object segments1 [INPUT, OPTIONAL, default=no default value.]

Specify the segment selection to be associated with 'ds1'.

Object segments2 [INPUT, OPTIONAL, default=no default value.]

Specify the segment selection to be associated with 'ds2'.

SpectrumContainer result [OUTPUT, OPTIONAL, default=no default value]

Result object containing the results of the operation applied.

See also

- SpectrumContainer
- select
- Developers Manual: herschel.ia.toolbox.spectrum.AddSpectrumTask

History

- `2011-08-08 - melchior`: renamed from AddSpectrum

1.9. ALL

Full Name:	herschel.ia.numeric.toolbox.basic.All
Alias:	ALL
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import All
Category	Arrays and datasets/Element selection

Description

Determines whether all the elements of an array satisfy an expression.

This function can return a global `True` or `False` result for the whole array, or test the expression along each dimension of a multidimensional array if the `dimension` parameter is specified. For example, you could test an expression on all the rows, or all the columns, of a two-dimensional array. See also the example below.

Example

Example 1: Apply ALL to an Int2d: check if all the array items, in the specified dimension, are lower than 3.

```
x = Int2d( [ [1,2], [1,4], [1,6] ])
print ALL(x&lt;3)    # All items =&gt; 0 (false)
print ALL(x&lt;3, 0) # Dim 0 =&gt; [ ALL(Int1d([1,1,1]) &lt; 3),
                   # ALL(Int1d([2,4,6]) &lt; 3) ] = [true, false]
print ALL(x&lt;3, 1) # Dim 1 =&gt; [ ALL(Int1d([1,2]) &lt; 3),
                   # ALL(Int1d([1,4]) &lt; 3), ALL(Int1d([1,6]) &lt; 3) = [true,
 false, false]
```

API Summary

Jython Syntax
`<y> = ALL(<array_expression> [,<dimension>])`

Properties
`Logical expression` **`array_expression`** `[INPUT, MANDATORY, default=no default value]`
`Integer` **`dimension`** `[INPUT, OPTIONAL, default=all dimensions]`
`Boolean or BoolNd` **`y`** `[OUTPUT, MANDATORY, default=no default value]`

API details

Properties

`Logical expression array_expression [INPUT, MANDATORY, default=no default value]`

The expression to be tested. You can use any expression involving an array and evaluating to `True` or `False` for each array element.

`Integer dimension [INPUT, OPTIONAL, default=all dimensions]`

The dimension along which to test the expression.

Boolean or BoolNd y [OUTPUT, MANDATORY, default=no default value]
The result of testing the expression.

See also

- Developers Manual: herschel.ia.numeric.toolbox.basic.All

1.10. AllPresent

Full Name:	herschel.ia.numeric.toolbox.basic.AllPresent
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import AllPresent
Category	Arrays and datasets/Element selection

Description

Checks whether all the bits in the specified bitmask are present in the elements of the input array.

For example, if the bitmask is 7 (111 in binary) and one array element is 5 (101 in binary), the function returns `False`, for that element, because one bit in the mask is not set in the array value.

Example

Example 1: Apply AllPresent to an Int1d

```
x = Int1d([0,1,2,3])
print AllPresent(3)(x)
#=&gt; [  (0 &amp; 3) == 3,   (1 &amp; 3) == 3,   (2 &amp; 3) == 3,   (3 &amp;
 3) == 3  ] = [false,false,false,true]
print x.apply(AllPresent(3)) # Another way of using AllPresent
```

API Summary

Jython Syntax

```
&lt;y&gt;=AllPresent(&lt;bitmask&gt;)(&lt;x&gt;)
```

Properties

```
Integer type array x [INPUT, MANDATORY, default=no default value]
Integer bitmask [INPUT, MANDATORY, default=no default value]
Bool1d y [OUTPUT, MANDATORY, default=no default value]
```

API details

Properties

`Integer type array x [INPUT, MANDATORY, default=no default value]`

The input array. It can be of any integer type: ByteNd, ShortNd, IntNd, LongNd.

`Integer bitmask [INPUT, MANDATORY, default=no default value]`

The bitmask.

`Bool1d y [OUTPUT, MANDATORY, default=no default value]`

The result of the comparison. Each array element is `True` if all the bits of the mask are present in the corresponding input array element, `False` otherwise.

See also

- AnyPresent

- NotPresent
- Developers Manual: herschel.ia.numeric.toolbox.basic.AllPresent

1.11. AmoebaFitter

Full Name:	herschel.ia.numeric.toolbox.fit.AmoebaFitter
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import AmoebaFitter
Category	Mathematics/Fitting

Description

Simulated annealing simplex finding minimum.

A simplex of (N+1) points is used to crawl downhill, using various kinds of steps: reflection, expansion, contraction and shinkage. At each iteration they are tried in this order until one is found which is successfull.

AmoebaFitter can be used in two modes: with simulated annealing on or off. The simulated annealing mode is invoked by setting the temperature to some value. By default it is off: temperature at zero.

When the temperature is set to (or left at) zero, AmoebaFitter acts as a simple Nelder-Mead downhill simplex method. With two advantages and one disadvantage. The pro's are that it is reasonably fast and that it does not need partial derivatives. The con is that it will fall into the first local minimum it encounters. No guarantee that this minimum has anything to do with the absolute minimum. This T=0 modus can only be used in mono-modal problems.

In the other modus, when the temperature is set at some value the simplex sometimes takes a uphill step, depending on the temperature at that moment. Steps downhill are always taken. In that way it is possible to climb out of local minima to find better ones. Meanwhile the temperature is steadily lowered, concentrating the search on the by now hopefully found absolute minimum. Of course this takes much more iterations and still there is **no guarantee** that the best value is found. But a better chance. The initial temperature which suggests itself is of the order of the difference between χ^2 at a random point and χ^2 at the minimum.

AmoebaFitter can be used with limits set to one or more of the parameters.

An introduction to the use of fit package can be found here.

At least once have a look at the background on the organization and structure of the package.

The fit/demo directory contains worked examples on almost all aspects of of the package.

Example

Example 1: AmoebaFitter (for non-linear models)

```
# assume x and y are Double1d data arrays.
x = Double1d.range(100) / 10
y = 3.5 * SIN( x + 0.4 )                # make sine
rg = RandomGauss( seed=12345L )           # Gaussian random number generator
y += rg( Double1d(100) ) * 0.2           # add noise
sine = SineModel( )                        # sinusiodal model
lolim = Double1d([1,-10,-10])
hilim = Double1d([100,10,10])
sine.setLimits( lolim, hilim )           # set limits on the model parameters
amfit = AmoebaFitter( x, sine )
amfit.setTemperature( 10 )               # set a temperature to escape local
 minima
param = amfit.fit( y )
stdev = amfit.getStandardDeviation()     # stdevs on the parameters
chisq = amfit.getChiSquared()
```

Example 1: AmoebaFitter (for non-linear models)

```
scale = amfit.getScale()                # noise scale
yfit  = amfit.getResult()               # fitted values
yfit  = sine( x )                       # fitted values (same as previous)
yband = amfit.monteCarloError()           # 1 sigma confidence region
# for diagnostics (or just for fun)
amfit = AmoebaFitter( x, sine )
amfit.setTemperature( 10 )               # set a temperature to escape local
 minima
amfit.setVerbose( 10 )                   # report every 10th iteration
plotter = IterationPlotter()            # from herschel.ia.toolbox.fit
amfit.setPlotter( plotter, 20 )           # make a plot every 20th iteration
param = amfit.fit( y )
```

Limitations

1. AmoebaFitter is **not** guaranteed to find the global minimum.

2. The calculation of the evidence is an Gaussian approximation which is only exact for linear models with a fixed scale.

Miscellaneous

In case of problems look at the trouble shooting section.

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.AmoebaFitter

1.12. angles2RotationMatrix

Full Name:	herschel.ia.toolbox.pointing.angles2RotationMatrix
Type:	Jython Task -
Import:	from herschel.ia.toolbox.pointing import angles2RotationMatrix
Category	Toolboxes/Pointing

Description

Function to go from Euler angles to a 3x3 rotation matrix.

API Summary

Jython Syntax
`rotationMatrix = angles2RotationMatrix(angle1, angle2, angle3)`

Properties
`Double1d` **`angle1`** `[INPUT, MANDATORY, default=NO default value]`
`Double1d` **`angle2`** `[INPUT, MANDATORY, default=NO default value]`
`Double1d` **`angle3`** `[INPUT, MANDATORY, default=NO default value]`
`Double2d` **`rotationMatrix`** `[OUTPUT, MANDATORY, default=NO default value ]`

API details

Properties

`Double1d angle1 [INPUT, MANDATORY, default=NO default value]`
rotation angles over 1st axis (radians)

`Double1d angle2 [INPUT, MANDATORY, default=NO default value]`
rotation angles over 2nd axis (radians)

`Double1d angle3 [INPUT, MANDATORY, default=NO default value]`
rotation angles over 3rd axis (radians)

`Double2d rotationMatrix [OUTPUT, MANDATORY, default=NO default value ]`
rotation matrix

History

- `2013-04-22 -` BV: Initial version

1.13. AnnotationToolbox

Full Name:	herschel.ia.gui.image.gui.AnnotationToolbox
Type:	Java Class -
Import:	from herschel.ia.gui.image.gui import AnnotationToolbox
Category	Images/Display

Description

A toolbox to draw annotations.

AnnotationToolbox constructs a toolbox where the user can select which annotation to display. Using the mouse, the annotation can be put on the image. The jython command to reconstruct the annotation is also shown.

API Summary

Constructors
`AnnotationToolbox (Display d)` Constructor for the annotation toolbox.
`AnnotationToolbox (Display d, boolean isComponent)` Constructs the annotation toolbox.

Methods
`getComponent` Returns the component that contains the annotation toolbox.
`close` Closes the annotation toolbox.

API Details

Constructors

`AnnotationToolbox (Display d)`

Constructor for the annotation toolbox.

Argument

`Display` **`d`** `[INPUT, MANDATORY, default=no default value]`

The Display for which the toolbox should be made.

`AnnotationToolbox (Display d, boolean isComponent)`

Constructs the annotation toolbox.

If useAsComponent is true, the annotation toolbox is component based.

Arguments

`Display` **`d`** `[INPUT, MANDATORY, default=no default value]`

The Display for which the toolbox should be made.

`boolean` **`isComponent`** `[INPUT, MANDATORY, default=no default value]`

AnnotationToolbox (Display d, boolean isComponent)
true if the AnnotationToolbox should be component based, false otherwise.

Methods

getComponent
Returns the component that contains the annotation toolbox.

close
Closes the annotation toolbox.

See also

- Developers Manual: herschel.ia.gui.image.gui.AnnotationToolbox

1.14. annularSkyAperturePhotometry

Full Name:	herschel.ia.toolbox.image.AnnularSkyAperturePhotometryTask
Alias:	annularSkyAperturePhotometry
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import AnnularSkyAperturePhotometryTask
Category	Astronomical utilities/Photometry

Description

This is a task that performs aperture photometry of target, enclosed

by a circular aperture. The sky is estimated from a concentric annular aperture, with configurable inner and outer radii. You need to specify the source location (i.e. the center of the circular aperture) either in pixel coordinates (with the `centerX` and `centerY` parameters) or in sky coordinates (with the `centerRa` and `centerDec` parameters), the radii of the sky annulus and which of five possible sky estimation algorithms should be used. The input image might have a flag dataset attached to it. Pixels that have been flagged are omitted from the calculation.

The following sky estimation algorithms are available (note that the `algorithm` parameter only accepts integer values):

- `algorithm` = 0 : **average** : The average intensity of the pixels contributing to the flux in the sky aperture. In case fractional pixels are used, the pixels are weighted according to their overlap with the sky aperture. Otherwise we use the average of the intensities of the contributing pixels.
- `algorithm` = 1 : **median** : The median intensity of the pixels contributing to the flux in the sky aperture. In case fractional pixels are used, the pixels are weighted according to their overlap with the sky aperture. Else, we use the median of the intensities of the contributing pixels.
- `algorithm` = 2 : **mean-median** : Only pixels which are less than 1.5 times the standard deviation (w.r.t. the average intensity) away from the median are used and the mean-median is the average of the remaining pixels. The same remarks hold concerning the fractional pixels.
- `algorithm` = 3 : **synthetic mode** : The mean-median algorithm is repeated in an iterative process up to ten times or until the average does not change anymore. The mean median is then (3 * median) - (2 * average) of the remaining intensities. The same remarks hold concerning the fractional pixels.
- `algorithm` = 4 : **daophot** : Sky estimation algorithm as adapted from the IDL mmm.pro routine. Have a look at the documentation of IDL AstroLib for more information. This algorithm is the default option.

The background corrected flux density in the aperture is calculated using the following pseudo code:

```
Target (bg subtr) = Total(Jy) - Background(Jy/pixel) * #pixels(in the target
 aperture)
```

where `Total(Jy)` is the total flux density in the user selected aperture, `Background (Jy/pixel)` is the average background/sky estimation in the annular ring, calculated with the selected algorithm and `#pixels` is the number of pixels in the target aperture. By default the sky estimation uses fractional pixels but there is an option to switch to entire pixels. The errors are calculated the same way as in the IDL aper.pro routine (have a look at the documentation of IDL AstroLib for more information). These errors only apply for CCD observations. The three factors that contribute to the error on the target flux are :

- scatter in sky values

- random photon noise
- uncertainty in mean sky brightness

The error for the target flux including the background is defined as the square-root of the (absolute value of the) total flux in the target aperture (including the background). The error for the sky is defined as the standard deviation on the sky value. Note that this standard deviation is calculated only with the sky intensities that are used to calculate the sky intensity (as there are sky estimation algorithms that reject sky values). The squared error for the target flux is thus :

- the absolute value of the target flux of which the sky contribution has been subtracted
- the standard deviation of the sky value * the surface of the target aperture
- the squared standard deviation on the sky value

Units of the output: The unit of the integrated flux is the unit of the image, multiplied with "pixel". So aperture photometry on an image in Jy/pixel will yield an integrated flux in Jy. For images in Jy/beam, the integrated flux will be expressed in Jy/beam*pixel, if you don't convert the image to Jy/pixel yourself beforehand. This can be done with the ConvertImageUnitTask. Additionally, if the input unit is MJy/sr, the output will still be in Jy (the task does the conversion).

Examples

Example 1: This is an example how you can use the

```
annularSkyAperturePhotometry by specifying the target center
in pixel coordinates : result =
annularSkyAperturePhotometry(image = myImage, centerX =
183.2, centerY = 96.5, \ radiusArcsec = 5.0,innerArcsec =
10.0, outerArcsec = 50.0, fractional = 1, algorithm = 4)
```

Example 2: This is an example how you can use the

```
annularSkyAperturePhotometry by specifying the target center
in sky coordinates : result =
annularSkyAperturePhotometry(image = myImage, centerRa =
"05:46:45.9", centerDec = "-51:07:57.0", \ radiusArcsec =
5.0,innerArcsec = 10.0, outerArcsec = 50.0, fractional = 0,
algorithm = 1, centroid = True)
```

API Summary

Properties
Image **image** [INPUT, MANDATORY, default=None The image on which to]
Double **centerX** [INPUT, OPTIONAL, default=NaN The x-pixel-coordinate of]
Double **centerY** [INPUT, OPTIONAL, default=NaN The y-pixel-coordinate of]
String **centerRa** [INPUT, OPTIONAL, default="centerRa" The right ascension]
String **centerDec** [INPUT, OPTIONAL, default="centerDec" The declination]
Double **radiusPixels** [INPUT, OPTIONAL, default=NaN The target radius]

Properties
Double **radiusArcsec** [INPUT, OPTIONAL, default=NaN The target radius in]
boolean **centroid** [INPUT, OPTIONAL, default=false Indicates whether]
Integer **algorithm** [INPUT, OPTIONAL, default=4 The sky estimation]
Double **innerPixels** [INPUT, OPTIONAL, default=NaN The inner radius of the]
Double **outerPixels** [INPUT, OPTIONAL, default=NaN The outer radius of the]
Double **innerArcsec** [INPUT, OPTIONAL, default=NaN The inner radius of the]
Double **outerArcsec** [INPUT, OPTIONAL, default=NaN The outer radius of the]
boolean **fractional** [INPUT, OPTIONAL, default=true Indicates whether you]
String **centerRa** [INPUT, OPTIONAL, default="centerRA" The right ascension]
AnnularSkyAperturePhotometryProduct **result** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Image image [INPUT, MANDATORY, default=None The image on which to]

perform aperture photometry.

Double centerX [INPUT, OPTIONAL, default=NaN The x-pixel-coordinate of]

the target center.

Double centerY [INPUT, OPTIONAL, default=NaN The y-pixel-coordinate of]

the target center.

String centerRa [INPUT, OPTIONAL, default="centerRa" The right ascension]

of the target center. Using this only makes sense if the input image has a valid Wcs attached to it. The input format for the right ascension is "hh mm ss.s", "hh:mm:ss.s" or "dd.ddd".

String centerDec [INPUT, OPTIONAL, default="centerDec" The declination]

of the target center. Using this only makes sense if the input image has a valid Wcs attached to it. The input format for the declination is "[+/-]dd mm ss.s", "[+/-]dd:mm:ss.s" or "[+/-]dd.ddd".

Double radiusPixels [INPUT, OPTIONAL, default=NaN The target radius]

(i.e. the radius of the circular target aperture) in pixels.

Double radiusArcsec [INPUT, OPTIONAL, default=NaN The target radius in]

arcsec. Using this only makes sense if the input image has a valid Wcs attached to it and the pixel scaling is the same along both axis (in absolute value).

boolean centroid [INPUT, OPTIONAL, default=false Indicates whether]

centroiding on the target is needed.

Integer algorithm [INPUT, OPTIONAL, default=4 The sky estimation]

algorithm. Possible values are 0 (average), 1 (median), 2 (mean-median), 3 (sythetic mode) and 4 (daophot).

Double innerPixels [INPUT, OPTIONAL, default=NaN The inner radius of the]

annular sky aperture in pixels.

Double outerPixels [INPUT, OPTIONAL, default=NaN The outer radius of the]

annular sky aperture in pixels

Double innerArcsec [INPUT, OPTIONAL, default=NaN The inner radius of the]

annular sky aperture in arcsec.

Double outerArcsec [INPUT, OPTIONAL, default=NaN The outer radius of the]

annular sky aperture in arcsec.

boolean fractional [INPUT, OPTIONAL, default=true Indicates whether you]

want to use fractional pixel to estimate the sky intensity.

String centerRa [INPUT, OPTIONAL, default="centerRA" The right ascension]

of the target center. Using this only makes sense if the input image has a valid Wcs attached to it. The input format for the right ascension is "hh mm ss.s", "hh:mm:ss.s" or "dd.ddd". Deprecated : use centerRa

AnnularSkyAperturePhotometryProduct result [OUTPUT, MANDATORY, default=no default value]

None The result of the aperture photometry. Here you can find the flux in the target aperture (with and without background contribution) and the sky aperture, the number of pixels in the apertures and the error on the fluxes. From the curve of growth you can judge whether a good value for the target radius was chosen (under the assumption that the given sky value was accurate enough). The sky intensity plot helps to determine whether the given annulus is drawn at a location appropriate for sky estimation.

See also

- Developers Manual: herschel.ia.toolbox.image.AnnularSkyAperturePhotometryTask

1.15. AnnularSkyAperturePhotometryProduct

Full Name:	herschel.ia.dataset.image.AnnularSkyAperturePhotometryProduct
Type:	Java Class -
Import:	from herschel.ia.dataset.image import AnnularSkyAperturePhotometryProduct
Category	Astronomical utilities/Photometry

Description

This is a class to deal with the results of aperture photometry with a circular target aperture and an annular sky aperture.

This class is used to store the output of the {@link herschel.ia.toolbox.image.AnnularSkyAperturePhotometryTask AnnularSkyAperturePhotometryTask}. Aperture specific information is stored in metadata, all other data (results of calculations) are stored in TableDatasets.

Example

Example 1: Script demonstrating the use of an AnnularSkyAperturePhotometryProduct

```
# get hold of the pixel coordinates of the target center :
coords = product.getTargetCenterPixelCoordinates()
row = coords[1]
column = coords[0]
# get hold of the sky coordinates of the target center :
coords = product.getTargetCenterSkyCoordinates()
ra = coords[0]
dec = coords[1]
# get hold of the target radius in pixels :
print product.getTargetRadiusPixels()
# get hold of the target radius in arcsec :
print product.getTargetRadiusArcsec()
# get hold of the inner radius of the sky annulus in pixels :
print product.getInnerRadiusPixels()
# get hold of the outer radius of the sky annulus in pixels :
print product.getOuterRadiusPixels()
# get hold of the inner radius of the sky annulus in arcsec :
print product.getInnerRadiusArcsec()
# get hold of the outer radius of the sky annulus in arcsec :
print product.getOuterRadiusArcsec()
# check which sky estimation algorithm was used :
print product.getAlgorithm()
# check whether you are using fractional or entire pixels :
print product.getPixels()
# get hold of the intensity unit (i.e. the unit of the image) :
print product.getUnit()
# get hold of the (integrated) flux unit :
print product.getFluxUnit()
# get hold of the results table (you can save it with the asciiTableWriter) :
table = product.getTable()
# get hold of the total flux in the target aperture (incl. background) :
print product.getTargetPlusSkyTotal()
# get hold of the total flux in the target aperture (background subtracted) :
print product.getTargetTotal()
# get hold of the curve of growth as a table (you can save it with the
 asciiTableWriter):
table = product.getCurveOfGrowth()
# plot the curve of growth:
plot = PlotXY(product.getGrowthRadius(), product.getGrowthFlux())
plot.setXtitle("Target radius [pixels]");
plot.setYtitle("Target flux (sky subtr.) [" + product.getFluxUnit() + "]")
# get hold of the sky intensity plot (as a table) and save it (you can save it
 with the asciiTableWriter):
table = product.getSkyIntensityPlot()
# plot the sky intensity plot:
```

Example 1: Script demonstrating the use of an AnnularSkyAperturePhotometryProduct

```
plot = PlotXY(product.getSkyRadius(), product.getSkyIntensity())
plot.setXtitle("Radius [pixels]")
plot.setYtitle("Intensity [" + product.getUnit() + "]")
plot.getSubPlot(0).getLayer(0).getStyle().setLine(0)
```

API Summary

Constructor
AnnularSkyAperturePhotometryProduct The constructor of a new AnnularSkyAperturePhotometryProduct.

Methods
setRadii (double innerPixels, double outerPixels) Setting the inner and outer radius of the sky annulus.
setRadii (double innerPixels, double innerArcsec, double outerPixels, double outerArcsec) Setting the inner and outer radius of the sky annulus.
setSkyIntensityPlot (Double1d radius, Double1d intensity) Setting the sky intensity plot.
double **getInnerRadiusPixels** Returns the inner radius of the sky annulus in pixels.
double **getInnerRadiusArcsec** Returns the inner radius of the sky annulus in arcsec.
double **getOuterRadiusPixels** Returns the outer radius of the sky annulus in pixels.
double **getOuterRadiusArcsec** Returns the outer radius of the sky annulus in arcsec.
TableDataset **getSkyIntensityPlot** Returns the sky intensity plot for this AnnularSkyAperturePhotometryProduct.
Double1d **getSkyRadius** Returns the sky radius in pixels for the sky intensity plot.
Double1d **getSkyIntensity** Returns the sky intensity for the sky intensity plot.

API Details

Constructor

AnnularSkyAperturePhotometryProduct
The constructor of a new AnnularSkyAperturePhotometryProduct. A new SkyAperturePhotometryProduct is constructed.

Methods

setRadii (double innerPixels, double outerPixels)
Setting the inner and outer radius of the sky annulus.

setRadii (double innerPixels, double outerPixels)

Sets the inner and outer radius of the sky annulus in pixels for this AnnularSkyAperturePhotometryProduct to the given numbers of pixels.

Arguments

`double` **innerPixels** `[INPUT, MANDATORY, default=no default value]`

The inner radius in pixels, as double

`double` **outerPixels** `[INPUT, MANDATORY, default=no default value]`

The outer radius in pixels, as double

setRadii (double innerPixels, double innerArcsec, double outerPixels, double outerArcsec)

Setting the inner and outer radius of the sky annulus.

Sets the inner and outer radius of the sky annulus for this AnnularSkyAperturePhotometryProduct to the given numbers of pixels and arcsec.

Arguments

`double` **innerPixels** `[INPUT, MANDATORY, default=no default value]`

The inner radius in pixels, as double

`double` **innerArcsec** `[INPUT, MANDATORY, default=no default value]`

The inner radius in arcsec, as double

`double` **outerPixels** `[INPUT, MANDATORY, default=no default value]`

The outer radius in pixels, as double

`double` **outerArcsec** `[INPUT, MANDATORY, default=no default value]`

The outer radius in arcsec, as double

setSkyIntensityPlot (Double1d radius, Double1d intensity)

Setting the sky intensity plot.

Sets the sky intensity plot for this AnnularSkyAperturePhotometryTask.

Arguments

`Double1d` **radius** `[INPUT, MANDATORY, default=no default value]`

The inner radius for the sky intensity plot, as Double1d

`Double1d` **intensity** `[INPUT, MANDATORY, default=no default value]`

The intensities estimated within an annulus with the specified inner radii and a fixed outer radius, as Double1d

double **getInnerRadiusPixels**

Returns the inner radius of the sky annulus in pixels.

Returns the inner radius of the sky annulus for this AnnularSkyAperturePhotometryProduct in pixels.

Return

double

The inner radius of the sky annulus for this AnnularSkyAperturePhotometryProduct in pixels.

double **getInnerRadiusArcsec**

Returns the inner radius of the sky annulus in arcsec.

double getInnerRadiusArcsec

Returns the inner radius of the sky annulus for this AnnularSkyAperturePhotometryProduct in arcsec.

Return

double

The inner radius of the sky annulus for this AnnularSkyAperturePhotometryProduct in arcsec.

double getOuterRadiusPixels

Returns the outer radius of the sky annulus in pixels.

Returns the outer radius of the sky annulus for this AnnularSkyAperturePhotometryProduct in pixels.

Return

double

Tthe outer radius of the sky annulus for this AnnularSkyAperturePhotometryProduct in pixels.

double getOuterRadiusArcsec

Returns the outer radius of the sky annulus in arcsec.

Returns the outer radius of the sky annulus for this AnnularSkyAperturePhotometryProduct in arcsec.

Return

double

The outer radius of the sky annulus for this AnnularSkyAperturePhotometryProduct in arcsec.

TableDataset getSkyIntensityPlot

Returns the sky intensity plot for this AnnularSkyAperturePhotometryProduct.

Returns a table with the sky intensity plot.

Return

TableDataset

The sky intensity plot for this AnnularSkyAperturePhotometryProduct.

Double1d getSkyRadius

Returns the sky radius in pixels for the sky intensity plot.

Returns the sky radius in pixels for the sky intensity plot for this AnnularSkyAperturePhotometryProduct.

Return

Double1d

The sky radius in pixels for the sky intensity plot for this AnnularSkyAperturePhotometryProduct.

Double1d getSkyIntensity

Returns the sky intensity for the sky intensity plot.

Returns the sky intensity for the sky intensity plot for this AnnularSkyAperturePhotometryProduct.

Return

Double1d **getSkyIntensity**
Double1d The sky intensity for the sky intensity plot for this AnnularSkyAperturePhotometryProduct.

See also

- Developers Manual: herschel.ia.dataset.image.AnnularSkyAperturePhotometryProduct

1.16. ANY

Full Name:	herschel.ia.numeric.toolbox.basic.Any
Alias:	ANY
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Any
Category	Arrays and datasets/Element selection

Description

Determines whether any element of an array satisfies an expression.

This function can return a global `True` or `False` result for the whole array, or test the expression along each dimension of a multidimensional array if the `dimension` parameter is specified. For example, you could test an expression on all the rows, or all the columns, of a two-dimensional array. See also the example below.

Example

Example 1: Apply ANY to an Int2d: check if any of the array items, in the specified dimension, is lower than 3.

```
x = Int2d( [ [1,2], [3,4], [5,6] ])
print ANY(x&lt;3)    # Any item =&gt; 1 (true)
print ANY(x&lt;3, 0) # Dim 0 =&gt; [ ANY(Int1d([1,3,5]) &lt; 3),
                   # ANY(Int1d([2,4,6]) &lt; 3) ] = [true, true]
print ANY(x&lt;3, 1) # Dim 1 =&gt; [ ANY(Int1d([1,2]) &lt; 3),
                   # ANY(Int1d([3,4]) &lt; 3), ANY(Int1d([5,6]) &lt; 3) = [true,
 false, false]
```

API Summary

Jython Syntax
<y> = ANY(<array_expression> [,<dimension>])

Properties
Logical expression **array_expression** [INPUT, MANDATORY, default=no default value]
Integer **dimension** [INPUT, OPTIONAL, default=all dimensions]
Boolean or BoolNd **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Logical expression array_expression [INPUT, MANDATORY, default=no default value]

The expression to be tested. You can use any expression involving an array and evaluating to `True` or `False` for each array element.

Integer dimension [INPUT, OPTIONAL, default=all dimensions]

The dimension along which to test the expression.

Boolean or BoolNd y [OUTPUT, MANDATORY, default=no default value]
The result of testing the expression.

See also

- ALL
- Developers Manual: herschel.ia.numeric.toolbox.basic.Any

1.17. AnyPresent

Full Name:	herschel.ia.numeric.toolbox.basic.AnyPresent
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import AnyPresent
Category	Arrays and datasets/Element selection

Description

Checks whether any of the bits in the specified bitmask are present in the elements of the input array.

For example, if the bitmask is 7 (111 in binary) and one array element is 5 (101 in binary), the function returns `True` for that element, because two out of three bits in the mask are also set in the array value.

Example

Example 1: Apply AnyPresent to an Int1d

```
x = Int1d([0,1,2,3])
print AnyPresent(3)(x)
#=&gt; [  (0 &amp; 3) != 0,   (1 &amp; 3) != 0,   (2 &amp; 3) != 0,   (3 &amp;
 3) != 0  ] =  [false,true,true,true]
print x.apply(AnyPresent(3)) # Another way of using AnyPresent
```

API Summary

Jython Syntax

```
&lt;y&gt;=AnyPresent(&lt;bitmask&gt;)(&lt;x&gt;)
```

Properties

`Integer type array x [INPUT, MANDATORY, default=no default value]`

`Integer bitmask [INPUT, MANDATORY, default=no default value]`

`Boolean array y [OUTPUT, MANDATORY, default=no default value]`

API details

Properties

`Integer type array x [INPUT, MANDATORY, default=no default value]`

The input array. It can be of any integer type: ByteNd, ShortNd, IntNd, LongNd.

`Integer bitmask [INPUT, MANDATORY, default=no default value]`

The bitmask.

`Boolean array y [OUTPUT, MANDATORY, default=no default value]`

The result of the comparison. Each array element is `True` if any of the bits of the mask are present in the corresponding input array element, `False` otherwise.

See also

- AllPresent

- NotPresent
- Developers Manual: herschel.ia.numeric.toolbox.basic.AnyPresent

1.18. ARCCOS

Full Name:	herschel.ia.numeric.toolbox.basic.ArcCos
Alias:	ARCCOS
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import ArcCos
Category	Mathematics/Trigonometry

Description

Returns the inverse cosine of a number or array.

If the input is an array, the output is an array of the same type and size, with inverse cosine values instead of the original elements.

Example

Example 1: Applying ARCCOS to a Float1d

```
x = Float1d([0,0.5])
print  ARCCOS(x) # [1.5707964,1.0471976]
```

API Summary

Jython Syntax

```
&lt;y&gt; = ARCCOS(&lt;x&gt;)
```

Properties

Number or array **x** [INPUT, MANDATORY, default=no default value]

Number or array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Number or array x [INPUT, MANDATORY, default=no default value]

The value or values of which to compute the inverse cosine. All input values must be in the range [-1,1]. Complex numbers are not allowed.

Number or array y [OUTPUT, MANDATORY, default=no default value]

The inverse cosine value or values, in radians.

See also

- COS
- COSH
- Developers Manual: herschel.ia.numeric.toolbox.basic.ArcCos

1.19. ARCCOSH

Full Name:	herschel.ia.numeric.toolbox.basic.ArcCosH
Alias:	ARCCOSH
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import ArcCosH
Category	Mathematics/Trigonometry

Description

Returns the inverse hyperbolic cosine of a number or array.

If the input is an array, the output is an array of the same type and size, with inverse hyperbolic cosine values instead of the original elements.

Example

Example 1: Applying ARCCOSH to a Float1d

```
x=Float1d([0,0.5])
print  ARCCOSH(x) # [2.0,1.31696]
```

API Summary

Jython Syntax

```
&lt;y&gt; = ARCCOSH(&lt;x&gt;)
```

Properties

Number or array x [INPUT, MANDATORY, default=no default value]

Number or array y [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Number or array x [INPUT, MANDATORY, default=no default value]

The value or values of which to compute the inverse hyperbolic cosine. All input values must be in the range [1, infinity] for ARCCOSH and (-1, 1) for ARCTANH. All values are accepted for ARCSINH. Complex numbers are not allowed.

Number or array y [OUTPUT, MANDATORY, default=no default value]

The inverse hyperbolic cosine value or values.

See also

- COSH
- COS
- Developers Manual: herschel.ia.numeric.toolbox.basic.ArcCosH

1.20. ARCSIN

Full Name:	herschel.ia.numeric.toolbox.basic.ArcSin
Alias:	ARCSIN
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import ArcSin
Category	Mathematics/Trigonometry

Description

Returns the inverse sine of a number or array.

If the input is an array, the output is an array of the same type and size, with inverse sine values instead of the original elements.

Example

Example 1: Applying ARCSIN to a Float1d

```
x = Float1d([0,0.5])
print  ARCSIN(x) # [0.0,0.5235988]
```

API Summary

Jython Syntax

```
&lt;y&gt; = ARCSIN(&lt;x&gt;)
```

Properties

Number or array **x** [INPUT, MANDATORY, default=no default value]

Number or array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Number or array x [INPUT, MANDATORY, default=no default value]

The value or values of which to compute the inverse sine. All input values must be in the range [-1,1]. Complex numbers are not allowed.

Number or array y [OUTPUT, MANDATORY, default=no default value]

The inverse sine value or values, in radians.

See also

- SIN
- SINH
- Developers Manual: herschel.ia.numeric.toolbox.basic.ArcSin

1.21. ARCSINH

Full Name:	herschel.ia.numeric.toolbox.basic.ArcSinH
Alias:	ARCSINH
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import ArcSinH
Category	Mathematics/Trigonometry

Description

Returns the inverse hyperbolic sine of a number or array.

If the input is an array, the output is an array of the same type and size, with inverse hyperbolic sine values instead of the original elements.

Example

Example 1: Applying ARCSINH to a Float1d

```
x=Float1d([0,0.5])
print  ARCSINH(x) # [1.0,0.88137358702]
```

API Summary

Jython Syntax

```
&lt;y&gt; = ARCSINH(&lt;x&gt;)
```

Properties

Number or array **x** [INPUT, MANDATORY, default=no default value]

Number or array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Number or array x [INPUT, MANDATORY, default=no default value]

The value or values of which to compute the inverse hyperbolic sine. All input values must be in the range [1, infinity] for ARCCOSH and (-1, 1) for ARCTANH. All values are accepted for ARCSINH. Complex numbers are not allowed.

Number or array y [OUTPUT, MANDATORY, default=no default value]

The inverse hyperbolic sine value or values.

See also

- SINH
- SIN
- Developers Manual: herschel.ia.numeric.toolbox.basic.ArcSinH

1.22. ARCTANCONTFRAC

Full Name:	herschel.ia.numeric.toolbox.basic.ArcTanContfrac
Alias:	ARCTANCONTFRAC
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import ArcTanContfrac
Category	Mathematics/Trigonometry

Description

Returns the inverse tangent of a number or array using Euler's continued fraction approximation.

If the input is an array, the output is an array of double type and the same size, with inverse tangent values instead of the original elements.

The accuracy of the result improves with a the number of iterations. The least accurate result is for an input value x = +/- 1.

Some examples of accuracy, where *n* is the number of iterations. The accuracy is calculated as (ARCTAN(x) - ARCTANCONTFRAC(x))/ARCTAN(x).

- n = 5: 4*10E-5 for x = 1, <4*10E-9 for x = 0.4
- n = 10: 6*10E-9 for x = 1, <1*10E-16 for x = 0.4
- n = 15: 9*10E-13 for x = 1, <1*10E-16 for x = 0.4
- n = 20: 4*10E-16 for x = 1, <1*10E-16 for x = 0.4

Examples

Example 1: Applying ARCTANCONTFRAC to a Float1d

```
x = Float1d([0,0.5])
print  ARCTANCONTFRAC(x) # [0.0,0.4636476]
```

Example 2: Applying ARCTANCONTFRAC to a Float1d using 10 iterations

```
x = Float1d([0,0.5])
print  ARCTANCONTFRAC.PROCEDURE_10.of(x) # [0.0,0.4636476]
```

Example 3: Applying ARCTANCONTFRAC to a Float1d using a custom number of 40 iterations

```
from herschel.ia.numeric.toolbox.basic import ArcTanContfrac
x = Float1d([0,0.5])
print  ArcTanContfrac(40).of(x) # [0.0,0.4636476]
```

API Summary

Jython Syntax

```
&lt;y&gt; = ARCTANCONTFRAC(&lt;x&gt;)
```

Properties

Number or array **x** [INPUT, MANDATORY, default=no default value]

Properties
Integer **n** [INPUT, MANDATORY, default=no default value]
Number or array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Number or array x [INPUT, MANDATORY, default=no default value]

The value or values of which to compute the inverse tangent. Complex numbers are not allowed.

Integer n [INPUT, MANDATORY, default=no default value]

The number of iterations for the continued fraction algoRithm. The default is 15.

Number or array y [OUTPUT, MANDATORY, default=no default value]

The inverse tangent value or values, in radians.

See also

- TAN
- TANH
- Developers Manual: herschel.ia.numeric.toolbox.basic.ArcTanContfrac

1.23. ARCTAN

Full Name:	herschel.ia.numeric.toolbox.basic.ArcTan
Alias:	ARCTAN
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import ArcTan
Category	Mathematics/Trigonometry

Description

Returns the inverse tangent of a number or array.

If the input is an array, the output is an array of the same type and size, with inverse tangent values instead of the original elements.

Example

Example 1: Applying ARCTAN to a Float1d

```
x = Float1d([0,0.5])
print  ARCTAN(x) # [0.0,0.4636476]
```

API Summary

Jython Syntax

```
&lt;y&gt; = ARCTAN(&lt;x&gt;)
```

Properties

Number or array **x** [INPUT, MANDATORY, default=no default value]

Number or array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Number or array x [INPUT, MANDATORY, default=no default value]

The value or values of which to compute the inverse tangent. All input values must be in the range [-1,1]. Complex numbers are not allowed.

Number or array y [OUTPUT, MANDATORY, default=no default value]

The inverse tangent value or values, in radians.

See also

- TAN
- TANH
- Developers Manual: herschel.ia.numeric.toolbox.basic.ArcTan

1.24. ARCTANH

Full Name:	herschel.ia.numeric.toolbox.basic.ArcTanH
Alias:	ARCTANH
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import ArcTanH
Category	Mathematics/Trigonometry

Description

Returns the inverse hyperbolic tangent of a number or array.

If the input is an array, the output is an array of the same type and size, with inverse hyperbolic tangent values instead of the original elements.

Example

Example 1: Applying ARCTANH to a Float1d

```
x=Float1d([0,0.5])
print  ARCTANH(x) # [0.5,0.549306144334]
```

API Summary

Jython Syntax

```
&lt;y&gt; = ARCTANH(&lt;x&gt;)
```

Properties

Number or array x [INPUT, MANDATORY, default=no default value]

Number or array y [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Number or array x [INPUT, MANDATORY, default=no default value]

The value or values of which to compute the inverse hyperbolic tangent. All input values must be in the range [1, infinity] for ARCCOSH and (-1, 1) for ARCTANH. All values are accepted for ARCSINH. Complex numbers are not allowed.

Number or array y [OUTPUT, MANDATORY, default=no default value]

The inverse hyperbolic tangent value or values.

See also

- TANH
- TAN
- Developers Manual: herschel.ia.numeric.toolbox.basic.ArcTanH

1.25. ArctanModel

Full Name:	herschel.ia.numeric.toolbox.fit.ArctanModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import ArctanModel
Category	Mathematics/Fitting

Description

ArcTangus Model.

$$f(x:p) = p_0 * \arctan(p_1 * (x - p_2))$$

where p_0 = amplitude, p_1 = slope and p_2 = offset. As always x = input.

The parameters are initialized at {2/pi, 1.0, 0.0}. It is a non-linear model.

See example

Example

Example 1: ArctanModel

```
arct = ArctanModel( )            # arctangus
print arct.getNumberOfParameters()        # 3
```

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.ArctanModel

1.26. ArrayAssistant

Full Name:	herschel.ia.numeric.toolbox.fit.ArrayAssistant
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import ArrayAssistant
Category	Mathematics/Fitting

Description

ArrayAssistant contains 2 methods to assist with more dimensional fitting.

1. getIndices:

   Generates indices for data arrays of any dimension.

   To be used as input in the Fitter classes.

2. resizeData:

   Resizes the data arrays into a 1-dimensional array.

   To be used as data in the Fitter.

Example

Example 1: ArrayAssistant

```
# Suppose y is a 2-dimensional map of something
y = Double2d(100,100)                    # some tbd map
aass = ArrayAssistant()
input = aass.getIndices( y )             # input values derived from map y
some2dModel = PolySurfaceModel( 2 )        # some 2d model
fitter = Fitter( input, some2dModel )
y1d = aass.resizeData( y )                # re-arranges the map into a 1d form
pars = fitter.fit( y1d )                 # needed in the fitter
yfit = some2dModel( input )                # result is a Double1d
yfit2d = aass.resizeData( yfit, y )        # transform back into map, same size
 as y
```

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.ArrayAssistant

1.27. asciiTableReader

Full Name:	herschel.ia.toolbox.util.AsciiTableReaderTask
Alias:	asciiTableReader
Type:	Java Task -
Import:	from herschel.ia.toolbox.util import AsciiTableReaderTask

Description

Task for reading ASCII table files

Task for reading ASCII files consisting of columns of numbers (or other data) with an optional header describing the columns. The result is placed into a TableDataset (herschel.ia.dataset.TableDataset). An extensive tutorial for table reading and writing is available in the DAG (see Exchanging data with text files in the "See also" section at the end of this page).

General priorities of optional parameters

- **tableType** has precedence over everything: if it is set to other than ADVANCED (default) only **skipRows** is taken into account (and null parameters)
- if **readConfigFile** is used (and **tableType** left to ADVANCED) then everything else is ignored (except **writeConfigFile**)
- if a **parser** is defined (only available via command line) (tableType left to ADVANCED and readConfigFile not provided) then, all parser related parameters will be ignored (**ignorePattern**, **skipRows**, **columnType**, **delimiter** and **columnNames**)
- if no **parser** is defined (tableType left to ADVANCED and readConfigFile not provided) then parser related parameters will be taken into account (**ignorePattern**, **skipRows**, **columnType**, **delimiter** and **columnNames**)
- if a **template** is passed then no header will be read (no matter what is the value of other parameters: **columnType**, **columnNames** and the respective configuration of a **parser**, if passed, are ignored)

By default, if only the **file** parameter is passed, the task will try to read a CSV table. If **nulls** (only available via command line) is passed, **addUserNulls** will be ignored. If the file to be read contains a header and a **template** is provided, HIPE will complain. The solution is to use **skipRows** for the amount of lines needed to start at the table data.

The tableType parameter supports these values: "CSV", "SPACES", "SEXTRACTOR", "IPAC", "FIXED" and "ADVANCED", these map to the types of tables they can read:

- SPACES: An ASCII file consisting of data items separated by spaces. **tbl** file extension is recognised as being a table of this type in the Hipe Navigator. Except for special values, this reads other formats too (like TopCat ASCII).
- CSV : An ASCII file consisting of an optional header and data items separated by commas, **csv** file extension is recognised as being a table of this type in the Hipe Navigator.
- SEXTRACTOR : An ASCII file consisting of a header with a line per column and data separated by whitespace. Vectors are represented with a column per coordinate (not as 3d data).
- IPAC : An ASCII file consisting of keywords, comments, a header and fixed-width data. Comments and keywords are ignored, types are just checked (Herschel types are used) and empty values are just documented.

- FIXED : An ASCII file consisting of data items starting and finishing in fixed column positions (will use whitespace if needed).
- ADVANCED : (default) Capable of reading any type of tabular data. Requires filling up the rest of inputs as needed.

Two types of (optional) headers are supported automatically:

- Column name header, a line started with #, with the names separated with TableType separators ("," for CSV and spaces for Spaces tables)
- Herschel header, a 4 line header, each line with as much elements as columns, **separated with TableType separators** (or the same separator that your parser will use in the case of ADVANCED):
  - Column names (can have no data)
  - Column type (must have data), possible values are: "Boolean", "Byte", "Short", "Integer", "Long", "Float", "Double", "Complex" and "String"
  - Units (can have no data)
  - Descriptions (can have no data)

  So, a Herschel header must, at least, give the types of the data of all columns. Note that even if a line has no data **the separators must appear** (as the same mechanism that reads the data reads the header). In the case of spaces, something that is not a space must be present for each column. To enable header guessing in the case of ADVANCED tables, you need to use **columnType** with a GUESS_* value. In the case of fixed width tables the header will be in fixed width format. See examples below.

Sample data for SPACES option

(Without a header) You can see that the amount of space that needs to separate data is free.

```
300    2.242516943
310  2.24262218
```

(With a Herschel header) SPACES files can also have a Herschel header but it must be complete, for example:

```
Frequency Flux
Double    Double
GHz []
fixed_grid Neptune
300        2.242516943
310        2.24262218
```

Notes:

- Note that the header elements are also separated by any amount of space
- Note the underscore in the first column description (as a space would be interpreted as the end of the header cell).
- Note that a null or empty unit can be written as []

(With a column name header) SPACES files can also have a column names header, for example:

```
# Frequency   Flux
300        2.242516943
```

```
310        2.24262218
```

Note that the column names are separated with any number of spaces.

Sample data for `CSV` option

(Without a header) CSV file without any header

```
1249423389383,26318.0
1249423901383,26321.0
1249424413383,26324.0
```

Note that in CSV files leading and trailing space may be significant (read as part of cells with text content).

(With a column names header) CSV file with column names header

```
#finetime,KD232303
1249423389383,26318.0
1249423901383,26321.0
1249424413383,26324.0
```

Note that the header elements are separated with just a ","

(With a full Herschel header) CSV file with full headers and just two data rows follows:

```
Frequency,Flux
Double,Double
GHz,Jy
fixed grid,Neptune
300,2.242516943
310,2.24262218
```

(With a minimal Herschel header) A sample CSV file with the minimal (just the types) Herschel header would be:

```
,
Double,Double
,
,
300,2.242516943
310,2.24262218
```

Note that the commas must always be there or you will get "Column index is out of range" errors.

Some advanced tips about the reading process

There are several points where the tool may fail to process your data. To better diagnose the problem it is recommended that you increase the logging in modules **ia_toolbox_util** and **ia_io**'s ascii :

1. **Locating the start of your table** : the following parameters are useful to guide this:
   - **skipRows** to tell the tool to ignore the first skipRows lines
   - **ignorePattern** to tell the tool about comment (non data) lines
   - **ignoreWarn** to see log messages about the ignored lines
2. **Reading the header of the table** : there are too many header formats and the tool may not understand yours. The following parameters are helpful for dealing with header problems:

- **skipRows** to tell the tool to start reading after the header
- **tableType** (for some values like "SEXTRACTOR", "IPAC" or SPACES for ASCII Topcat) it can cope directly with those table headers
- **template** to pass a table header built by code in HIPE (TableTemplate)

3. **Reading some special columns (null treatment)** : In some cases, if the tool finds some strange values in columns, it will read the column data as Strings. Here are some sample cases (we are working on adding functionality to help in these cases):
   - **"Null" values** : The tool will create a column of Strings (String1d) if some of the values cannot be mapped to any other known type. The null-related parameters (Nulls tab in the GUI, plus others available via command line) allow to configure how to deal with special values: the parameters **nulls** or **addUserNulls** are used for nulls identification. To replace the recognised nulls with a correct value of the column type the **null<Type>** parameters are used . As a last resort, and if you want to preserve the input exactly, **columnType**'s "ALL_STRING" can be used to tell the tool not to try to guess the types when there are special values.
   - **Empty values or not enough values in one line** : this is similar to the previous case, but this can also cause the reading to fail if the tool sees that the data as not rectangular. You should fill every empty value with something before reading. One useful technique is to add an additional column to the table that tags the structure of the data in some row (to know which values to ignore). Passing "FIXED" in **tableType** or (if you are not using the command line), passing a fixed-width parser (FixedWidthParser) can help in some cases.
   - **Too many values in one line** : in some cases you can use quotes to avoid that some cells are split as multiple cells. Also using a fixed-width parser (if possible) can help to avoid this problem.
4. **Others:**
   - You can import and use some existing Python libraries for some specific formats
   - If all else fails, you can convert your table into a FITS file and use fitsReader to read it.

Examples

Example 1: Reading a file with fields separated by commas (either without header, a one line header or a Herschel header)

```
filepath = "path_to_file/filename"
table = asciiTableReader(file=filepath, tableType="CSV")
```

Example 2: Read a raw space-separated ASCII table filling up advanced parameters

```
# test_space.dat contents
## start
#1 2 3
#
#2 3 4
#5 6 7
#
#
## end
from herschel.ia.io.ascii import AsciiParser
tds = asciiTableReader(file="test_space.dat",
  delimiter=" ",
  ignorePattern= '^\s*$|^\s*#',
  ignoreWarn= True,
  columnType=AsciiParser.GUESS_TRY)
# the ignorePattern value will match lines with just spaces or empty and,
# also lines that just contain #-started comment lines
```

Example 2: Read a raw space-separated ASCII table filling up advanced parameters

```
# tds will hold a TableDataset with the data, blank lines will be ignored,
# and warnings will be issued for ignored lines
print tds
#{description="", meta=[], columns=[c0, c1, c2]}
print tds.getColumn(0)
#{description="", data=[1,2,5], unit=null}
print tds.getColumn(1)
#{description="", data=[2,3,6], unit=null}
print tds.getColumn(2)
#{description="", data=[3,4,7], unit=null}
```

Example 3: Reading a CVS file with a Herschel header

```
# file.ascii contents
## start:
# c0,c1,c2
# Byte,Byte,Byte
# ,,
# ,,
# 1,2,3
# 2,3,4
# 5,6,7
## end
tableDataset = asciiTableReader(file="file.ascii")
print tableDataset
#{description="", meta=[], columns=[c0, c1, c2]}
print tableDataset.getColumn(0)
#{description="", data=[1,2,5], unit=null}
print tableDataset.getColumn(1)
#{description="", data=[2,3,6], unit=null}
print tableDataset.getColumn(2)
#{description="", data=[3,4,7], unit=null}
```

Example 4: Read a file with an invalid header, delimiter is tab

```
# file.ascii contents
## start:
# Byte    Byte    Byte
#
#
# 1       2       3
# 2       3       4
# 5       6       7
## end
tableDataset = asciiTableReader(file='file.ascii', delimiter='\t',
 columnType=AsciiParser.GUESS_TRY)
print tableDataset
#{description="", meta=[], columns=[c0, c1, c2]}
# All columns are String1d, and "Byte" appear as data
print tableDataset.getColumn(0)
#{description="", data=["Byte","null","null","1","2","5"], unit=null}
print tableDataset.getColumn(1)
#{description="", data=["Byte","null","null","2","3","6"], unit=null}
print tableDataset.getColumn(2)
#{description="", data=["Byte","null","null","3","4","7"], unit=null}
```

Example 5: Reading a TopCat ASCII table (SPACES)

```
# topcat.ascii contents
## start:
## ra            dec
#  194.9440701   28.0733714
#  195.0409305   28.1142592
## end
tableDataset = asciiTableReader(file="topcat.ascii", tableType="SPACES")
print tableDataset
```

Example 5: Reading a TopCat ASCII table (SPACES)

```
#{description="", meta=[], columns=[ra, dec]}
print tableDataset.getColumn(0)
#{description="", data=[194.9440701,195.0409305], unit=null}
print tableDataset.getColumn(1)
#{description="", data=[28.0733714,28.1142592], unit=null}
```

Example 6: Reading a SExtractor table

```
# sextractor.ascii contents
## start:
##   1 NUMBER                Running object number
          []
##   2 ALPHA_J2000           Right ascension of barycenter (J2000)
          [deg]
#         1 175.1835742
#         2 175.2211794
## end
tableDataset = asciiTableReader(file="sextractor.ascii",
 tableType="SEXTRACTOR")
print tableDataset
#{description="", meta=[], columns=[NUMBER, ALPHA_J2000]}
print tableDataset.getColumn(0)
#{description="Running object number", data=[1,2], unit=null}
print tableDataset.getColumn(1)
#{description="Right ascension of barycenter (J2000)",
 data=[175.1835742,175.2211794], \
# unit=deg [1 deg = 0.017453292519943295 rad]}
```

Example 7: Reading an IPAC table

```
# ipac.ascii contents
## start:
#\ ___ Ks photometric uncer...
#\
#|         ra|        dec|
#|     double|     double|
#|        deg|        deg|
#|       null|       null|
# 194.9440701  28.0733714
## end
tableDataset = asciiTableReader(file="ipac.ascii", tableType="IPAC")
print tableDataset
#{description="", meta=[], columns=[ra, dec]}
print tableDataset.getColumn(0)
#{description="Empty values appear as null", data=[194.9440701], unit=deg [1
 deg = 0.017453292519943295 rad]}
print tableDataset.getColumn(1)
#{description="Empty values appear as null", data=[28.0733714], unit=deg [1 deg
 = 0.017453292519943295 rad]}
```

Example 8: Reading a fixed width table mapping nulls

```
# fixed.ascii contents:
## start:
#1   text   1.1    false
#NAN none   blah   nan
#2   char   3.33   true
## end
tableDataset = asciiTableReader(file="fixed.ascii", tableType="FIXED",
 addUserNulls="blah")
print tableDataset
#{{description="", meta=[], columns=[c0, c1, c2, c3]
print tableDataset.getColumn(0)
#{description="", data=[1,0,1], unit=null}
print tableDataset.getColumn(1)
#{description="", data=["text","null","char"], unit=null}
```

Example 8: Reading a fixed width table mapping nulls

```
print tableDataset.getColumn(1).data[1]
#None
print tableDataset.getColumn(2)
#{description="", data=[1.1,NaN,3.33], unit=null}
print tableDataset.getColumn(3)
#{description="", data=[false,false,true], unit=null}
```

API Summary

Jython Syntax

Full Signature:

table = asciiTableReader(<file> [, <tableType>="ADVANCED", <readConfigFile>, <writeConfigFile>, <parser>,

<ignorePattern>= "^\\s*#", <ignoreWarn>=False, <skipRows>, <columnType>="GUESS_ALL", <delimiter>=",",

<columnNames>=False, <template>, <nulls>, <addUserNulls>, <nullBoolean>=False, <nullByte>=0,

<nullShort>=0, <nullInteger>=0, <nullLong>=0L, <nullFloat>=Float.NaN, <nullDouble>=Double.NaN, <nullComplex>,

<nullString>])

Null-related parameters (nulls has priority over addUserNulls, null<Type>s over template nulls):

<nulls>, <addUserNulls>, <nullBoolean>=False, <nullByte>=0, <nullShort>=0, <nullInteger>=0, <nullLong>=0L,

<nullFloat>=Float.NaN, <nullDouble>=Double.NaN, <nullComplex>, <nullString>

Basic (TableType is different from ADVANCED):

table=asciiTableReader(<file>, <tableType> [, <skipRows>, ...null-related...])

Advanced (tableType=ADVANCED, default):

Reading a configuration (readConfigFile):

table=asciiTableReader(<file>, <readConfigFile> [, <writeConfigFile>])

With a (configured) parser:

table = asciiTableReader(<file> [, <writeConfigFile>], <parser> [, <ignoreWarn>= False,

<template>, ...null-related...]])

Without a parser (the system will internally configure one):

table = asciiTableReader(<file> [, <writeConfigFile>]

[<ignorePattern>="^\\s*#", <ignoreWarn>=False, <skipRows>, <columnType>="GUESS_ALL", <delimiter>=",",

Jython Syntax
<columnNames>=False, <template>, ...null-related...]])

Properties
String **file** [INPUT, MANDATORY, default=no default value]
String **tableType** [INPUT, OPTIONAL, default="ADVANCED"]
String **table** [OUTPUT, MANDATORY, default=no default value]
String **readConfigFile** [INPUT, OPTIONAL, default=no default value]
String **writeConfigFile** [INPUT, OPTIONAL, default=no default value]
AsciiParser **parser** [INPUT, OPTIONAL, default=no default value]
String **ignorePattern** [INPUT, OPTIONAL, default='^\s*#' (ignores # comment lines)]
Boolean **ignoreWarn** [INPUT, OPTIONAL, default=false (default AsciiTableTool.warnWhenIgnore value: false)]
Integer **skipRows** [INPUT, OPTIONAL, default=no default value (do not skip)]
Object **columnType** [INPUT, OPTIONAL, default='GUESS_ALL']
String **delimiter** [INPUT, OPTIONAL, default="," (comma separator)]
Boolean **colunmNames** [INPUT, OPTIONAL, default=False]
TableTemplate **template** [INPUT, OPTIONAL, default=no default]
TypeNullDefinitions **nulls** [INPUT, OPTIONAL, default=no default]
String **addUserNulls** [INPUT, OPTIONAL, default=no default]
Boolean **nullBoolean** [INPUT, OPTIONAL, default=False]
Byte **nullByte** [INPUT, OPTIONAL, default=0]
Short **nullShort** [INPUT, OPTIONAL, default=0]
Integer **nullInteger** [INPUT, OPTIONAL, default=0]
Long **nullLong** [INPUT, OPTIONAL, default=0]
Float **nullFloat** [INPUT, OPTIONAL, default=Float.NaN]
Double **nullDouble** [INPUT, OPTIONAL, default=Double.NaN]
Complex **nullComplex** [INPUT, OPTIONAL, default=no default]
String **nullString** [INPUT, OPTIONAL, default=no default]

Limitations

- Only rectangular (same columns in all the rows) tables can be read. Holes (null Strings or Complex) in non-CSV tables (or any format that does not have explicit cell delimiters) are not supported.
- If the options in tableType do not read your file as desired, you will have to use the ADVANCED option (and the rest of parameters).
- To read as fixed-width you need to pass a FixedWidthParser
- If you use a TableTemplate null recognition and translation will be based solely on the template
- If your data is separated by spaces, do not try to manually add a Herschel table header to the file: you would have to add a lot of dummy data and spaces to satisfy the parser

API details

Properties

String file [INPUT, MANDATORY, default=no default value]

Input file.

String tableType [INPUT, OPTIONAL, default="ADVANCED"]

The type of table in the input file. Valid values are "CSV", "SPACES", "SEXTRACTOR", "IPAC", "FIXED" and "ADVANCED". Only if ADVANCED is used the rest of the input parameters will be taken into account.

String table [OUTPUT, MANDATORY, default=no default value]

TableDataset object loaded from the file.

String readConfigFile [INPUT, OPTIONAL, default=no default value]

Configuration file where the formatter (AsciiFormatter), parser (AsciiParser) and table template (TableTemplate) must be specified. When readConfigFile parameter is specified, any parameter related to parser or to table template are not allowed.

String writeConfigFile [INPUT, OPTIONAL, default=no default value]

If a file is specified, an output configuration file will be created.

AsciiParser parser [INPUT, OPTIONAL, default=no default value]

AsciiParser object used to extract cells from Herschel header and data table, the rest of parse* parameters will modify it. If parser is used: ignorePattern, ignoreWarn, skipRows, delimiter and ColumnNames are ignored. Providing a parser you can control in detail how to read the data.

String ignorePattern [INPUT, OPTIONAL, default='^\s*#' (ignores # comment lines)]

Identifies comment lines. If a line contains this text or matches this regular expression, the line will be ignored. AsciiParser.setIgnore(ignorePattern) is called, if not passed, the default behaviour is to ignore # comment lines. Pre-loaded values in the GUI:

- "^\\s*#" (default): comment lines started with #, i.e.

```
# a comment text
```

- "^\\s*$" (empty lines): empty lines or lines just with whitespace (spaces, tabs ...), i.e.

```

```

- "^\\s*$|^\\s*#" (the combination of the previous two, with a |): empty or comment lines

Boolean ignoreWarn [INPUT, OPTIONAL, default=false (default AsciiTableTool.warnWhenIgnore value: false)]

Specifies if the parser must issue a warning if a line is ignored (emit only if true). AsciiTableTool.setWarnWhenIgnore(expression) is called.

Integer skipRows [INPUT, OPTIONAL, default=no default value (do not skip)]

Number of rows to skip when reading a file. AsciiParser.setSkip(number of lines) is called.

Object columnType [INPUT, OPTIONAL, default='GUESS_ALL']

Specifies if the parser should guess column types. If passed GUESS_NONE, the file should contain some header (or you need to pass a template) (use skipRows for skipping HCSS header or comment these lines), as an HCSS header already contains the types of the columns. AsciiParser.setGuess(guessType) is called. Type is Integer or String (constant name).

Valid options:

- 0 - AsciiParser.GUESS_NONE: (default) file must contains template or template must be provided (no guess)
- 1 - AsciiParser.GUESS_TRY: guess types based on the first 100 rows
- 2 - AsciiParser.GUESS_ALL: guess types based on all rows
- 11 - AsciiParser.ALL_STRING: each cell is a string (no guess required)
- 12 - AsciiParser.ALL_BOOLEAN: each cell is a boolean (no guess required)
- 13 - AsciiParser.ALL_BYTE: each cell is a byte (two's complement 8-bit (signed) value, no guess required)
- 14 - AsciiParser.ALL_SHORT: each cell is a short (two's complement 16-bit (signed) value, no guess required)
- 15 - AsciiParser.ALL_INTEGER: each cell is an integer (two's complement 32-bit (signed) value, no guess required)
- 16 - AsciiParser.ALL_LONG: each cell is a long (two's complement 64-bit (signed) value, no guess required)
- 17 - AsciiParser.ALL_FLOAT: each cell is a float (IEEE 754's 32-bit floating point, no guess required)
- 18 - AsciiParser.ALL_DOUBLE: each cell is a double (IEEE 754's 64-bit floating point, no guess required)
- 19 - AsciiParser.ALL_COMPLEX: each cell is a complex (no guess required)

String delimiter [INPUT, OPTIONAL, default="," (comma separator)]

Specifies the field separator. If it is one character, a CsvParser is selected (CSV format). If it is longer (an expression), a RegExpParser is selected (variable width columns). Pre-loaded values in the GUI:

- "," (default): the values are separated by commas, i.e.

```
1,2,3
```

- "\\t+" (tabs): the values are separated by tabs, i.e.

```
1    2    3
```

- "\\s+" (any whitespace, includes the previous): the values are separated by any mix of spaces & tabs, i.e.

String delimiter [INPUT, OPTIONAL, default="," (comma separator)]

```
1 2   3
```

Boolean columnNames [INPUT, OPTIONAL, default=False]

Specifies if the parser must read column names when reading a file. AsciiParser.firstRowAreColumnNames(expression) is called.

TableTemplate template [INPUT, OPTIONAL, default=no default]

Specifies the HCSS Header for the data (file has no header and no guessing is done)

TypeNullDefinitions nulls [INPUT, OPTIONAL, default=no default]

Specify all the values that will be interpreted as null when reading the file (not shown in GUI).

String addUserNulls [INPUT, OPTIONAL, default=no default]

Comma-separated string that specifies additional values that will be interpreted as null when reading the file (ignored if nulls is passed).

Boolean nullBoolean [INPUT, OPTIONAL, default=False]

Value to replace nulls with in Boolean1d columns (not shown in GUI).

Byte nullByte [INPUT, OPTIONAL, default=0]

Value to replace nulls with in Byte1d columns (not shown in GUI).

Short nullShort [INPUT, OPTIONAL, default=0]

Value to replace nulls with in Short1d columns (not shown in GUI).

Integer nullInteger [INPUT, OPTIONAL, default=0]

Value to replace nulls with in Integer1d columns.

Long nullLong [INPUT, OPTIONAL, default=0]

Value to replace nulls with in Long1d columns.

Float nullFloat [INPUT, OPTIONAL, default=Float.NaN]

Value to replace nulls with in Float1d columns.

Double nullDouble [INPUT, OPTIONAL, default=Double.NaN]

Value to replace nulls with in Double1d columns.

Complex nullComplex [INPUT, OPTIONAL, default=no default]

Value to replace nulls with in Complex1d columns (by default null will be used) (not shown in GUI).

String nullString [INPUT, OPTIONAL, default=no default]

Value to replace nulls with in String1d columns (by default null will be used).

See also

- asciiTableWriter

- Chapter 2 in *Data Analysis Guide*
- Section 2.28 in *Data Analysis Guide*
- http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html
- http://irsa.ipac.caltech.edu/applications/DDGEN/Doc/ipac_tbl.html
- http://www.stecf.org/software/PYTHONtools/astroasciidata/manual/asciidata/node2.html
- Developers Manual: herschel.ia.toolbox.util.AsciiTableReaderTask

History

- `2007-11-22 - JCS`: initial version
- `2008-08-22 - JDS`: added optional parameter parserIgnoreWarn (SCR HCSS-3674), better jython doc
- `2011-02-15 - JDS`: Added tableType (simpleAsciiTableReader deprecation)
- `2012-09-15 - JDS`: Added additional formats (SExtractor, IPAC)
- `2012-10-15 - JDS`: Review of URM
- `2013-05-22 - JDS`: parameter rename, changed priorities
- `2013-07-22 - JDS`: null conversion
- `2013-08-23 - JDS`: using rules (different call forms)

1.28. asciiTableWriter

Full Name:	herschel.ia.toolbox.util.AsciiTableWriterTask
Alias:	asciiTableWriter
Type:	Java Task -
Import:	from herschel.ia.toolbox.util import AsciiTableWriterTask
Category	Input-output

Description

Task for writing TableDatasets into ASCII (text) files.

Only TableDatasets without empty or multidimensional columns can be written to text files (your table must be rectangular). You can also pass a Product with just a complying TableDataset. An extensive tutorial for table reading and writing is available in the DAG (see Exchanging data with text files in the "See also" section at the end of this page).

General priorities of optional parameters

- if **readConfigFile** is defined then the rest of the optional parameters should not be used (except for **writeConfigFile** and **warn**) (it will abort if formatter or template are also passed)
- if **formatter** is defined and given to the task via command line (with **readConfigFile** not provided) then **tableType**, **writeHeader**, **writeMetadata** and **metadataPrefix** will be ignored
- if **readConfigFile** and **formatter** are not provided, all the other optional parameters are taken into account (default behaviour)

When using a **template**, the names must match the names of the columns in the table. A template allows not only to replace the header but, also, to exclude columns and to reorder them. If you want to reorder the rows of a table, you can use **sortTable** before writing it with this task.

Examples

Example 1: writes a TableDataset, with a header, in a file

```
tds = TableDataset()
l1 = Long1d([1,2,3,4,5])
f1 = Float1d([100.0,101.0,102.0,103.0,104.0])
s1 = String1d(['a','b','c','d','e'])
tds.addColumn("Table Column
 0",Column(l1,herschel.share.unit.Power.WATTS,"column_1 Description"))
tds.addColumn("Table Column
 1",Column(f1,herschel.share.unit.Frequency.MEGAHERTZ,"column_2 Description"))
tds.addColumn("Table Column
 2",Column(s1,herschel.share.unit.Unit.parse('SpecialUnit'),"column_3
 Description"))
asciiTableWriter(file="file.ascii",table=tds)
#it generates the following output:
# Table Column 0,Table Column 1,Table Column 2
# Long,Float,String
# W,MHz,SpecialUnit
# column_1 Description,column_2 Description,column_3 Description
# 1,100.0,a
# 2,101.0,b
# 3,102.0,c
# 4,103.0,d
# 5,104.0,e
```

Example 2: writes a TableDataset, Metadata and description included, in a file

```
tds = TableDataset()
l1 = Long1d([1,2,3,4,5])
f1 = Float1d([100.0,101.0,102.0,103.0,104.0])
s1 = String1d(['a','b','c','d','e'])
tds.addColumn("Table Column
 0",Column(l1,herschel.share.unit.Power.WATTS,"column_1 Description"))
tds.addColumn("Table Column
 1",Column(f1,herschel.share.unit.Frequency.MEGAHERTZ,"column_2 Description"))
tds.addColumn("Table Column
 2",Column(s1,herschel.share.unit.Unit.parse('SpecialUnit'),"column_3
 Description"))
tds.meta['meta_0'] = StringParameter("meta_0 value", "meta_0 Description")
tds.meta['meta_1'] = LongParameter(5,"meta_1
 Description",herschel.share.unit.Temperature.CELSIUS)
asciiTableWriter(file="file.ascii",table=tds, writeMetadata=True)
#it generates the following output:
# # Meta data {
# #  meta_0={description="meta_0 Description", string="meta_0 value"}
# #  meta_1={description="meta_1 Description", long=5, unit= C [274.15 K]}
# # } MetaData
# Table Column 0,Table Column 1,Table Column 2
# Long,Float,String
# W,MHz,SpecialUnit
# column_1 Description,column_2 Description,column_3 Description
# 1,100.0,a
# 2,101.0,b
# 3,102.0,c
# 4,103.0,d
# 5,104.0,e
```

Example 3: writes a TableDataset, without header, in a file

```
tds = TableDataset()
l1 = Long1d([1,2,3,4,5])
f1 = Float1d([100.0,101.0,102.0,103.0,104.0])
s1 = String1d(['a','b','c','d','e'])
tds.addColumn("Table Column
 0",Column(l1,herschel.share.unit.Power.WATTS,"column_1 Description"))
tds.addColumn("Table Column
 1",Column(f1,herschel.share.unit.Frequency.MEGAHERTZ,"column_2 Description"))
tds.addColumn("Table Column
 2",Column(s1,herschel.share.unit.Unit.parse('SpecialUnit'),"column_3
 Description"))
tds.meta['meta_0'] = StringParameter("meta_0 value", "meta_0 Description")
tds.meta['meta_1'] = LongParameter(5,"meta_1
 Description",herschel.share.unit.Temperature.CELSIUS)
asciiTableWriter(file="file.ascii",table=tds, writeHeader=False)
#it generates the following output:
# 1,100.0,a
# 2,101.0,b
# 3,102.0,c
# 4,103.0,d
# 5,104.0,e
```

Example 4: writes a TableDataset, without header and with Metadata and description included, in a file

```
tds = TableDataset()
l1 = Long1d([1,2,3,4,5])
f1 = Float1d([100.0,101.0,102.0,103.0,104.0])
s1 = String1d(['a','b','c','d','e'])
tds.addColumn("Table Column
 0",Column(l1,herschel.share.unit.Power.WATTS,"column_1 Description"))
tds.addColumn("Table Column
 1",Column(f1,herschel.share.unit.Frequency.MEGAHERTZ,"column_2 Description"))
tds.addColumn("Table Column
 2",Column(s1,herschel.share.unit.Unit.parse('SpecialUnit'),"column_3
 Description"))
```

Example 4: writes a TableDataset, without header and with Metadata and description included, in a file

```
tds.meta['meta_0'] = StringParameter("meta_0 value", "meta_0 Description")
tds.meta['meta_1'] = LongParameter(5,"meta_1
 Description",herschel.share.unit.Temperature.CELSIUS)
asciiTableWriter(file="file.ascii",table=tds,writeHeader=False,writeMetadata=True)
#it generates the following output:
# # Meta data {
# #  meta_0={description="meta_0 Description", string="meta_0 value"}
# #  meta_1={description="meta_1 Description", long=5, unit= C [274.15 K]}
# # } MetaData
# 1,100.0,a
# 2,101.0,b
# 3,102.0,c
# 4,103.0,d
# 5,104.0,e
```

Example 5: writes a TableDataset, with an specific delimiter, with header, in a file

```
tds = TableDataset()
l1 = Long1d([1,2,3,4,5])
f1 = Float1d([100.0,101.0,102.0,103.0,104.0])
s1 = String1d(['a','b','c','d','e'])
tds.addColumn("Table Column
 0",Column(l1,herschel.share.unit.Power.WATTS,"column_1 Description"))
tds.addColumn("Table Column
 1",Column(f1,herschel.share.unit.Frequency.MEGAHERTZ,"column_2 Description"))
tds.addColumn("Table Column
 2",Column(s1,herschel.share.unit.Unit.parse('SpecialUnit'),"column_3
 Description"))
tds.meta['meta_0'] = StringParameter("meta_0 value", "meta_0 Description")
tds.meta['meta_1'] = LongParameter(5,"meta_1
 Description",herschel.share.unit.Temperature.CELSIUS)
csvFormatter = CsvFormatter()
csvFormatter.delimiter = ' '
asciiTableWriter(file="file.ascii",table=tds,formatter=csvFormatter)
#it generates the following output:
# "Table Column 0" "Table Column 1" "Table Column 2"
# Long Float String
# W MHz SpecialUnit
# "column_1 Description" "column_2 Description" "column_3 Description"
# 1 100.0 a
# 2 101.0 b
# 3 102.0 c
# 4 103.0 d
# 5 104.0 e
```

Example 6: writes a TableDataset, with an specific delimiter, with Header and with Metadata, in a file

```
tds = TableDataset()
l1 = Long1d([1,2,3,4,5])
f1 = Float1d([100.0,101.0,102.0,103.0,104.0])
s1 = String1d(['a','b','c','d','e'])
tds.addColumn("Table Column
 0",Column(l1,herschel.share.unit.Power.WATTS,"column_1 Description"))
tds.addColumn("Table Column
 1",Column(f1,herschel.share.unit.Frequency.MEGAHERTZ,"column_2 Description"))
tds.addColumn("Table Column
 2",Column(s1,herschel.share.unit.Unit.parse('SpecialUnit'),"column_3
 Description"))
tds.meta['meta_0'] = StringParameter("meta_0 value", "meta_0 Description")
tds.meta['meta_1'] = LongParameter(5,"meta_1
 Description",herschel.share.unit.Temperature.CELSIUS)
csvFormatter = CsvFormatter()
csvFormatter.delimiter = ' '
csvFormatter.commented = True
asciiTableWriter(file="file.ascii",table=tds,formatter=csvFormatter)
#it generates the following output:
```

Example 6: writes a TableDataset, with an specific delimiter, with Header and with Metadata, in a file

```
# # Meta data {
# #  meta_0={description="meta_0 Description", string="meta_0 value"}
# #  meta_1={description="meta_1 Description", long=5, unit= C [274.15 K]}
# # } MetaData
# "Table Column 0" "Table Column 1" "Table Column 2"
# Long Float String
# W MHz SpecialUnit
# "column_1 Description" "column_2 Description" "column_3 Description"
# 1 100.0 a
# 2 101.0 b
# 3 102.0 c
# 4 103.0 d
# 5 104.0 e
```

Example 7: writes a TableDataset, with an specific delimiter, without Header, in a file

```
tds = TableDataset()
l1 = Long1d([1,2,3,4,5])
f1 = Float1d([100.0,101.0,102.0,103.0,104.0])
s1 = String1d(['a','b','c','d','e'])
tds.addColumn("Table Column
 0",Column(l1,herschel.share.unit.Power.WATTS,"column_1 Description"))
tds.addColumn("Table Column
 1",Column(f1,herschel.share.unit.Frequency.MEGAHERTZ,"column_2 Description"))
tds.addColumn("Table Column
 2",Column(s1,herschel.share.unit.Unit.parse('SpecialUnit'),"column_3
 Description"))
tds.meta['meta_0'] = StringParameter("meta_0 value", "meta_0 Description")
tds.meta['meta_1'] = LongParameter(5,"meta_1
 Description",herschel.share.unit.Temperature.CELSIUS)
csvFormatter = CsvFormatter()
csvFormatter.delimiter = ' '
csvFormatter.header = False
asciiTableWriter(file="file.ascii",table=tds,formatter=csvFormatter)
#it generates the following output:
# 1 100.0 a
# 2 101.0 b
# 3 102.0 c
# 4 103.0 d
# 5 104.0 e
```

Example 8: writes a TableDataset, with a fixed column width, with Header, in a file

```
tds = TableDataset()
l1 = Long1d([1,2,3,4,5])
f1 = Float1d([100.0,101.0,102.0,103.0,104.0])
s1 = String1d(['a','b','c','d','e'])
tds.addColumn("Table Column
 0",Column(l1,herschel.share.unit.Power.WATTS,"column_1 Description"))
tds.addColumn("Table Column
 1",Column(f1,herschel.share.unit.Frequency.MEGAHERTZ,"column_2 Description"))
tds.addColumn("Table Column
 2",Column(s1,herschel.share.unit.Unit.parse('SpecialUnit'),"column_3
 Description"))
tds.meta['meta_0'] = StringParameter("meta_0 value", "meta_0 Description")
tds.meta['meta_1'] = LongParameter(5,"meta_1
 Description",herschel.share.unit.Temperature.CELSIUS)
fixedWidthFormatter = FixedWidthFormatter()
fixedWidthFormatter.commented = True
fixedWidthFormatter.sizes = [25,25,25]
asciiTableWriter(file="file.ascii",table=tds,formatter=fixedWidthFormatter)
#it generates the following output:
# # Meta data {
# #  meta_0={description="meta_0 Description", string="meta_0 value"}
# #  meta_1={description="meta_1 Description", long=5, unit= C [274.15 K]}
# # } MetaData
# Table Column 0           Table Column 1           Table Column 2
```

Example 8: writes a TableDataset, with a fixed column width, with Header, in a file

```
# Long                    Float                   String
# W                       MHz                     SpecialUnit
# column_1 Description    column_2 Description    column_3 Description
# 1                       100.0                   a
# 2                       101.0                   b
# 3                       102.0                   c
# 4                       103.0                   d
# 5                       104.0                   e
```

Example 9: writes a TableDataset using a template and a configuration file

```
tds = TableDataset()
l1 = Long1d([1,2,3,4,5])
f1 = Float1d([100.0,101.0,102.0,103.0,104.0])
s1 = String1d(['a','b','c','d','e'])
tds.addColumn('Table Column 0',Column(l1))
tds.addColumn('Table Column 1',Column(f1))
tds.addColumn('Table Column 2',Column(s1))
tt=TableTemplate(3)
tt.names = ['Table Column 0','Table Column 1','Table Column 2']
tt.types = ['Long', 'Float', 'String']
tt.units = [herschel.share.unit.Power.WATTS.name,
 herschel.share.unit.Frequency.MEGAHERTZ.name,
   herschel.share.unit.Unit.parse('SpecialUnit').name]
tt.descriptions = ['column_1 Description','column_2 Description','column_3
 Description']
asciiTableWriter(file="file.ascii",table=tds,template=tt,writeHeader=False,writeConfigFile="con
#it generates the following output:
# 1,100.0,a
# 2,101.0,b
# 3,102.0,c
# 4,103.0,d
# 5,104.0,e
#it generates 'config.bin' file also. It contains the current formatter
 settings.
#From now on, it can be applied to other TableDatasets:
tds = TableDataset()
l1 = Long1d([10,20,30,40,50])
f1 = Float1d([200.0,201.0,202.0,203.0,204.0])
s1 = String1d(['a0','b0','c0','d0','e0'])
tds.addColumn('Table Column 0',Column(l1))
tds.addColumn('Table Column 1',Column(f1))
tds.addColumn('Table Column 2',Column(s1))
asciiTableWriter(file='file2.ascii',table=tds,readConfigFile='config.bin')
#it generates the following output:
#Table Column 0,Table Column 1,Table Column 2
#Long,Float,String
#W,MHz [1000000.0 Hz],SpecialUnit
#column_1 Description,column_2 Description,column_3 Description
# 10,200.0,a0
# 20,201.0,b0
# 30,202.0,c0
# 40,203.0,d0
# 50,204.0,e0
```

Example 10: writes a TableDataset with a template selecting and reordering columns

```
tds = TableDataset()
l1 = Long1d([1,2,3,4,5])
f1 = Float1d([100.0,101.0,102.0,103.0,104.0])
s1 = String1d(['a','b','c','d','e'])
tds.addColumn("Table Column
 0",Column(l1,herschel.share.unit.Power.WATTS,"column_1 Description"))
tds.addColumn("Table Column
 1",Column(f1,herschel.share.unit.Frequency.MEGAHERTZ,"column_2 Description"))
tds.addColumn("Table Column
 2",Column(s1,herschel.share.unit.Unit.parse('SpecialUnit'),"column_3
 Description"))
```

Example 10: writes a TableDataset with a template selecting and reordering columns

```
# template to write only columns 2 and 0, in that order
tt=TableTemplate(2)
tt.names = ['Table Column 2','Table Column 0']
tt.types = ['String', 'Long']
tt.units = [str(herschel.share.unit.Unit.parse('SpecialUnit')),
 str(herschel.share.unit.Power.WATTS)]
tt.descriptions = ['tt column_2 Description','tt column_0 Description']
asciiTableWriter(file='file2.ascii',table=tds,template=tt)
#it generates the following output:
# Table Column 2,Table Column 0
# String,Long
# SpecialUnit,W
# tt column_2 Description,tt column_0 Description
# a,1
# b,2
# c,3
# d,4
# e,5
#
```

API Summary

Jython Syntax

Complete signature

asciiTableWriter(<table>, <file> [, <table-Type>="CSV", <readConfigFile>, <writeConfigFile>, <formatter>,

<writeHeader>=True, <writeMetadata>=True, <metadat-aPrefix>="#", <template>, <warn>=False])

With a configuration File (readConfigFile)

asciiTableWriter(<table>, <file> , <readConfig-File> [, <writeConfigFile>, <warn>=False])

With a formatter

asciiTableWriter(<table>, <file> [<writeConfig-File>] , <formatter>

[, <template>, <warn>=False])

Without neither formatter nor configuration

asciiTableWriter(<table>, <file> [, <table-Type>="CSV", <writeConfigFile>,

<writeHeader>=True, <writeMetadata>=False, <meta-dataPrefix>="#", <template>, <warn>=False])

Properties

Annotatable **table** [INPUT, MANDATORY, default=null]
String **file** [INPUT, MANDATORY, default=null]
String **tableType** [INPUT, OPTIONAL, default="CSV"]
String **readConfigFile** [INPUT, OPTIONAL, default=null]
String **writeConfigFile** [INPUT, OPTIONAL, default=null]
AsciiFormatter **formatter** [INPUT, OPTIONAL, default=default AsciiTableTool formatter.]
Boolean **writeHeader** [INPUT, OPTIONAL, default=True]

Properties
Boolean **writeMetadata** [INPUT, OPTIONAL, default=True]
String **metadataPrefix** [INPUT, OPTIONAL, default="#"]
TableTemplate **template** [INPUT, OPTIONAL, default=None: extracted from the table]
Boolean **warn** [INPUT, OPTIONAL, default=False]

Limitations

- Only TableDatasets without empty or multidimensional columns can be written to text files.
- Empty tables will not be saved
- (In both of the previous cases, you can save them as FITS files with simpleFitsWriter)

API details

Properties

Annotatable table [INPUT, MANDATORY, default=null]

TableDataset (or Product with just one TableDataset) object to be saved.

String file [INPUT, MANDATORY, default=null]

Output file.

String tableType [INPUT, OPTIONAL, default="CSV"]

The format of the written table. Valid values are "CSV", "FIXED". If formatter is used this parameter will be ignored.

String readConfigFile [INPUT, OPTIONAL, default=null]

Configuration file where the formatter (AsciiFormatter), parser (AsciiParser) and table template (TableTemplate) must be specified. When readConfigFile parameter is specified, any parameter related to parser or table template are not allowed. You can generate the file using writeConfigFile (for usage in subsequent write operations).

String writeConfigFile [INPUT, OPTIONAL, default=null]

If a file is specified, an output configuration file will be created (to be used as readConfigFile in subsequent write operations).

AsciiFormatter formatter [INPUT, OPTIONAL, default=default AsciiTableTool formatter.]

AsciiFormatter object (specifies how to write the cells and the table header).

Boolean writeHeader [INPUT, OPTIONAL, default=True]

Specifies whether an HCSS table header is written AsciiFormatter.setHeader(true/false) is called.

Boolean writeMetadata [INPUT, OPTIONAL, default=True]

Specifies whether comment lines with metadata are written before the table. AsciiFormatter.setCommented(true/false) is called.

String metadataPrefix [INPUT, OPTIONAL, default="#"]
Specifies the prefix of all comments. AsciiFormatter.setCommentPrefix(expression) is called.

TableTemplate template [INPUT, OPTIONAL, default=None: extracted from the table]
Specifies the header for the file.

Boolean warn [INPUT, OPTIONAL, default=False]
If true, asks confirmation before overwriting the table file.

See also

- asciiTableReader
- simpleFitsWriter
- sortTable
- Chapter 2 in *Data Analysis Guide*
- Developers Manual: herschel.ia.toolbox.util.AsciiTableWriterTask

History

- `2007-11-22 - JCS`: initial version
- `2009-12-09 - JDS`: warn parameter
- `2012-02-20 - JDS`: Accept also Products with just one TableDataset
- `2013-04-25 - JDS`: default warn to false
- `2013-05-22 - JDS`: parameter rename, changed priorities, more documentation

1.29. astrometryFix

Full Name:	herschel.ia.toolbox.astro.AstrometryFixTask
Alias:	astrometryFix
Type:	Java Task -
Import:	from herschel.ia.toolbox.astro import AstrometryFixTask
Category	Astronomical utilities

Description

This task changes the astrometry in the input data to make it consistent with some other data.

The task first estimates the astrometric offset (either by cross-correlation of images, or by stacking on source positions), and then applies that offset to the input data. These steps are described below.

Applying the correction to the input data

- For image data in the "data" parameter, the correction is applied to the WCS only; the image data remains unchanged.
- The offset is translated into a new WCS as follows. First, the reference pixel of the input image is found. Then the offset is applied. The (ra, dec) of this (reference pixel plus the offset) is then taken to be the (ra, dec) of the reference pixel itself (crval1 and crval2 in the WCS). What this means is that it is assumed that the orientation of the map at the reference pixel is correct.
- If the "data" is an ObservationContext, then an attempt will be made to apply the astrometric correction to the Level-1 data. (This is currently supported for SPIRE observations only.) If the task is unable to apply such a correction, a warning message will be displayed in the log. The correction is applied by (1) taking the (ra, dec) values, and translating them to pixel coordinates using the original WCS of the first image in the Level-2 context, and then (2) converting those pixel coordinates into new (ra, dec) values using the *new* (corrected) WCS of that same first image.
- If the "data" and/or "reference" parameter has more than one image (for example, contained in the ObservationContext), then more than one determination of the offset will be performed, as described below, and a weighted mean of the different offsets is performed to determine the one offset to apply to all of the input images.

Estimating the astrometric offset using image data

- If the "reference" parameter contains image data, the correction is determined by cross-correlation of images between the "data" parameter and the "reference" parameter.
- The correction is calculated near the reference pixel of the input ("data") image(s), so it is important that the reference image(s) overlap with the reference pixel of the input image(s).
- First, the reference image is regridded onto the same WCS as the input ("data") image (using the regrid task). Then a cross-correlation is performed, with the images offset by an integer number of pixels in the x and y direction. This produces a 2x2 grid with the cross-correlation of the images at various offsets. A Gaussian fit is performed on this 2x2 grid to find the peak position, and this is taken to be the astrometric offset. If the input image is large, only the central region is used. If the reference image is larger than required, only the central region will be used. In these cases, a log message will be given.
- If the "data" and/or "reference" parameter has more than one image (for example, contained in the ObservationContext), then more than one determination of the offset will be performed, between different pairs of images. A weighted mean of the different offsets is then performed to determine

the one offset to apply to all of the input images. If both input data and reference have more than one image, then the number of images in the "data" and "reference" parameters must be the same, and a comparison will be performed between the first "data" image and the first "reference" image, then between the second "data" image and the second "reference" image, and so on. If one of the parameters has one image and the other has more than one image, then a comparison will be performed between that one image and each of the images in the other parameter in turn.

Estimating the astrometric offset using source positions

- If the "reference" parameter contains source positions, the correction is determined by stacking at those source positions.
- For each source position, if it lies well within the image, the pixels around that position are taken and a bilinear interpolation is performed on the exact source position. (If NaN pixels are found near to the source position, that source position is skipped.) These smaller images are then added together, and a Gaussian fit is then performed to find the peak position. The offset between this peak position and the centre is taken to be the astrometric offset.
- If the input parameter has more than one image (for example, contained in the ObservationContext), then stacking is performed for each of the input images in turn. A weighted mean of the different offsets is then performed to determine the one offset to apply to all of the input images.

API Summary

Jython Syntax
`data = astrometryFix(&lt;data&gt;, &lt;reference&gt; [, &lt;offsetMaxArcsec&gt;=30.0, &lt;tempStorage&gt;=False] )`

Properties
`Object` **`data`** `[INOUT, MANDATORY, default=no default value]`
`Object` **`reference`** `[INOUT, MANDATORY, default=no default value]`
`Double` **`offsetMaxArcsec`** `[INPUT, OPTIONAL, default=30.0]`
`Boolean` **`tempStorage`** `[INPUT, OPTIONAL, default=false]`

API details

Properties

`Object data [INOUT, MANDATORY, default=no default value]`

Input SimpleImage, MapContext, PacsContext or ObservationContext to have the astrometry corrected. The output will be of the same type. For SimpleImage, MapContext and PacsContext, the output will be a new product, and the input product will be unaltered. For ObservationContext, the input will itself be altered and returned, with a new Level-2 context, and (if possible: SPIRE only) with the ra and dec values in the original Level-1 context changed.

`Object reference [INOUT, MANDATORY, default=no default value]`

Reference SimpleImage, MapContext, PacsContext, ObservationContext, SourceListProduct, SourceListDataset or PositionList for the astrometry correction

`Double offsetMaxArcsec [INPUT, OPTIONAL, default=30.0]`

Maximum allowed offset for the astrometry correction. The task will fail if the offset is measured to be greater than this value

Boolean tempStorage [INPUT, OPTIONAL, default=false]
Flag to use temporary storage ProductSink when changing the astrometry in Level-1 data

See also

- Developers Manual: herschel.ia.toolbox.astro.AstrometryFixTask

History

- 2011-09-27 - AJS: first release version

1.30. AttribQuery

Full Name:	herschel.ia.pal.query.AttribQuery
Type:	Java Class -
Import:	from herschel.ia.pal.query import AttribQuery
Category	Data access

Description

Attribute Query formulates a query on the attributes of a Product.

In principle this can be the fastest query on the Product Access Layer. Known attributes are:

- type : String
- creator : String
- creationDate: FineTime
- startDate : FineTime
- endDate : FineTime
- modelName : String
- instrument : String
- description : String

Example

Example 1: Example of a query on attributes

```
date = SimpleTimeFormat().parse("2008-10-31T12:00:00
 TAI").microsecondsSince1958()
q = AttribQuery(Product, "p", "p.creationDate &lt; " + str(date) + "L and
 p.creator = 'Me'")
```

See also

- Developers Manual: herschel.ia.pal.query.AttribQuery

1.31. autoCorrelation

Full Name:	herschel.ia.toolbox.image.AutoCorrelationTask
Alias:	autoCorrelation
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import AutoCorrelationTask
Category	Images/Analysis

Description

This is a task to calculate the linear Pearson correlation coefficients of an image.

This is a task to calculate the linear Pearson correlation coefficients of an image. We start from a M x N image (M = number of rows, N = number of columns) and calculate the N x N linear Pearson coeffient matrix . The i'th row and j'th column element corresponds to the correlation of the i'th and j'th columns of the M x N matrix.

Example

Example 1: This is an example of how to use the autoCorrelation :

```
correlation = autoCorrelation(image = myImage)
```

API Summary

Properties
`Image image [INPUT, MANDATORY, default=None]`
`Double2d correlation [OUTPUT, MANDATORY, default=None]`

API details

Properties

`Image image [INPUT, MANDATORY, default=None]`

This is the input image.

`Double2d correlation [OUTPUT, MANDATORY, default=None]`

This is the matrix with the linear Pearson corelation coefficients.

See also

- Developers Manual: herschel.ia.toolbox.image.AutoCorrelationTask

1.32. automaticContour

Full Name:	herschel.ia.toolbox.image.AutomaticContourTask
Alias:	automaticContour
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import AutomaticContourTask
Category	Images/Analysis

Description

This is a task that generates the contours for a given number of contour levels, distribution and extreme values.

Available distributions are linear, log or ln. The output product is an ImageContour product, which holds the generated contours. This product can be dragged over an image explorer for visual inspection. The user can also specify in this task which colors should be used for visualization.

Example

Example 1: This is an example of how to use automaticContour :

```
from java.awt.Color import GREEN
from java.awt.Color import RED
contours = automaticContour(image = image, levels = 2, min = 3.7, max = 4.2,
 distribution = 1,colors = [GREEN, RED])
contours = automaticContour(image = image, levels = 2, min = 3.7, max = 4.2,
 distribution = 1)
```

API Summary

Properties
Image **image** [INPUT, MANDATORY, default=None]
Integer **levels** [INPUT, MANDATORY, default=None]
Double **min** [INPUT, MANDATORY, default=None]
Double **max** [INPUT, MANDATORY, default=None]
Integer **distribution** [INPUT, OPTIONAL, default=1 (log)]
Color[] **colors** [INPUT, OPTIONAL, default=None (BLUE is used then)]
ImageContour **contours** [OUTPUT, MANDATORY, default=None]

API details

Properties

Image image [INPUT, MANDATORY, default=None]

This is the input image.

Integer levels [INPUT, MANDATORY, default=None]

This is the number of contour levels.

Double min [INPUT, MANDATORY, default=None]

This is the minimum contour value.

Double max [INPUT, MANDATORY, default=None]

This is the maximum contour value.

Integer distribution [INPUT, OPTIONAL, default=1 (log)]

This is the distribution of the contour values. Possible values are 0 : lin, 1 : log and 2 : ln.

Color[] colors [INPUT, OPTIONAL, default=None (BLUE is used then)]

These are colors to visualize the contours.

ImageContour contours [OUTPUT, MANDATORY, default=None]

These are the resulting contours.

See also

- Developers Manual: herschel.ia.toolbox.image.AutomaticContourTask

1.33. avg

Full Name:	herschel.ia.toolbox.spectrum.AverageSpectrumTask
Alias:	avg
Type:	Java Task -
Import:	from herschel.ia.toolbox.spectrum import AverageSpectrumTask
Category	Spectra/Analysis

Description

Task for averaging spectra.

The average operation is applied to the 'flux' values contained in the so called spectral segments. These values are averaged on a per frequency or wavelength bin (or whatever the wavescale unit is) basis without checking whether the frequencies (or wavelengths) assigned to these bins are well aligned across the different spectra. If this is not the case, the spectra should first be resampled to a common wavescale grid. If available, the weights (sometimes interpreted as inverse square error) are summed up and the per pixel flag values propagated following a bit-wise OR logic.

The input data with the spectra to be averaged can be provided in different forms: Either any data object that implements `SpectrumContainer` (e.g. `Spectrum1d, Spectrum2d, SpectralSimpleCube`) or any array of such. `SpectrumContainer`'s provides a uniform way to access spectrum data organized in quite different structures.

For a successful processing of the task, the data included in the containers should all be consistent. This means that the number of segments found in all the spectra in the containers and their lengths should be equal; furthermore, the segment indices should be the same for all the containers (check with `ds.getSegmentIndices()`). Otherwise an exception is thrown.

Different schemes are available for selecting the point spectra or the segments to be averaged. The most simple scheme is to specify lists of point spectrum indices (`selection=[1,3,4,2]`). In this case, the average is just taken over the point spectra with corresponding indices. Instead of specifying a list of indices (which requires the knowledge of these) other recipes to specify a selection are available. Typical recipes test attribute values to match certain values (e.g. `selection={"bbtype":[6031, 6032]}`) or to be included in a given interval (e.g. `selection={"LoFrequency":(999.98,1000.02)}`). These recipes we refer to as selection models. The different options to specify selections can be combined. Here, an AND logic is adopted.

See the parameter descriptions and the examples below for further detail. Alternatively, look in the `SelectSpectrum`-task.

There are further options that can be set by the `variant`-parameter. Available are:

- "flux": Take simple average of flux arrays, add weight arrays, propagate flags adopting a bit-wise OR and take the wave array of `ds2`;
- "flux-weight": Take weighted average of flux arrays, add weight arrays, propagate flags adopting a bit-wise OR and take the wave array of `ds2`;
- "flux-flag": Take simple average of flux arrays but ignore flagged bins, add weight arrays, propagate flags adopting a bit-wise OR and take the wave array of `ds2`;
- "flux-flag-weight": Take weighted average of flux arrays but ignore flagged bins, add weight arrays, propagate flags adopting a bit-wise OR and take the wave array of `ds2`;

For each of the above option an alternative exists with '-wave' appended. In those cases, the wave arrays of `ds1` and `ds2` are computed by computing an average consistent with the type of average

applied to the flux values (simple, using weights or flags or both). Note that not all implementations of `SpectrumContainer` allow to have individual wavescale defined for each and every point spectrum. Furthermore, note that for PACS and SPIRE, we rather have error and mask instead of weights and flags, respectively. Typically, the PACS and SPIRE-specific implementations of the spectrum data containers map the error to weights (e.g. by defining the weights to be inversely squared error). Furthermore, the mask is just mapped 'as is' to flags. As a result if a PACS user wants to compute an average weighted with inversely-error the option `variant='flux-weight'` can be used.

Spectra that just contain NaN's as flux values are not automatically ignored in the average. This implies that if such a spectrum is present the average will turn out to be NaN - once e.g. the bins in this spectrum are not flagged or a simple average is taken without considering flags. This behavior can, however, be configured by the parameter `filterNaNSpectra`. By default, this parameter is set to True.

Similarly, you can specify what segments to consider in the averaging. In the most simple case, you can just specify the indices of segments to be considered `segments=[1,3,4]`. In more advanced situations you can configure a `SegmentSelection`-object and pass that as `segments`-parameter to the task.

Sometimes, the selection model also provides a natural segmentation of the data into groups. If you want to calculate the average on a per group basis then you can set the per_group flag to true.

Using the keyword grouping the user may specify a grouping model which allows to partition the spectra included in the container into suitable sub-groups. When a grouping model is specified, the average is calculated for each sub-group, separately. Grouping and selections can be combined such that the selection model(s) impose(s) constraints on the spectra to be included in the groups.

Other attributes that characterize the individual spectra in the container may also be processed. (Examples for such attributes typically are integration time, observation time, position in sky and many other instrument-specific parameters.) The default is that a result value is set if the values of all the input spectra are the same. For the different instruments and datatypes, different 'attribute operations' are and can be configured. Consult the developers manual for further details. An easy way to inspect what attribute operations have been configured for given input spectra is by executing

`avgOp = herschel.ia.toolbox.spectrum.operations.Average() print avgOp.getSpecification(spectrum)`

A further option to configure the avg-task in a given HIPE session is to register attribute operations from the command line:

`avg.register("integrations", "ADD")` Note that this will only survive as long as you use the same instance 'avg'. Hence you can reset by creating a new instance: `avg = AverageSpectrum()`. Furthermore, note that the operations registered in this way are not returned by `print avgOp.getSpecification(spectrum)`.

Examples

Example 1: Just a straight average

```
global spectra # defined elsewhere
avg = AverageSpectrumTask()
averagedSpectra = avg(ds=spectra)
```

Example 2: Restrict to segments with indices '1' and '3':

```
global spectra # defined elsewhere
avg = AverageSpectrumTask()
spectraOut = avg(ds=spectra, segments=[1,3])
```

Example 3: Using selections and segments

```
global spectra # defined elsewhere
```

Example 3: Using selections and segments

```
avg = AverageSpectrumTask()
# just select by index:
spectraOut = avg(ds=spectra, selection=[0,1,2,3])
# select by looking up suitable attributes attached to the spectra to be
 selected (here: "bbtype" matched):
spectraOut = avg(ds=spectra, selection={"bbtype":[6031, 6613]})
# select by looking up suitable attributes attached to the spectra to be
 selected
# (here: "LoFrequency" in given interval):
spectraOut = avg(ds=spectra, selection={"LoFrequency":(550.0,570.0)})
# select by looking up suitable attributes attached to the spectra to be
 selected
# (here: "Chopper" in given ranges):
spectraOut = avg(ds=spectra, selection={"Chopper":(4.,7.)})
# same as above - in addition, restrict to the segments with segment indices
 '1', '2':
sspectraOut = avg(ds=spectra, selection={"Chopper":(4.,7.)}, segments=[1,2])
# for cubes you can specify the selection by spaxel coordinates - just pass a
 list of tuples (col,row)
# where 'col' is the column and 'row' the row index:
spectraOut = avg(ds=spectra, selection=[(1,3),(0,4),(4,3)])
# or all combined - as mentioned in the description above adopting a AND logic:
spectraOut = avg(ds=spectra,
                 selection={"bbtype":[6031, 6613],"Chopper":
([-4.4,5.9],0.1),"LoFrequency":(550.0,570.0)})
```

Example 4: Using the 'variant' parameter

```
global spectra # defined elsewhere
avg = AverageSpectrumTask()
# Compute a straight average of the flux values, OR all the flags and
# sum up all the weights included in the input spectra:
spectraOut = avg(ds=spectra, variant="flux")
# Compute a weighted average of the flux values, OR all the flags and
# sum up all the weights included in the input spectra:
spectraOut = avg(ds=spectra, variant="flux-weight")
# Compute a straight average of all the non-flagged flux values
# (or, for a given pixel, do a straight average
# in case there are no non-flagged values and AND all the flag values) and
# sum up the weights of the contributing pixels:
spectraOut = avg(ds=spectra, variant="flux-flag")
# The same as above, but eliminating flux values that have just the flags 1,4
 or 16:
spectraOut = avg(ds=spectra, variant="flux-flag", flagToIgnore=21)
# The same as above, but computing a weighted average:
spectraOut = avg(ds=spectra, variant="flux-flag-weight", flagToIgnore=21)
```

Example 5: Computing weighted average with external weight

```
global spectra # defined elsewhere
spectraOut = avg(ds=spectra, variant="flux-weight", weights="integrations")
weights = Double1d(spectra.pointSpectrumCount,1)
weights[0]=2
spectraOut = avg(ds=spectra, variant="flux-weight", weights=weights)
```

Example 6: Computing per group averages

```
global spectra # defined elsewhere
# the result of the following will contain two rows: one with the average of
 all bbtype=6031 spectra and
# the other with the bbtype=6032 spectra.
spectraOut = avg(ds=spectra, selection={"bbtype":[6031, 6032]},
 per_group=True)
# the following attempts to build disjoint groups of spectra - each group
 characterized by Chopper values
# that are within a tolerance of 0.1 the same.
```

Example 6: Computing per group averages

```
from herschel.ia.toolbox.spectrum.selections.models import RangesGroupingModel
averagedSpectraPerGroup = avg(ds=spectra,
 grouping=RangesGroupingModel("Chopper", 0.1))
```

Example 7: Computing averages against a registered attribute

```
global spectra # defined elsewhere
# register attribute operation so that the attribute 'integrations' is summed
 up over all input spectra:
avg = AverageSpectrumTask()
avg.register("integrations","ADD")
averagedSpectra = avg(ds=spectra)
```

API Summary

Properties
Object **ds** [INPUT, OPTIONAL, default=no default value.]
Object **selection** [INPUT, OPTIONAL, default=None.]
PyDictionary|Map<String,Set<Object>> **lookup selection** [INPUT, OPTIONAL, default=no default value.]
PyList **index selection** [INPUT, OPTIONAL, default=No default value.]
PyDictionary **filter meta** [INPUT, OPTIONAL, default=No default value.]
Object **segments** [INPUT, OPTIONAL, default=no default value.]
String **variant** [INPUT, OPTIONAL, default="flux-weight-flag".]
Integer **flagToIgnore** [INPUT, OPTIONAL, default=no default value.]
Boolean **per_group** [INPUT, OPTIONAL, default=no default value.]
GroupingModel **grouping** [INPUT, OPTIONAL, default=no default value.]
Boolean **filterNaNSpectra** [INPUT, OPTIONAL, default=True.]
Object **weights** [INPUT, OPTIONAL, default=no default value]
SpectrumContainer **result** [OUTPUT, OPTIONAL, default=no default value]

API details

Properties

Object ds [INPUT, OPTIONAL, default=no default value.]

Input data to be processed by the task. Several types are possible:

- SpectrumContainer
- Array of SpectrumContainer, i.e. SpectrumContainer[] (e.g. `[ds1,ds2,ds3]`)
- List of SpectrumContainer, i.e. List<SpectrumContainer>
- (Any product with implementations of `SpectrumContainer` inside.)

Object ds [INPUT, OPTIONAL, default=no default value.]

Examples of `SpectrumContainer` are `Spectrum1d, Spectrum2d, SpectralSimpleCube`.

Object selection [INPUT, OPTIONAL, default=None.]

Specification of what point spectra the average should be restricted to. Different ways to specify these selections are possible:

- Specify a list of indices (in jython) of the point spectra for which the average should be taken. For cubes, you can alternatively also pass a list of spaxel coordinates (a list of integer tuples `(row,col)`).
- Use a dictionary (in jython) to specify a selection model (lookup for attributes, filter attributes to be included in ranges or intervals).
- Pass a string representation of a jython list or a jython dictionary. The string will be parsed and the processing delegated to the two cases above.
- Pass any java instance that implements the SelectionModel interface (for the advanced user).

See the examples below or the `SelectSpectrum`-task for how to specify selections.

PyDictionary|Map<String,Set<Object>> lookup_selection [INPUT, OPTIONAL, default=no default value.]

Specify a PyDictionary with keys given by the attribute name to look up and a list of admissible values. Behind the scenes, a DiscreteValueSelectionModel is instantiated. This parameter is actually obsolete but is kept for historical reasons (use `selection`).

PyList index_selection [INPUT, OPTIONAL, default=No default value.]

Specify a PyList with the indices of the point spectra to be considered. This parameter is actually obsolete but is kept for historical reasons (use `selection`).

PyDictionary filter_meta [INPUT, OPTIONAL, default=No default value.]

Restrict the operation on spectrum containers with meta data that match given values. This only applies in case more than one container is passed as input data to the task.

Object segments [INPUT, OPTIONAL, default=no default value.]

Specify what segments the operation should be applied to. There are two options available:

- Either pass an instance of a SegmentSelection which gives the information on what segments for each point spectrum included in the container.
- Specify a PyList of segment indices.

String variant [INPUT, OPTIONAL, default="flux-weight-flag".]

Specify the variant of processing mode you would like to run (including flags / weights / ...). HCSS defaults include: "flux" / "flux-wave" / "flux-wave-weight" / "flux-weight-flag" / "flux-flag-wave" / "flux-weight-flag-wave"

Integer flagToIgnore [INPUT, OPTIONAL, default=no default value.]

Applies only if a 'variant' with the substring 'flag' has been specified. By setting this parameter you can configure which flags should be ignored and which should be propagated as infor-

Integer flagToIgnore [INPUT, OPTIONAL, default=no default value.]

mative flags. For example, if you want to ignore channels that have the flag 1,4 or 16 set you specify a `flagToIgnore=21`. By default, the flagToIgnore is set to 0 which means that any flagged channel is ignored.

Boolean per_group [INPUT, OPTIONAL, default=no default value.]

Specify that the average should be calculated on a per group basis - once a selection model is specified that provides a natural grouping of the data.

GroupingModel grouping [INPUT, OPTIONAL, default=no default value.]

Grouping model that can be used to partition the point spectra into groups and calculate the average on a per group basis. When a grouping model is specified, the 'per_group'-flag is obsolete.

Boolean filterNaNSpectra [INPUT, OPTIONAL, default=True.]

If set to true, only point spectra that contain at least one single non NaN value are included in the average.

Object weights [INPUT, OPTIONAL, default=no default value]

Weights to be used in the weighted average when the weighted average option is set with the `variant` parameter (`variant` should contain 'weight'). Either pass a PyList of double, a Double1d or the name of an attribute (such as `"integration time"` in case of a `Spectrum2d`). If a PyList or Double1d are passed their length must be identical to the number of `PointSpectrum` found in the data. These weights do not need to be normalized but should be non-negative. As attribute names you can pass any attribute name with elements of type `float`, `double`, `short`, `integer`, `long` or `boolean`. The weighted average computation works only for single `SpectrumContainer` data passed to the task. The operations applied to the attributes are not affected by these weights. If per pixel weights (or error) are specified these weights a weighted average of these per pixel weights will be included in the result. Flags are propagated unless 'flag' is also specified in the variant. In this case, flagged pixels are ignored and the weighted average with the non-flagged is computed on a per pixel basis.

SpectrumContainer result [OUTPUT, OPTIONAL, default=no default value]

Result object containing the results of the operation applied.

See also

- SpectrumContainer
- Developers Manual: herschel.ia.toolbox.spectrum.AverageSpectrumTask

History

- `2011-08-08 - melchior`: renamed from AverageSpectrum
- `2011-09-28 - melchior`: add selection by spaxels - tuples (row,col)
- `2012-01-24 - melchior`: more flexible weighted average

1.34. axisAngle2RotationMatrix

Full Name:	herschel.ia.toolbox.pointing.axisAngle2RotationMatrix
Type:	Jython Task -
Import:	from herschel.ia.toolbox.pointing import axisAngle2RotationMatrix
Category	Toolboxes/Pointing

Description

Calculate rotation matrix about an arbitrary axis.

Example

Example 1: axis=Double1d([0.,1.,0.])

```
rom = axisAngle2RotationMatrix(axis,1.3)
```

API Summary

Jython Syntax

```
rotationMatrix = axisAngle2RotationMatrix(axis, angle)
```

Properties

Double1d **axis** [INPUT, MANDATORY, default=NO default value]

double **angle** [INPUT, MANDATORY, default=NO default value]

Double2d **rotationMatrix** [OUTPUT, MANDATORY, default=NO default value]

API details

Properties

Double1d axis [INPUT, MANDATORY, default=NO default value]

3-element vector defining the rotation axis

double angle [INPUT, MANDATORY, default=NO default value]

rotation angle over the axis (radians)

Double2d rotationMatrix [OUTPUT, MANDATORY, default=NO default value]

rotation matrix

History

- 2013-04-22 - BV: Initial version
- 2013-09-26 - BV: HCSS-18567 Review disabled jexamples

1.35. bg

Full Name:	herschel.ia.toolbox.util.BackgroundTask
Alias:	bg
Type:	Java Task -
Import:	from herschel.ia.toolbox.util import BackgroundTask
Category	Session utilities

Description

Execute Jython commands in the background.

The bg task is used to execute Jython commands in a background thread. The future is returned to allow operations with the result: check its state, wait for completion, cancel it, consult failure causes if failed. You can execute long running independent work items in background, freeing the interpreter to execute interactive work at the same time.

Examples

Example 1: Execute function in the background

```
def myfunc():
   print "myfunc called"
   pass
future = bg("myfunc()")
```

Example 2: Print in another thread

```
bg("print 'hi'")
```

API Summary

Jython Syntax

```
future = bg(&lt;command&gt;)
```

Properties

```
String command [INPUT, MANDATORY, default=no default value]
Future future [OUTPUT, MANDATORY, default=no default value]
```

Limitations

- Only applicable for a HIPE session.
- Careless multithreaded programming can lead to race conditions and even deadlocks.
- This task is obviously asynchronous, it may end after (or even before!) the work carried over by the interpreter's thread.

API details

Properties

String command [INPUT, MANDATORY, default=no default value]

A string representing the command to be executed by the Jython interpreter

Future future [OUTPUT, MANDATORY, default=no default value]
The future that executes the command

See also

- http://download.oracle.com/javase/7/docs/api/java/util/concurrent/Future.html
- Developers Manual: herschel.ia.toolbox.util.BackgroundTask

History

- `2004-07-13 - NdC`: first release.
- `2010-03-17 - JDS`: Appears as a running job (spinning wheel).
- `2012-04-09 - JSS`: Usage of pool thread; the output is a Future instead of a Thread.

1.36. Bilinear

Full Name:	herschel.ia.numeric.toolbox.interp.Bilinear
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.interp import Bilinear
Category	Mathematics/Interpolation

Description

The Bilinear class is an interpolation algorithm used to compute the value between regular grid points in an 2D array.

When a point falls outside the array, the nearest neighbor or a user supplied value for 'missing data' will be returned.

Examples are given below.

array: Input, a double2d data array

r: Input, an array or list of row indices

c: Input, an array or list of column indices

grid:

The grid keyword is used to determine the usage of the r,c inputs, and the format of the output. If grid is not set, the location arrays, r, c must have the same number of elements (N). The result is a Double1d array of size N corresponding to the interpolated value. In short, r,c are a list of coordinates to interpolate.

If grid is set, let M be the number of elements in r, let N be the number of elements in c. The result is a Double2d array with dimensions [M,N]. The element [i,j] of the result contains the interpolated value at position r[i], c[j].

missing:

The value to return for elements outside the bounds of images. If this keyword is not specified, interpolated positions that fall outside the bounds of the array images - that is, elements of the x or y arguments that are either less than zero or greater than the largest subscript in the corresponding dimension of images - are set equal to the value of the nearest element of images.

Special notes for using the task on image arrays

All 2D Herschel arrays are stored in ROW-order. This means for an index [i,j], 'i' corresponds to the row, and 'j' to the column. For images, convention is usually such that 'x,y' are used for the column and row respectively. Thus, to use the bilinear class on images, users should take care to reverse the x,y, notation to have it conform with the row-ordered format.

Examples

Example 1: All data points are inside the boundary

```
from herschel.ia.numeric.toolbox.interp import  Bilinear
array=Double2d([[0,10],[20,30],[10,2]])
r=[0, 1.5]
c=[0.5,0.5]
print Bilinear(r,c)(array)
# [5,15.5]
```

Example 2: Single interpolation

```
# Interpolate between 0 and 10 (midpoint in first row)
# Make a 2d array.  2 rows, 3 columns
from herschel.ia.numeric.toolbox.interp import  Bilinear
array=Double2d([[0,10],[20,30],[10,2]])
r=[0]
c=[0.5]
print Bilinear(r,c)(array)
# [5]
```

Example 3: Multiple point interpolation

```
# Interpolate between 0 and 10 (midpoint in first row)
# Interpolate between 20 and 2 (midpoint between second row, first column, and
 third row, second column
# Make a 2d array.  2 rows, 3 columns
from herschel.ia.numeric.toolbox.interp import  Bilinear
array=Double2d([[0,10],[20,30],[10,2]])
r=[0, 1.5]
c=[0.5,0.5]
print Bilinear(r,c)(array)
# [5,15.5]
```

Example 4: Interpolate beyond last point in the first row.

```
# Make a 2d array.  2 rows, 3 columns
from herschel.ia.numeric.toolbox.interp import  Bilinear
array=Double2d([[0,10],[20,30],[10,2]])
r=[0]
c=[2]
print Bilinear(r,c)(array)
# [10.0] # answer is snapped to the nearest neighbor
```

Example 5: Forcing a 'missing' value for interpolation outside the array

```
# Make a 2d array.  2 rows, 3 columns
from herschel.ia.numeric.toolbox.interp import  Bilinear
array=Double2d([[0,10],[20,30],[10,2]])
r=[0]
c=[2]
print Bilinear(r,c,-1)(array)
# [-1.0]
```

Example 6: Extract a 2x2 resampled grid out of the 3x2 input array

```
# Make a 2d array.  2 rows, 3 columns
from herschel.ia.numeric.toolbox.interp import  Bilinear
array=Double2d([[0,10],[20,30],[10,2]])
r=[0.5,1.5]
c=[0,2]
print Bilinear(Bilinear.GRID, r,c)(array)
#[10.0,20.0],
#[15.0,16.0]
#]
```

Example 7: resample an array which is treated as an image.

```
from herschel.ia.numeric.toolbox.interp import  Bilinear
image=RESHAPE(Double1d.range(16), [4,4])
x=[-0.5,2]
y=[0.5,1.5]
print Bilinear(Bilinear.GRID, y,x, -1.0)(image)
# [
# [-1.0,4.0],
```

Example 7: resample an array which is treated as an image.

```
# [-1.0,8.0]
# ]
#IDL syntax print,Bilinear(TRANSPOSE(images),xB,yB)
```

API Summary

Constructors
Bilinear (Double1d x, Double1d y)
Bilinear (double[] x, double[] y)
Bilinear (Boolean grid, Double1d x, Double1d y)
Bilinear (Boolean grid, double[] x, double[] y)
Bilinear (Double1d x, Double1d y, double missing)
Bilinear (double[] x, double[] y, double missing)
Bilinear (Boolean grid, Double1d x, Double1d y, double missing)
Bilinear (Boolean grid, double[] x, double[] y, double missing)

Method
ArrayData **of** (Double2d images)

API Details

Constructors

Bilinear (Double1d x, Double1d y)

Arguments

Double1d **x** [INPUT, MANDATORY, default=no default value]

The array containing the x "virtual subscripts" of the images for which to interpolate values.

Double1d **y** [INPUT, MANDATORY, default=no default value]

The array containing the y "virtual subscripts" of the images for which to interpolate values

Error

Array

size error The length of array x and y has to be equal. The exception will be thrown if the lengths of the array x and y are not equal.

Bilinear (double[] x, double[] y)

Arguments

double[] **x** [INPUT, MANDATORY, default=no default value]

The array containing the x "virtual subscripts" of the images for which to interpolate values.

double[] **y** [INPUT, MANDATORY, default=no default value]

The array containing the y "virtual subscripts" of the images for which to interpolate values

Error

Array

size error The length of array x and y has to be equal. The exception will be thrown if the lengths of the array x and y are not equal.

Bilinear (Boolean grid, Double1d x, Double1d y)

Arguments

Boolean **grid** [INPUT, MANDATORY, default=no default value]

If grid is true and if the x contains m elements and y contains n elements, the result has dimensions [m, n], where the element [i,j] of the result contains the value of Images interpolated at position (xi, yj). If the grid is false, the result is the same as calling Bilinear without the grid argument.

Double1d **x** [INPUT, MANDATORY, default=no default value]

The array containing the x "virtual subscripts" of the images for which to interpolate values.

Double1d **y** [INPUT, MANDATORY, default=no default value]

The array containing the y "virtual subscripts" of the images for which to interpolate values

Bilinear (Boolean grid, double[] x, double[] y)

Arguments

Boolean **grid** [INPUT, MANDATORY, default=no default value]

If grid is true and if the x contains m elements and y contains n elements, the result has dimensions [m, n], where the element [i,j] of the result contains the value of Images interpolated at position (xi, yj). If the grid is false, the result is the same as calling Bilinear without the grid argument.

double[] **x** [INPUT, MANDATORY, default=no default value]

The array containing the x "virtual subscripts" of the images for which to interpolate values.

double[] **y** [INPUT, MANDATORY, default=no default value]

The array containing the y "virtual subscripts" of the images for which to interpolate values

Bilinear (Double1d x, Double1d y, double missing)

Arguments

Double1d **x** [INPUT, MANDATORY, default=no default value]

The array containing the x "virtual subscripts" of the images for which to interpolate values.

Double1d **y** [INPUT, MANDATORY, default=no default value]

The array containing the y "virtual subscripts" of the images for which to interpolate values

double **missing** [INPUT, MANDATORY, default=no default value]

When missing is given, the value outside the boundary is set to missing.

Bilinear (double[] x, double[] y, double missing)

Arguments

double[] **x** [INPUT, MANDATORY, default=no default value]

The array containing the x "virtual subscripts" of the images for which to interpolate values.

double[] **y** [INPUT, MANDATORY, default=no default value]

The array containing the y "virtual subscripts" of the images for which to interpolate values

double **missing** [INPUT, MANDATORY, default=no default value]

When missing is given, the value outside the boundary is set to missing.

Bilinear (Boolean grid, Double1d x, Double1d y, double missing)

Arguments

Boolean **grid** [INPUT, MANDATORY, default=no default value]

Bilinear (Boolean grid, Double1d x, Double1d y, double missing)

If grid is true and if the x contains m elements and y contains n elements, the result has dimensions [m, n], where the element [i,j] of the result contains the value of Images interpolated at position (xi, yj). If the grid is false, the result is the same as calling Bilinear without the grid argument.

`Double1d` **`x`** `[INPUT, MANDATORY, default=no default value]`

The array containing the x "virtual subscripts" of the images for which to interpolate values.

`Double1d` **`y`** `[INPUT, MANDATORY, default=no default value]`

The array containing the y "virtual subscripts" of the images for which to interpolate values

`double` **`missing`** `[INPUT, MANDATORY, default=no default value]`

When missing is given, the value outside the boundary is set to missing.

Bilinear (Boolean grid, double[] x, double[] y, double missing)

Arguments

`Boolean` **`grid`** `[INPUT, MANDATORY, default=no default value]`

If grid is true and if the x contains m elements and y contains n elements, the result has dimensions [m, n], where the element [i,j] of the result contains the value of Images interpolated at position (xi, yj). If the grid is false, the result is the same as calling Bilinear without the grid argument.

`double[]` **`x`** `[INPUT, MANDATORY, default=no default value]`

The array containing the x "virtual subscripts" of the images for which to interpolate values.

`double[]` **`y`** `[INPUT, MANDATORY, default=no default value]`

The array containing the y "virtual subscripts" of the images for which to interpolate values

`double` **`missing`** `[INPUT, MANDATORY, default=no default value]`

When missing is given, the value outside the boundary is set to missing.

Method

***ArrayData* of (Double2d images)**

Argument

`Double2d` **`images`** `[INPUT, MANDATORY, default=no default value]`

A two dimension data array.

Return

ArrayData

the interpolated image values at the (xi, yj).

See also

- CubicSplineInterpolator
- NearestNeighborInterpolator
- http://en.wikipedia.org/wiki/Bilinear
- Developers Manual: herschel.ia.numeric.toolbox.interp.Bilinear

History

- `2007-03-05 - LZ`: first version

- 2007-03-24 - LZ: revised version
- 2008-03-27 - LZ: implemented SPR4608
- 2009-02-24 - LZ: Updated documentation
- 2009-02-25 - LZ: add URM SPR-6124

1.37. BinCentres

Full Name:	herschel.ia.numeric.toolbox.basic.BinCentres
Alias:	BinCentres
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import BinCentres
Category	Mathematics/Histograms

Description

Returns an array of histogram bin centres for your data, based on bin size.

There are two steps involved in using this tool. First you create an object of type `BinCentres` by specifying the bin size of the histogram you want to create. Then you pass your data array to this object, getting the array of bin centres as output. You can use this array of bin centres to create a plot of your histogram, together with the **Histogram** function (see the link in the *See also* section). Look at the examples and the *Jython Syntax* section below to learn more.

Examples

Example 1: Plot a histogram with BinCentres for a Double1d

```
from herschel.ia.numeric.toolbox.basic import Histogram
from herschel.ia.numeric.toolbox.basic import BinCentres
d = Double1d(200000,10.0)
d[90000:115000] = 20.0
d[99000:110000] = 22.0
d[99900:100100] = 23.0
d[99990:100010] = 24.0
d[100000] = 24.5
rnd = RandomGauss()
for ix in range(200000):
  d[ix] += rnd.calc(0.1)
p = PlotXY (d)
binsize = STDDEV (d) * 2.35
print binsize
hist=Histogram(binsize)
bins=BinCentres(binsize)
p2=PlotXY(bins(d),hist(d))
style=p2.getStyle()
style.setChartType(Style.HISTOGRAM)
p.close ()
p2.close ()
```

Example 2: Plot a histogram with BinCentres for a Byte1d

```
from herschel.ia.numeric.toolbox.basic import Histogram
from herschel.ia.numeric.toolbox.basic import BinCentres
vals = [0, 27, 25, 60, 62, 70, 71, 73, 74, 100, 120, 92, 85, 116]
d = Byte1d (vals)
p = PlotXY (d)
binsize = 5
print binsize
hist=Histogram(binsize)
bins=BinCentres(binsize)
p2=PlotXY(bins(d),hist(d))
style=p2.getStyle()
style.setChartType(Style.HISTOGRAM)
p.close ()
p2.close ()
```

Example 3: Plot a histogram with BinCentres for a Double5d

```
from herschel.ia.numeric.toolbox.basic import Histogram
from herschel.ia.numeric.toolbox.basic import BinCentres
d = Double5d (5,4,3,2,1, 2.0)
d.set (4,3,2,1,0, 10.0)
d.set (3,3,2,1,0, 9.0)
d.set (2,3,2,1,0, 8.0)
d.set (1,3,2,1,0, 7.0)
binsize = 1.0
hist = Histogram (binsize)
bins = BinCentres (binsize)
p=PlotXY(bins(d),hist(d))
style=p.getStyle()
style.setChartType(Style.HISTOGRAM)
p.close ()
```

API Summary

Jython Syntax

```
bins = BinCentres(&lt;binsize&gt;)
&lt;c&gt; = bin(&lt;x&gt;)
```

Properties

Number **binsize** [INPUT, MANDATORY, default=no default value]
Array **x** [INPUT, MANDATORY, default=no default value]
Double1d **c** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Number binsize [INPUT, MANDATORY, default=no default value]

The size of the histogram bins.

Array x [INPUT, MANDATORY, default=no default value]

The data of which to compute the bins.

Double1d c [OUTPUT, MANDATORY, default=no default value]

The bin centres.

See also

- Histogram
- Developers Manual: herschel.ia.numeric.toolbox.basic.BinCentres

1.38. BinomialModel

Full Name:	herschel.ia.numeric.toolbox.fit.BinomialModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import BinomialModel
Category	Mathematics/Fitting

Description

General binomial model of arbitrary degree.

$$f(x,y:p) = \sum p_k * x^k * y^{degree - k}$$

where the sum is over k running from 0 to degree (inclusive).

It is a 2-dimensional linear model.

Examples of the generated polynomials assuming that input = (x,y):

Table 1.1. Polynomial table.

degree	nr param	polynomial
0	1	p_0
1	2	$p_0\ y + p_1\ x$
2	3	$p_0\ y^2 + p_1\ y\ x + p_2\ x^2$
3	4	$p_0\ y^3 + p_1\ y^2\ x + p_2\ y\ x^2 + p_3\ x^3$

Example

Example 1: BinomialModel

```
poly = BinomialModel( 3 )            # 3rd degree polynomial
print poly.getNumberOfParameters()       # 4
```

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.BinomialModel

1.39. Bool1d

Full Name:	herschel.ia.numeric.Bool1d
Type:	Java Class -
Import:	from herschel.ia.numeric import Bool1d
Category	Arrays and datasets

Description

A rectangular numeric boolean array of rank 1.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Bool1d

1.40. Bool2d

Full Name:	herschel.ia.numeric.Bool2d
Type:	Java Class -
Import:	from herschel.ia.numeric import Bool2d
Category	Arrays and datasets

Description

A rectangular numeric boolean array of rank 2.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Bool2d

1.41. Bool3d

Full Name:	herschel.ia.numeric.Bool3d
Type:	Java Class -
Import:	from herschel.ia.numeric import Bool3d
Category	Arrays and datasets

Description

A rectangular numeric boolean array of rank 3.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Bool3d

1.42. Bool4d

Full Name:	herschel.ia.numeric.Bool4d
Type:	Java Class -
Import:	from herschel.ia.numeric import Bool4d
Category	Arrays and datasets

Description

A rectangular numeric boolean array of rank 4.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Bool4d

1.43. Bool5d

Full Name:	herschel.ia.numeric.Bool5d
Type:	Java Class -
Import:	from herschel.ia.numeric import Bool5d
Category	Arrays and datasets

Description

A rectangular numeric boolean array of rank 5.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Bool5d

1.44. BoxCarFilter

Full Name:	herschel.ia.numeric.toolbox.filter.BoxCarFilter
Alias:	BoxCarFilter
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.filter import BoxCarFilter
Category	Mathematics/Signal processing

Description

Creates a box car filter, that can be applied to numeric arrays of rank 1 and 2.

Example

Example 1: Apply BoxCarFilter on a Int1d

```
x=Int1d([1,3,2,4,6,5])
f=BoxCarFilter(3)
print f(x) # [1.3333333333333333,1.9999999999999998,
           #  3.0,3.9999999999999996,5.0,3.6666666666666665]
```

API Summary

Jython Syntax

```
<f>=BoxCarFilter(<width> [,<center>=true|false] [,<edge>=Convolution.ZEROES|CIRCULAR|REPEAT])
<x>=<f>(<x>)
```

Properties

integer **width** [INPUT, MANDATORY, default=no default value]

boolean **center** [INPUT, OPTIONAL, default=true]

Convolution.ZEROES|CIRCULAR|REPEAT **edge** [INPUT, OPTIONAL, default=Convolution.ZEROES]

API details

Properties

integer width [INPUT, MANDATORY, default=no default value]

It must be an integer value

boolean center [INPUT, OPTIONAL, default=true]

Set center to true or false

Convolution.ZEROES|CIRCULAR|REPEAT edge [INPUT, OPTIONAL, default=Convolution.ZEROES]

Set edge to the following:

- edge=Convolution.ZEROES: Set result to zero at edges.

Convolution.ZEROES|CIRCULAR|REPEAT edge [INPUT, OPTIONAL, default=Convolution.ZEROES]

- edge=Convolution.CIRCULAR: Wrap around at edges (circular convolution).
- edge=Convolution.REPEAT: Repeat the edge values of input array.

See also

- Convolution
- GaussianFilter
- Developers Manual: herschel.ia.numeric.toolbox.filter.BoxCarFilter

History

- 2010-12-14 - AS: Add URM category; correct URM example.

1.45. boxCarSmoothing

Full Name:	herschel.ia.toolbox.image.BoxCarSmoothingTask
Alias:	boxCarSmoothing
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import BoxCarSmoothingTask
Category	Images/Analysis

Description

This is a task to smooth/filter an image using a convolution with a box car.

This is a task to smooth/filter an image using a convolution with a box car, for which the user must specify the width.

Example

Example 1: This is an example of how to use the boxCarSmoothing :

```
smoothed = boxCarSmoothing(image = myImage, width = 3)
```

API Summary

Properties
Image **image** [INPUT, MANDATORY, default=None]
Integer **width** [INPUT, OPTIONAL, default=3]
Image **smoothed** [OUTPUT, MANDATORY, default=None]

API details

Properties

Image image [INPUT, MANDATORY, default=None]

This is the input image.

Integer width [INPUT, OPTIONAL, default=3]

This is the width of the box car function in pixels.

Image smoothed [OUTPUT, MANDATORY, default=None]

This is the resulting smoothed/filtered image.

See also

- Developers Manual: herschel.ia.toolbox.image.BoxCarSmoothingTask

1.46. Byte1d

Full Name:	herschel.ia.numeric.Byte1d
Type:	Java Class -
Import:	from herschel.ia.numeric import Byte1d
Category	Arrays and datasets

Description

A rectangular numeric byte array of rank 1.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Byte1d

1.47. Byte2d

Full Name:	herschel.ia.numeric.Byte2d
Type:	Java Class -
Import:	from herschel.ia.numeric import Byte2d
Category	Arrays and datasets

Description

A rectangular numeric byte array of rank 2.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Byte2d

1.48. Byte3d

Full Name:	herschel.ia.numeric.Byte3d
Type:	Java Class -
Import:	from herschel.ia.numeric import Byte3d
Category	Arrays and datasets

Description

A rectangular numeric byte array of rank 3.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Byte3d

1.49. Byte4d

Full Name:	herschel.ia.numeric.Byte4d
Type:	Java Class -
Import:	from herschel.ia.numeric import Byte4d
Category	Arrays and datasets

Description

A rectangular numeric byte array of rank 4.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Byte4d

1.50. Byte5d

Full Name:	herschel.ia.numeric.Byte5d
Type:	Java Class -
Import:	from herschel.ia.numeric import Byte5d
Category	Arrays and datasets

Description

A rectangular numeric byte array of rank 5.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Byte5d

1.51. CachedPool

Full Name:	herschel.ia.pal.pool.cache.CachedPool
Type:	Java Class -
Import:	from herschel.ia.pal.pool.cache import CachedPool
Category	Data access

Description

Implementation of a PAL ProductPool that adds caching functionality to a remote ProductPool.

Example

Example 1: Creating a ProductStorage, allowing cached access to the HSA and saving to local disk.

```
cachedHsa=CachedPool( hsa )
local=LocalStoreFactory.getStore("local")
store=ProductStorage([local, cachedHsa])
```

API Summary

Constructors
CachedPool (ProductPool remote) Creates an instance of a CachedPool that is wrapped around another
CachedPool (ProductPool remote, ProductPool delegated) Creates an instance of a CachedPool that is wrapped around another

API Details

Constructors

CachedPool (ProductPool remote)

Creates an instance of a CachedPool that is wrapped around another

supplied ProductPool.

Argument

ProductPool **remote** [INPUT, MANDATORY, default=The ProductPool]

this CachedPool should cache.

Return

CachedPool

An instance of the CachedPool.

CachedPool (ProductPool remote, ProductPool delegated)

Creates an instance of a CachedPool that is wrapped around another

supplied ProductPool.

Arguments

CachedPool (ProductPool remote, ProductPool delegated)

`ProductPool` **`remote`** `[INPUT, MANDATORY, default=The ProductPool this CachedPool]`

should cache.

`ProductPool` **`delegated`** `[INPUT, MANDATORY, default=The ProductPool used to store ]`

the locally cached products.

Return

CachedPool

An instance of the CachedPool.

See also

- Developers Manual: herschel.ia.pal.pool.cache.CachedPool

1.52. calc_errcov

Full Name:	herschel.ia.toolbox.pointing.calc_errcov
Type:	Jython Task -
Import:	from herschel.ia.toolbox.pointing import calc_errcov
Category	Toolboxes/Pointing

Description

Calculates the (rotation angle) error covariance matrix.

Example

Example 1: from herschel.share.fltdyn.math import Quaternion

```
num_stars = 5
num_bad   = 1
id_bad    = Int1d(2)
stars_id  = Int1d(18)
stars_brf = Double2d(18,3)
measerr   = 1.0e-5
id_bad[0] = 345
for star in range(num_stars):
  stars_id[star]    =...
  stars_brf[star,:] =...
quat_align = Quaternion(0, 0, 0, 1)
(errcov_aca, status) = calc_errcov(num_stars, num_bad, id_bad, stars_id,
 stars_brf, \
                                    measerr, quat_align)
```

API Summary

Jython Syntax

```
(errcov_aca, status) = calc_errcov(num_stars, num_bad, id_bad,
stars_id, stars_brf, \
measerr, quat_align)
```

Properties
Int1d **num_stars** [INPUT, MANDATORY, default=NO default value]
Int1d **num_bad** [INPUT, MANDATORY, default=NO default value]
Int1d **id_bad** [INPUT, MANDATORY, default=NO default value]
Int1d **stars_id** [INPUT, MANDATORY, default=NO default value]
Double2d **stars_brf** [INPUT, MANDATORY, default=NO default value]
Double **measerr** [INPUT, MANDATORY, default=NO default value]
Quaternion **quat_align** [INPUT, MANDATORY, default=NO default value]
Double2d **errcov_aca** [OUTPUT, MANDATORY, default=NO default value]
Int1d **status** [OUTPUT, MANDATORY, default=NO default value]

API details

Properties

Int1d num_stars [INPUT, MANDATORY, default=NO default value]

No. of stars, n, with which to determine attitude

Int1d num_bad [INPUT, MANDATORY, default=NO default value]

Number of bad stars, n_b, which were eliminated from the estimation

Int1d id_bad [INPUT, MANDATORY, default=NO default value]

IDs of bad stars eliminated from the estimation

Int1d stars_id [INPUT, MANDATORY, default=NO default value]

IDs in star catalogue

Double2d stars_brf [INPUT, MANDATORY, default=NO default value]

Measured star vectors (BRF)

Double measerr [INPUT, MANDATORY, default=NO default value]

Measurement error (1 sigma, rad.)

Quaternion quat_align [INPUT, MANDATORY, default=NO default value]

Star tracker alignment quaternion

Double2d errcov_aca [OUTPUT, MANDATORY, default=NO default value]

(Rotation angle) error covariance matrix (ACA-frame)

Int1d status [OUTPUT, MANDATORY, default=NO default value]

Return status (0 = success, 1 = error)

History

- 2014-06-04 - CAS: Initial version
- 2014-06-06 - CAS: Import some much-needed modules.
- 2015-07-29 - CAS: HCSS-20528 The previous versions failed to take into account that certain stars had been deemed bad and consequently were not used in the estimation.

1.53. calcAttitude

Full Name:	herschel.ia.toolbox.pointing.CalcAttitudeTask
Alias:	calcAttitude
Type:	Jython Task -
Import:	from herschel.ia.toolbox.pointing import CalcAttitudeTask
Category	Toolboxes/Pointing

Description

Creates a new pointing product with reconstructed filterQuat.

Creates a new pointing product in which the filtered attitude quaternion, filterQuat, is replaced by the attitude quaternion computed by means of the gyro-based attitude reconstruction method. See HERSCHEL-HSC-TN-2069 for details.

Example

Example 1: obs = getObservation(1342246172, useHsa=True)

```
oldpp                  = obs.auxiliary.pointing
newpp                  = calcAttitude(oldpp, acmsProduct, tcHistoryProduct)
obs.auxiliary.pointing = newpp
```

API Summary

Jython Syntax

```
newpp = calcAttitude(oldpp)
```

Properties

PointingProduct **oldpp** [INPUT, MANDATORY, default=NO default value]
Double **prob_thresh** [INPUT, OPTIONAL, default=1.0e-4]
Int **star_set** [INPUT, OPTIONAL, default=0]
Double **ref_thresh** [INPUT, OPTIONAL, default=100.0]
Double **wind_len** [INPUT, OPTIONAL, default=400.0]
Double **rot_limit** [INPUT, OPTIONAL, default=0.5]
Double **toff_star** [INPUT, OPTIONAL, default=0.189]
PointingProduct **newpp** [OUTPUT, MANDATORY, default=NO default value]

API details

Properties

PointingProduct oldpp [INPUT, MANDATORY, default=NO default value]

Original pointing product from the observation context

Double prob_thresh [INPUT, OPTIONAL, default=1.0e-4]

Probability threshold used for deciding when fitting performed by calcStrAttitude is bad.

Int star_set [INPUT, OPTIONAL, default=0]

Stars to be used (0 = All, 1 = First nine, 2 = Last nine)

Double ref_thresh [INPUT, OPTIONAL, default=100.0]

The reference attitude is updated whenever it is found to differ by more than ref_thresh (arc-secs.) from the star tracker attitude measurement immediately prior to the current time.

Double wind_len [INPUT, OPTIONAL, default=400.0]

Nominal length of the time interval (moving window) containing the measurements used in the linear regressions performed by calcGyroAttitude.

Double rot_limit [INPUT, OPTIONAL, default=0.5]

All attitude measurements differing by more than rot_limit (deg.) from the reference attitude are excluded from the fitting.

Double toff_star [INPUT, OPTIONAL, default=0.189]

Time offset to be added to the OBTs of the star tracker attitude measurements before combining them with the gyro (angular) measurements (s).

PointingProduct newpp [OUTPUT, MANDATORY, default=NO default value]

New pointing product containing the gyro-based reconstructed attitude

History

- 2013-07-27 - BV: Initial version.
- 2013-08-13 - BV: Update ppVersion to 3.1 (v3 of STR distortions for late mission)
- 2013-09-26 - BV: HCSS-18567 Review disabled jexamples
- 2014-03-05 - BV: HCSS-18860 calcStrAttitude crashes when the 1 Hz and the 4 Hz data do not match
- 2014-03-05 - BV: HCSS-187331 Task compliancy
- 2014-06-10 - CAS: Updated to take into account changes to gyroAttitude and did some
- 2014-06-10 - CAS: Tidying-up to (hopefully) improve readability.
- 2014-06-23 - CAS: HCSS-19387 Added the various optional input parameters for calcStrAttitude and calcGyroAttitude to the calling interface.
- 2014-06-30 - CAS: HCSS-18749 Followed instructions in HCSS-18749 to make task compliant.
- 2014-10-03 - CAS: HCSS-19570 Updated to reflect recent changes to calcGyroAttitude.
- 2014-10-07 - CAS: HCSS-19454 Default value of toff_star updated.
- 2014-10-31 - CAS: HCSS-19121 To facilitate testing, the ability to exclude measurements from one of the four gyros has been added.

- 2014-11-18 - CAS: HCSS-19703 Input parameter prob_thresh now also passed to calcGyroAttitude.
- 2014-11-21 - CAS: HCSS-19386 Created new columns and populated with the probabilities and uncertainties.
- 2014-11-21 - CAS: HCSS-19386 Also handled the error cases associated with HCSS-19234 and HCSS-19721.
- 2015-07-13 - CAS: HCSS-19398 New input parameter passed to calcStrAttitude, which allows the selection of which set of star vector measurements is to be used.
- 2015-08-14 - CAS: HCSS-19267 Three new input parameters passed to calcStrAttitude.
- 2015-08-26 - CAS: HCSS-20333 Skip over the history dataset.
- 2015-10-05 - CAS: HCSS-20719 Code added to handle star tracker switchovers (where there are periods when STR-B was operational).
- 2016-20-09 - JPC: HCSS-21384 Consolidate CalcAttitudeTask to make use of equivalent functionality in auxprocessors

1.54. calcGyroAttitude

Full Name:	herschel.ia.toolbox.pointing.calcGyroAttitude
Type:	Jython Task -
Import:	from herschel.ia.toolbox.pointing import calcGyroAttitude
Category	Toolboxes/Pointing

Description

Constructs a table containing the reconstructed ACA-frame attitude.

See HERSCHEL-HSC-TN-2069 for further details.

Example

Example 1: obs = getObservation(1342246172, useHsa=True)

```
acmsProduct      = obs.auxiliary.acms
tcHistoryProduct = obs.auxiliary.teleCommHistory
oldpp            = obs.auxiliary.pointing
strAttitude      = calcStrAttitude(oldpp, acmsProduct, tcHistoryProduct)
gyroAttitude     = calcGyroAttitude(oldpp, acmsProduct, strAttitude)
```

API Summary

Jython Syntax

```
gyroAttitude = calcGyroAttitude(oldpp, acmsProduct, strAttitude)
```

Properties

PointingProduct **oldpp** [INPUT, MANDATORY, default=NO default value]
AcmsTelemetryProduct **acmsProduct** [INPUT, MANDATORY, default=NO default value]
TableDataset **strAttitude** [INPUT, MANDATORY, default=NO default value]
Double **ref_thresh** [INPUT, OPTIONAL, default=100.0]
Double **wind_len** [INPUT, OPTIONAL, default=400.0]
Double **rot_limit** [INPUT, OPTIONAL, default=0.5]
Double **toff_star** [INPUT, OPTIONAL, default=0.189]
Double **prob_thresh** [INPUT, OPTIONAL, default=1.0e-4]
Int **excl_gyro** [INPUT, OPTIONAL, default=0]
Int **debug** [INPUT, OPTIONAL, default=0]
TableDataset **gyroAttitude** [OUTPUT, MANDATORY, default=NO default value]

API details

Properties

PointingProduct oldpp [INPUT, MANDATORY, default=NO default value]

Original pointing product from the observation context

AcmsTelemetryProduct acmsProduct [INPUT, MANDATORY, default=NO default value]

ACMS telemetry product from the observation context

TableDataset strAttitude [INPUT, MANDATORY, default=NO default value]

Table containing corrected attitude measurements (from star tracker) See HERSCHEL-HSC-TN-2069 for details.

Double ref_thresh [INPUT, OPTIONAL, default=100.0]

The reference attitude is updated whenever it is found to differ by more than ref_thresh (arc-secs.) from the star tracker attitude measurement immediately prior to the current time.

Double wind_len [INPUT, OPTIONAL, default=400.0]

Nominal length of the time interval (moving window) containing the measurements used in the linear regression (s).

Double rot_limit [INPUT, OPTIONAL, default=0.5]

All attitude measurements differing by more than rot_limit (deg.) from the reference attitude are excluded from the fitting.

Double toff_star [INPUT, OPTIONAL, default=0.189]

Time offset to be added to the OBTs of the star tracker attitude measurements before combining them with the gyro (angular) measurements (s).

Double prob_thresh [INPUT, OPTIONAL, default=1.0e-4]

Probability threshold used for deciding when star tracker attitude is bad

Int excl_gyro [INPUT, OPTIONAL, default=0]

Number of the gyro from which measurements should be excluded. If set equal to zero, or any integer other than {1,2,3,4}, then measurements from all four gyros are used.

Int debug [INPUT, OPTIONAL, default=0]

Set equal to 1 for additional output (in gyroAttitude)

TableDataset gyroAttitude [OUTPUT, MANDATORY, default=NO default value]

Table containing reconstructed ACA-frame attitude. See HERSCHEL-HSC-TN-2069 for details.

History

- 2013-07-23 - BV: Initial version.

- 2013-07-30 - BV: Relax condition on window width for bias fit in slews.
- 2013-08-23 - BV: HCSS-18486 Take the reference x,y,z out of the slew. Use the same reference x,y,z for calculation back to ra,dec,roll.
- 2013-08-26 - BV: HCSS-18486 Use SLERP for STR attitude interpolation
- 2013-08-27 - BV: HCSS-18371 Correction for STR time stamps incorporated in calStrAttitude.
- 2013-08-27 - BV: Implementation of HCSS-18486 and HCSS-18371 was not complete.
- 2013-08-28 - BV: HCSS-18371 With str time offset, selection of start and end should be based on pointing product and available STR telemetry
- 2013-09-26 - BV: HCSS-18567 Review disabled jexamples
- 2014-06-09 - CAS: HCSS-***** Code replaced by a version developed by C.A. Stephenson.
- 2014-06-10 - CAS: Imported herschel.share.unit.
- 2014-06-10 - CAS: HCSS-18779 In association with this ticket, I have now added a column to gyroAttitude containing the maximum angular rotation from the reference attitude and allowed the User to select the length of the interval used to estimate the parameters. A 'dynamic reference attitude' had already been implemented on 09-Jun-2014.
- 2014-06-13 - CAS: HCSS-19373 Removed some unnecessary computations in order to speed up the processing (especially for long observations).
- 2014-06-17 - CAS: HCSS-19380 Added two optional input parameters, which allow the User to adjust both the relative timing between the gyro measurements and the star tracker attitude measurements and the absolute times of the reconstructed attitudes.
- 2014-06-18 - CAS: HCSS-18777 A fairly major re-write of the code in order to create the joint measurements (used in the linear regression) by interpolating the gyro (angular) measurements to the times of the star tracker attitude measurements. (Also modified the input parameter defining the length of the moving window.)
- 2014-06-18 - CAS: HCSS-18840 Correctly weighted the measurements using the variances computed in calcStrAttitude. Used the incomplete gamma function to interpret the values of the minimized cost functions as the probability that the 'fit is good'.
- 2014-06-19 - CAS: HCSS-19369 Exclude all attitude measurements differing by more than a rot_limit (optional parameter) from the fitting. This is to ensure that the small-angle approximation remains valid.
- 2014-06-19 - CAS: HCSS-19291 Output the (one-sigma) uncertainties in the reconstructed attitude about each of the ACA-frame axes. (The column showing the maximum angular rotation from the reference attitude is now only output if debug = 1.)
- 2014-06-23 - CAS: Added some logic to handle the cases where there are only one or two measurements available (for the parameter estimation).
- 2014-10-02 - CAS: HCSS-19566 Simplified specification of time offsets, replacing toff_gyro and toff_recon by toff_star.
- 2014-10-03 - CAS: HCSS-19570 Whilst fixing HCSS-19570, I have taken the opportunity of updating the default value of toff_star in the header.
- 2014-10-07 - CAS: HCSS-19454 Modified the default value of toff_star to take into account both the optimal value of toff_gyro (0.061), as calculated by D. Lutz (6/10/14), and the modification to calcStrAttitude associated with HCSS-19454.

- 2014-10-08 - CAS: HCSS-19579 The estimated drift rates and their associated uncertainties are now output whenever debug = 1.
- 2014-10-31 - CAS: HCSS-19121 To facilitate testing, the ability to exclude measurements from one of the four gyros has been added.
- 2014-11-18 - CAS: HCSS-19703 The new input parameter prob_thresh is now used to exclude bad quality measurements (in particular the identity quaternions corresponding to when it was not possible to estimate the star tracker attitude) from the fitting. They are also never used to update the reference attitude, but may still be used for the initial reference attitude (in this case a warning is issued).
- 2014-11-26 - CAS: HCSS-19839 Only overwrite the reconstructed attitude with the identity quaternion when we have been unable to estimate it. In such cases also set the standard deviations equal to zero.
- 2015-04-01 - CAS: HCSS-19775 If the initial star tracker attitude measurement is of bad quality (which may indicate that there is no measurement), then the reference attitude is initialized to the initial on-board attitude estimate.

1.55. calcStrAttitude

Full Name:	herschel.ia.toolbox.pointing.calcStrAttitude
Type:	Jython Task -
Import:	from herschel.ia.toolbox.pointing import calcStrAttitude
Category	Toolboxes/Pointing

Description

Constructs a table containing the corrected attitude measurements made by the (operational) star tracker.

See HERSCHEL-HSC-TN-2069 for further details.

Example

Example 1: obs = getObservation(1342246172, useHsa=True)

```
acmsProduct      = obs.auxiliary.acms
tcHistoryProduct = obs.auxiliary.teleCommHistory
oldpp            = obs.auxiliary.pointing
strAttitude      = calcStrAttitude(oldpp, acmsProduct, tcHistoryProduct)
```

API Summary

Jython Syntax

```
strAttitude = calcStrAttitude(oldpp, acmsProduct, tcHistoryProduct)
```

Properties

PointingProduct **oldpp** [INPUT, MANDATORY, default=NO default value]
AcmsTelemetryProduct **acmsProduct** [INPUT, MANDATORY, default=NO default value]
TeleCommandHistProduct **tcHistoryProduct** [INPUT, MANDATORY, default=NO default value]
Double **prob_thresh** [INPUT, OPTIONAL, default=1.0e-4]
Double **prob_fac** [INPUT, OPTIONAL, default=100.0]
Int **star_set** [INPUT, OPTIONAL, default=0]
Int **back_prop** [INPUT, OPTIONAL, default=2]
Double **measerr_init** [INPUT, OPTIONAL, default=3.0]
Double **alpha** [INPUT, OPTIONAL, default=0.1]
Int **debug** [INPUT, OPTIONAL, default=0]
TableDataset **strAttitude** [OUTPUT, MANDATORY, default=NO default value]
Int **status** [OUTPUT, MANDATORY, default=NO default value]

API details

Properties

PointingProduct oldpp [INPUT, MANDATORY, default=NO default value]

Original pointing product from the observation context

AcmsTelemetryProduct acmsProduct [INPUT, MANDATORY, default=NO default value]

ACMS telemetry product from the observation context

TeleCommandHistProduct tcHistoryProduct [INPUT, MANDATORY, default=NO default value]

TC history product from the observation context

Double prob_thresh [INPUT, OPTIONAL, default=1.0e-4]

Probability threshold used for deciding when fit is bad

Double prob_fac [INPUT, OPTIONAL, default=100.0]

A star is considered bad iff the p-value with the star excluded is more than prob_fac times greater than the p-value with it present

Int star_set [INPUT, OPTIONAL, default=0]

Stars to be used (0 = All, 1 = First nine, 2 = Last nine)

Int back_prop [INPUT, OPTIONAL, default=2]

Stars to be back-propagated (rotated back) prior to amendment of distortion correction (0 = None, 1 = First nine, 2 = Last nine)

Double measerr_init [INPUT, OPTIONAL, default=3.0]

Initial estimate of star vector measurement error (1-sigma, arcsec.)

Double alpha [INPUT, OPTIONAL, default=0.1]

Smoothing factor for low-pass filtering of estimated measurement error

Int debug [INPUT, OPTIONAL, default=0]

Set equal to 1 for additional output (in strAttitude)

TableDataset strAttitude [OUTPUT, MANDATORY, default=NO default value]

Table containing corrected attitude measurements (from star tracker) See HERSCHEL-HSC-TN-2069 for details.

Int status [OUTPUT, MANDATORY, default=NO default value]

Return status (0 = Success, 1 = ACMS telemetry product does not exist, 2 = TC history product does not exist, 3 = No/empty STR-specific diagnostics telemetry dataset, 4 = No/empty main

Int status [OUTPUT, MANDATORY, default=NO default value]
ACMS dataset, 5 = No STR-specific diagnostics telemetry during time interval spanned by old pointing product)

History

- 2013-04-09 - BV: Initial version
- 2013-04-10 - BV: Added imports for hipe integration
- 2013-06-10 - BV: Correct reference focal length / subtract color correction / swap indexes distortion map
- 2013-07-18 - BV: HCSS-18355 Change name of STR diagnostic dataset
- 2013-07-18 - BV: HCSS-18365 Smoothing of residual distortion corrections may fail
- 2013-07-18 - BV: HCSS-18388 Improvements in documentation
- 2013-07-29 - BV: HCSS-18425 Velocity vector projection is not correct
- 2013-07-29 - BV: HCSS-18039 Robustness against inconsistent 1Hz and 4Hz data
- 2013-07-31 - BV: HCSS-18431 Alerts about key input data that are missing
- 2013-08-01 - BV: Include herschel import explicitly
- 2013-08-02 - BV: HCSS-18433 Incorrect rotation of spacecraft velocity vector
- 2013-08-02 - BV: HCSS-18364 Indices in distortion map were swapped
- 2013-08-05 - BV: HCSS-18364 used FIX instead of ROUND to find index in distortion map
- 2013-08-08 - BV: HCSS-18348 Read alpha-C from ACMS product if available
- 2013-08-27 - BV: HCSS-18371 Attitudes of calcStrAttitude have incorrect time stamps
- 2013-08-27 - BV: HCSS-18480 Test for valid STR ID <3600 instead of <4000
- 2013-08-27 - BV: HCSS-18371 Time offset also applied for selection of packets to be used
- 2013-09-26 - BV: HCSS-18431 Check that at least two star vectors are found
- 2013-09-26 - BV: HCSS-18365 Re-implementation using residual distortion values within circle of 2392 subpixels radius only.
- 2013-09-26 - BV: HCSS-18388 Improve documentation + add units to output product
- 2013-09-26 - BV: HCSS-18567 Review disabled jexamples
- 2013-09-27 - BV: HCSS-18388 Corrected typo (time offset deleted) introduced with previous fix
- 2013-10-08 - BV: HCSS-18388 re-implementation: inaccurate self-documentation: correct comment
- 2013-10-08 - BV: HCSS-18431 re-implementation: if less than 2 stars used: no q-method and correct message
- 2013-10-08 - BV: HCSS-18566 Add a quality indicator to calcStrAttitude. Add an iteration to omit tracking stars that result in a large TASTE (cfr Shuster 2006)

- 2014-06-06 - CAS: HCSS-***** Code replaced by a version developed by C.A. Stephenson. Much of the processing has been farmed-out to the functions est_attitude, qMethod and calc_errcov. There have also been some significant changes made to the contents of the output table strAttitude.

- 2014-06-10 - CAS: The setting of acmsScm1hzRecordTime for the case where the '1Hz and 4HZ diagnostic ACMS do not match' had somehow got lost.

- 2014-06-20 - CAS: HCSS-18566 Column(s) giving the probability that the 'fit is good', i.e. that a value of TASTE as large as the value returned could have occurred by chance, have been added. A probability threshold is now used to decide when the fit is bad.

- 2014-06-23 - CAS: Corrected the description of the nominal value of the input parameter prob_thresh.

- 2014-10-07 - CAS: HCSS-19454 Modified array indexing in order to bring the case where the "1Hz and 4Hz diagnostic ACMS match" in line with the case where the "1Hz and 4Hz diagnostic ACMS do not match".

- 2014-10-28 - CAS: HCSS-18371 Removed on-board filtered attitudes from the output table.

- 2014-10-28 - CAS: HCSS-19589 The OBT is now taken from the STR-specific diagnostics dataset (always [when available]). It is noted that the timestamps of the on-board star tracker attitudes may sometimes be out by 0.5 sec (or so). Given that they are only used in connection with the aberration correction, this is not considered to be of any great importance.

- 2014-11-13 - BV: HCSS-19234 In absense of STR-diagnostic telemetry, calcStrAttitude should fail.

- 2014-11-17 - CAS: HCSS-19703 Write 'default' values, e.g. identity quaternions, to the output table whenever we are unable to estimate the attitude (which normally means when there are too few stars in the field-of-view). Transform the spacecraft velocity vector into the BRF only when the on-board s/w has been able to determine the star tracker attitude. (For this I simply check that between 2 and 18 stars were detected on-board.)

- 2014-11-17 - CAS: HCSS-19756 Star measurements are discarded, and therefore no attempt is made to obtain the residual distortion correction, whenever the centroid is found to lie outside of the central (operational) disc of the field-of-view.

- 2014-11-20 - CAS: HCSS-19234 A status is now returned from this function to indicate HCSS-19721 and HCSS-19782 possible errors.

- 2015-02-03 - CAS: HCSS-19968 A quick fix added to avoid attempting to use an invalid (all components zero) quaternion to transform the spacecraft velocity vector into BRF. It would be nicer to use the appropriate parameter from telemetry, but these parameters are not sufficiently well understood.

- 2015-02-12 - CAS: HCSS-20061 Made sure that the list of IDs passed to function est_attitude is correct (i.e. that it does not contain any of the IDs in the 'bad star' list).

- 2015-07-13 - CAS: HCSS-19398 Extended to allow the use of the additional star vector. HCSS-20392 measurements which are sometimes available when interlacing mode was active. Also extended to allow either of the two sets of measurements (i.e. those in positions 1--9 or those in positions 10--18) to be back-propagated (whenever interlacing mode was active) prior to amending the distortion correction. This latter flexibility was added as it was at the time not known which of these two sets of measurements came from the previous ACMS cycle. We now know it to be the second set. Star IDs now read as Double and later converted to Integer to avoid problem described in HCSS-20392.

- 2015-07-29 - CAS: HCSS-20528 The calling interface to function calc_errcov has changed as this function now takes into account the fact that there may be bad stars.

- 2015-08-12 - CAS: HCSS-19267 The star vector measurement error is now estimated 'on the fly' and a new method of detecting bad stars (now up to five) has been introduced.
- 2015-08-24 - CAS: HCSS-20490 In the case of repeated measurements of the same star, only the first will be used. This will be the (first) measurement from the current ACMS cycle.
- 2015-09-10 - CAS: HCSS-20647 If we are unable to estimate the star tracker attitude, then set the number of stars used equal to zero.
- 2015-09-10 - CAS: HCSS-20650 A line of code has been added to transform the rotation quaternion associated with the back-propagation of star vectors from the ACA-frame to the Bore-sight Reference Frame.
- 2015-09-29 - CAS: HCSS-20719 Code added to handle SVV resets (where there are periods when the on-board spacecraft velocity vector differed from the correct one).

1.56. CauchyErrorDistribution

Full Name:	herschel.ia.numeric.toolbox.fit.sample.CauchyErrorDistribution
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit.sample import CauchyErrorDistribution
Category	Mathematics/Fitting

Description

CauchyErrorDistribution aka Lorentz distribution.

Cauchy distr : $f(x) = s / (\pi * (s^2 + x^2))$

where x = residual and s = scale

For a simple way to choose the distribution in the NestedSampler use NestedSampler.setCauchyDistribution().

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.sample.CauchyErrorDistribution

1.57. CEIL

Full Name:	herschel.ia.numeric.toolbox.basic.Ceiling
Alias:	CEIL
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Ceiling
Category	Mathematics/Nearest integer

Description

Returns the largest integer greater than or equal to the input.

If the input is an array, the output is an array of the same type and size, with the computed values instead of the original elements.

Example

Example 1: Applying CEIL to a Float1d

```
x = Float1d([0.1, 0.5, 0.9])
print CEIL(x) # [1.0,1.0,1.0]
```

API Summary

Jython Syntax

```
&lt;y&gt; = CEIL(&lt;x&gt;)
```

Properties

Number or array **x** [INPUT, MANDATORY, default=no default value]

Number or array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Number or array x [INPUT, MANDATORY, default=no default value]

The value or values of which to compute the ceiling. Complex numbers are not allowed.

Number or array y [OUTPUT, MANDATORY, default=no default value]

The ceiling value or values.

See also

- FLOOR
- ROUND
- Developers Manual: herschel.ia.numeric.toolbox.basic.Ceiling

1.58. ChebyshevPolynomialModel

Full Name:	herschel.ia.numeric.toolbox.fit.ChebyshevPolynomialModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import ChebyshevPolynomialModel
Category	Mathematics/Fitting

Description

Chebyshev polynomial model of arbitrary degree.

$$f(x:p) = \sum p_k * T_k(x)$$

where the sum is over k running from 0 to degree (inclusive).

The T(x) are Chebyshev polynomials of the first kind which are defined recursively as:

$T_0(x) = 1$

$T_1(x) = x$

$T_n(x) = 2 x T_{n-1}(x) - T_{n-2}(x)$ for n >= 2

These polynomials are orthogonal, only when x is in [-1,+1].

It is a linear model.

See example

Example

Example 1: ChebyshevPolynomialModel

```
poly = ChebyshevPolynomialModel( 3 )         # 3rd degree polynomial
print poly.getNumberOfParameters()        # 4
```

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.ChebyshevPolynomialModel

1.59. ChiSquared

Full Name:	herschel.ia.numeric.toolbox.stat.ChiSquared
Alias:	ChiSquared
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.stat import ChiSquared
Category	Mathematics/Statistics

Description

ChiSquared computes the chi^2 and significance of the null hypothesis that an

observed set of frequencies, X, is drawn from the same distribution as an expected set of frequencies, Y. A high chi^2 indicates that it is unlikely. The significance of the result is also computed, where a small value suggests that the two distributions are quite different. The code used to determine these values is inspired by section 14.3 in Numerical Recipes.

Example

Example 1: ChiSquared

```
mF=Double1d([10,12,19,20])
eF=Double1d([11,11,20,20])
params=ChiSquared(mF,eF,1)
chi2=params.getChiSquared()
sig=params.getSignifcance()
df=params.getDegreesOfFreedom()
print chi2 # 0.2318181818181818
print sig  # 0.9722963136322458
print df   # 3
params=ChiSquared(mF,eF) #the number of constrains default to 1
chi2=params.getChiSquared()
sig=params.getSignifcance()
df=params.getDegreesOfFreedom()
```

API Summary

Jython Syntax

```
params = ChiSquared(x,y,nc)
```

Properties

Numeric1dData **x** [INPUT, MANDATORY, default=no default value]
Numeric1dData **y** [INPUT, MANDATORY, default=no default value]
Integer **nc** [INPUT, MANDATORY, default=default value=1]

API details

Properties

Numeric1dData x [INPUT, MANDATORY, default=no default value]

measured frequency, one dimensional numeric array

Numeric1dData y [INPUT, MANDATORY, default=no default value]
expected frequency, one dimensional numeric array

Integer nc [INPUT, MANDATORY, default=default value=1]
number of constraints

See also

- Developers Manual: herschel.ia.numeric.toolbox.stat.ChiSquared

1.60. CholeskyDecomposition

Full Name:	herschel.ia.numeric.toolbox.matrix.CholeskyDecomposition
Alias:	CholeskyDecomposition
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.matrix import CholeskyDecomposition
Category	Mathematics/Matrices

Description

CholeskyDecomposition implements the Cholesky method for decomposing a matrix.

Example

Example 1: ##

```
#####   CholeskyDecomposition    #####
########################################
#
### HIPE ###
A=Double2d (\
    [\
       [3944.2141901709658, -39.83013886638742, 24.856149542676924,
 -49.90362121047403],\
       [-39.83013886638742, 3958.7850982903415, 39.244285251565444,
 37.29778768650653],\
       [24.856149542676924, 39.244285251565444, 3952.2910884701473,
 -1.216735893731549],\
       [-49.90362121047403, 37.29778768650653, -1.216735893731549,
 3950.7089115298527],\
    ])
B = Double2d (\
[\
   [1184.8157857233916],\
   [-360.85066234104187],\
   [-426.7929837904702],\
   [-179.95015720518913],\
])
CholeskyD = CholeskyDecomposition(A)
solution = B.apply(CholeskyD)
print solution
[
[0.29968672246274],
[-0.08666981085100102],
[-0.10902299389485229],
[-0.04097866183941969]
]
isSpd = CholeskyD.isSpd
print isSpd
1
triangularFactor = CholeskyD.l
print triangularFactor
[
[62.8029791504429,0.0,0.0,0.0],
[-0.6342077940438997,62.91568070651636,0.0,0.0],
[0.39577978431778954,0.6277495758130477,62.86286962351098,0.0],
[-0.7946059547737568,0.5848119575853922,-0.020192561781548726,62.84691798442629]
]
```

API Summary

Jython Syntax
A=Double2d() B=Double2d() res=B.apply(CholeskyDecomposition(A)) CholeskyD = CholeskyDecomposition(A) solution = B.apply(CholeskyD) isSpd = CholeskyD.isSpd triangularFactor = CholeskyD.l

Property
Double2d **A** [INPUT, MANDATORY, default=no default value]

API details

Property

Double2d A [INPUT, MANDATORY, default=no default value]
Input must be a Double2d or Float2d array.

See also

- Developers Manual: herschel.ia.numeric.toolbox.matrix.CholeskyDecomposition

1.61. circleHistogram

Full Name:	herschel.ia.toolbox.image.CircleHistogramTask
Alias:	circleHistogram
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import CircleHistogramTask
Category	Images/Analysis

Description

This is a task to make a histogram of a region of interest on an image which is bounded by a circle.

This is a task to make a histogram of a region of interest on an image which is bounded by a circle. You must specify the number of bins for the histogram. By default the cut levels of the image will be used to construct the histogram, but it is also possible for you to specify the low and high cut level for the histogram. To define the position of the circle, you must specify the pixel or sky coordinates of the center of the center of the circle and its radius.

Example

Example 1: This is an example of how you can use the circleHistogram :

```
histogram = circleHistogram(image = myImage, bins = 200, lowCut = 100.0,
 highCut = 150.0,
                             centerX = 100.0, centerY = 230.0, radiusArcsec =
 50.0)
histogram = circleHistogram(image = myImage, bins = 200, lowCut = 100.0,
 highCut = 150.0,
                             centerRa = "05:46:45.9", centerDec = "-51:07:57.0",
                             radiusPixels = 50.0)
```

API Summary

Properties
Image **image** [INPUT, MANDATORY, default=None]
Double **lowCut** [INPUT, OPTIONAL, default=NaN]
Double **highCut** [INPUT, OPTIONAL, default=NaN]
Integer **bins** [INPUT, OPTIONAL, default=2000]
Double **centerX** [INPUT, OPTIONAL, default=NaN]
Double **centerY** [INPUT, OPTIONAL, default=NaN]
String **centerRa** [INPUT, OPTIONAL, default="centerRa"]
String **centerDec** [INPUT, OPTIONAL, default="centerDec"]
Double **radiusPixels** [INPUT, OPTIONAL, default=NaN]
Double **radiusArcsec** [INPUT, OPTIONAL, default=NaN]
String **centerRA** [INPUT, OPTIONAL, default="centerRA"]
CircleHistogramProduct **histogram** [OUTPUT, MANDATORY, default=None]

API details

Properties

Image image [INPUT, MANDATORY, default=None]

This is the input image.

Double lowCut [INPUT, OPTIONAL, default=NaN]

This is the low cut level you want to use for the histogram. If this is not specified, the image cut levels will be used.

Double highCut [INPUT, OPTIONAL, default=NaN]

This is the high cut level you want to use for the histogram. If this is not specified, the image cut levels will be used.

Integer bins [INPUT, OPTIONAL, default=2000]

This is the number of bins in the histogram.

Double centerX [INPUT, OPTIONAL, default=NaN]

The x-pixel-coordinate of the center of the circle.

Double centerY [INPUT, OPTIONAL, default=NaN]

The y-pixel-coordinate of the center of the circle.

String centerRa [INPUT, OPTIONAL, default="centerRa"]

This is the right ascension of the center of the circle. Using this only makes sense if the input image has a valid Wcs attached to it. The input format for the right ascension is "hh mm ss.s", "hh:mm:ss.s" or "hh.hhh".

String centerDec [INPUT, OPTIONAL, default="centerDec"]

This is the declination of the center of the circle. Using this only makes sense if the input image has a valid Wcs attached to it. The input format for the declination is "[+/-]dd mm ss.s", "[+/-]dd:mm:ss.s" or "[+/-]dd.ddd".

Double radiusPixels [INPUT, OPTIONAL, default=NaN]

This is the radius of the circle in pixels.

Double radiusArcsec [INPUT, OPTIONAL, default=NaN]

This is the radius of the circle in arcsec.

String centerRA [INPUT, OPTIONAL, default="centerRA"]

This is the right ascension of the center of the circle. Using this only makes sense if the input image has a valid Wcs attached to it. The input format for the right ascension is "hh mm ss.s", "hh:mm:ss.s" or "hh.hhh". Deprecated : Use centerRa

CircleHistogramProduct histogram [OUTPUT, MANDATORY, default=None]

This is the resulting histogram.

See also

- Developers Manual: herschel.ia.toolbox.image.CircleHistogramTask

1.62. CircleHistogramProduct

Full Name:	herschel.ia.dataset.image.CircleHistogramProduct
Type:	Java Class -
Import:	from herschel.ia.dataset.image import CircleHistogramProduct
Category	Images/Display

Description

A class to deal with the results of a circle histogram.

API Summary

Constructor
`CircleHistogramProduct`
The constructor of a new CircleHistogramProduct.

Methods
`setCenter (double centerX, double centerY)`
Sets the center for this CircleHistogramProduct to the given pixel coordinates.
`setCenter (double centerX, double centerY, String centerRA, String centerDec)`
Sets the center for this CircleHistogramProduct to the given pixel and sky coordinates.
`setRadius (double pixels)`
Sets the radius for this CircleHistogramProduct to the given number of pixels.
`setRadius (double pixels, double arcsec)`
Sets the radius for this ImageHistogramProduct to the given number of pixels and arcsec.
`Double1d` **`getCenterPixelCoordinates`**
Returns the center for this CircleHistogramProduct in pixel coordinates.
`String1d` **`getCenterSkyCoordinates`**
Returns the center for this CircleHistogramProduct in sky coordinates.
`double` **`getRadiusPixels`**
Returns the radius for this CircleHistogramProduct in pixels.
`double` **`getRadiusArcsec`**
Returns the radius for this CircleHistogramProduct in arcsec.

API Details

Constructor

`CircleHistogramProduct`
The constructor of a new CircleHistogramProduct.

Methods

`setCenter (double centerX, double centerY)`
Sets the center for this CircleHistogramProduct to the given pixel coordinates.

setCenter (double centerX, double centerY)

Arguments

`double` **centerX** `[INPUT, MANDATORY, default=no default value]`

The x-pixel-coordinate of the center, as double

`double` **centerY** `[INPUT, MANDATORY, default=no default value]`

The y-pixel-coordinate of the center, as double

setCenter (double centerX, double centerY, String centerRA, String centerDec)

Sets the center for this CircleHistogramProduct to the given pixel and sky coordinates.

Arguments

`double` **centerX** `[INPUT, MANDATORY, default=no default value]`

The x-pixel-coordinate of the center, as double

`double` **centerY** `[INPUT, MANDATORY, default=no default value]`

The y-pixel-coordinate of the center, as double

`String` **centerRA** `[INPUT, MANDATORY, default=no default value]`

The right ascension of the center, as String

`String` **centerDec** `[INPUT, MANDATORY, default=no default value]`

The declination of the center, as String

setRadius (double pixels)

Sets the radius for this CircleHistogramProduct to the given number of pixels.

Argument

`double` **pixels** `[INPUT, MANDATORY, default=no default value]`

The radius in pixels, as double

setRadius (double pixels, double arcsec)

Sets the radius for this ImageHistogramProduct to the given number of pixels and arcsec.

Arguments

`double` **pixels** `[INPUT, MANDATORY, default=no default value]`

The radius in pixels, as double

`double` **arcsec** `[INPUT, MANDATORY, default=no default value]`

The radius in arcsec, as double

Double1d **getCenterPixelCoordinates**

Returns the center for this CircleHistogramProduct in pixel coordinates.

Return

Double1d

Returns the center for this CircleHistogramProduct in pixel coordinates.

String1d **getCenterSkyCoordinates**

Returns the center for this CircleHistogramProduct in sky coordinates.

Return

String1d

Returns the center for this CircleHistogramProduct in sky coordinates.

double **getRadiusPixels**
Returns the radius for this CircleHistogramProduct in pixels. **Return** **double** Returns the radius for this CircleHistogramProduct in pixels.

double **getRadiusArcsec**
Returns the radius for this CircleHistogramProduct in arcsec. **Return** **double** Returns the radius for this CircleHistogramProduct in arcsec.

See also

- Developers Manual: herschel.ia.dataset.image.CircleHistogramProduct

1.63. clamp

Full Name:	herschel.ia.toolbox.image.ClampTask
Alias:	clamp
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import ClampTask
Category	Images/Analysis

Description

This is a task which restricts the range of intensity given low and high values in an image.

All intensity values below "low" are set to low, and all intensity values above "high" are set to "high".

Example

Example 1: This is how you can clamp an image betweem 11 and 240 :

```
clamped  = clamp(image = myImage, low = 11, high = 240)
```

API Summary

Jython Syntax

```
clamp(image, 11, 240)
```

Properties

Image **image** [INPUT, MANDATORY, default=None]

double **low** [INPUT, OPTIONAL, default=None]

double **high** [INPUT, OPTIONAL, default=None]

Image **clampedImage** [OUTPUT, MANDATORY, default=None]

API details

Properties

Image image [INPUT, MANDATORY, default=None]

This is the input image

double low [INPUT, OPTIONAL, default=None]

This is the low clamp value.

double high [INPUT, OPTIONAL, default=None]

This is the high clamp value.

Image clampedImage [OUTPUT, MANDATORY, default=None]

This is the resulting clamped image.

See also

- Developers Manual: herschel.ia.toolbox.image.ClampTask

1.64. clear

Full Name:	herschel.ia.toolbox.util.ClearTask
Alias:	clear
Type:	Java Task -
Import:	from herschel.ia.toolbox.util import ClearTask
Category	Session utilities

Description

Remove variables from the Jython session.

The clear task is used to clear a variable from the Jython session and reclaim the allocated memory. The task allows also removing all variables with the exception of reserved names. When used as in the following "clear()" deletes all parameters created by the user - including those from user setup files. Note that also, compared to 'del', it will not fail if some of the passed names do not exist. As it only removes variables, it will also protect you from inadvertently removing tasks, for example.

Examples

Example 1: clear a single variable

```
clear("variableName")
```

Example 2: clear a list of variables

```
clear("variableA, variableB, variableC")
```

Example 3: clear all variables created by the user - including those from user set up files

```
clear()
```

Example 4: Other forms of clear all

```
clear(all=True)
```

API Summary

Jython Syntax

```
clear([&lt;variable&gt;, &lt;all&gt;]) #general form
clear("variableName")
clear("variableA,variableB")
clear(all=True)
clear() # shortest form of the previous call
```

Properties

String **variable** [INPUT, OPTIONAL, default=no default value]
Boolean **all** [INPUT, OPTIONAL, default=True]

Limitations

- "variable" parameter has precedence over "all" parameter: if variable has a value, all is ignored.
- It just prints a warning if called when Jython is not running (not from the command line)

API details

Properties

String variable [INPUT, OPTIONAL, default=no default value]
The name of the variable to erase or a comma separated list of variables

Boolean all [INPUT, OPTIONAL, default=True]
The all option for erasing every variable

See also

- Developers Manual: herschel.ia.toolbox.util.ClearTask

History

- `2004-07-13 - NdC`: first release
- `2013-06-10 - JDS`: null not allowed, all default is false

1.65. ComboModel

Full Name:	herschel.ia.numeric.toolbox.fit.ComboModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import ComboModel
Category	Mathematics/Fitting

Description

ComboModel combines a number of copies of the same basic model.

Fixed relations can be set between similar parameters. The relations can be either multiplicative or additive. When these relations are set, they must be set for all models.

$$f(x:p) = \Sigma g(x:p)$$

where g(x:p) is a model (e.g. GaussModel)

See example

Example

Example 1: ComboModel

```
gauss = GaussModel()
combo = ComboModel( gauss, 3 )            # make a triplet
print combo
# take any combination or variation of the following
combo.setAddCombo( 1, Double1d([0,1,3]))  # define relative offsets
combo.setMulCombo( 2, Double1d( 3, 1.0 ))  # all the same widths
combo.setMulCombo( 0, Double1d([1.3,1,5])) # relative amplitudes
print combo.getNumberOfParameters()
print combo( Double1d.range(11)-5 )         # combo of 3 gauss between [-5,5]
# ... fitter etc. see LevenbergMarquardtFitter
```

Limitations

1. When all parameters are left free, precise initial parameters are needed to converge to the global optimum. 2. The model seems to be especially unstable when the basic models are overlapping. Fixing the widths relative to each other, helps. 3. Using a PolynomialModel (or similar ones) as basic model, is not going to work.

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.ComboModel

1.66. Complex1d

Full Name:	herschel.ia.numeric.Complex1d
Type:	Java Class -
Import:	from herschel.ia.numeric import Complex1d
Category	Arrays and datasets

Description

A rectangular numeric Complex array of rank 1.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Complex1d

1.67. Complex2d

Full Name:	herschel.ia.numeric.Complex2d
Type:	Java Class -
Import:	from herschel.ia.numeric import Complex2d
Category	Arrays and datasets

Description

A rectangular numeric Complex array of rank 2.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Complex2d

1.68. Complex3d

Full Name:	herschel.ia.numeric.Complex3d
Type:	Java Class -
Import:	from herschel.ia.numeric import Complex3d
Category	Arrays and datasets

Description

A rectangular numeric Complex array of rank 3.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Complex3d

1.69. Complex4d

Full Name:	herschel.ia.numeric.Complex4d
Type:	Java Class -
Import:	from herschel.ia.numeric import Complex4d
Category	Arrays and datasets

Description

A rectangular numeric Complex array of rank 4.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Complex4d

1.70. Complex5d

Full Name:	herschel.ia.numeric.Complex5d
Type:	Java Class -
Import:	from herschel.ia.numeric import Complex5d
Category	Arrays and datasets

Description

A rectangular numeric Complex array of rank 5.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Complex5d

1.71. compress

Full Name:	herschel.ia.toolbox.util.CompressTask
Alias:	compress
Type:	Java Task -
Import:	from herschel.ia.toolbox.util import CompressTask
Category	Session utilities

Description

Compresses files and directories.

Compresses into a (relative path) archive a user-chosen file or directory (and its contents). All arguments are mandatory. Does not support refreshing (updating).

Example

Example 1: Zipping a file with GZIP

```
compress("deleteMe", "./myzip.gzip", "GZ")
```

API Summary

Jython Syntax

```
compress(&lt;inputpath&gt;, &lt;archive&gt;, &lt;compression&gt;)
```

Properties

```
String inputpath [INPUT, MANDATORY, default=null]
String archive [INPUT, MANDATORY, default=null]
String compression [INPUT, OPTIONAL, default=&quot;ZIP&quot;]
```

Limitations

Does not support refreshing (updating) contents.

API details

Properties

`String inputpath [INPUT, MANDATORY, default=null]`

Path of the file or directory to be compressed. Must exist.

`String archive [INPUT, MANDATORY, default=null]`

Path of the archive to be produced. Must not exist.

`String compression [INPUT, OPTIONAL, default=&quot;ZIP&quot;]`

Type of compression to apply. Possible values:

- "ZIP" (use zip, default)

String compression [INPUT, OPTIONAL, default="ZIP"]

- "TAR" (use tar)
- "GZ" (use gzip, only accepts a file)
- "TGZ" ((use tar and then gzip the tar archive)

See also

- decompress
- Developers Manual: herschel.ia.toolbox.util.CompressTask

History

- `2010-08-17 - JDS`: first release

1.72. computePVMap

Full Name:	herschel.ia.toolbox.cube.ComputePVMapTask
Alias:	computePVMap
Type:	Java Task -
Import:	from herschel.ia.toolbox.cube import ComputePVMapTask
Category	Data cubes/Analysis

Description

This task computes a Position-Velocity (PV) map, which is a cross-section of a spectral cube along a given slit.

The map shows a 2D image the flux as function of velocity and spatial position along a slit. The spatial position (in arcsec) runs along the horizontal axis and the velocity along the vertical axis.

The map is computed by extracting N small slits which are perpendicular to the requested slit. The small slits have a length equal to the slit width. The same extraction algorithm as in extractRegion-Spectrum is used.

The unit of the PV map flux is X/Sr (where X may be e.g. Jy or K), since one needs to take into account the area of the small slits. The WCS units are velocity along the vertical axis and arcsec since start of slit along the horizontal axis.

The real RA and Declination coordinates of each spectrum in the PV diagram are stored in a separate table dataset 'slitPosition'

API Summary

Properties
SpectralSimpleCube **cube** [INPUT, MANDATORY, default=no default value]
Double **reference** [INPUT, MANDATORY, default=no default value]
Double **startRow** [INPUT, MANDATORY, default=no default value]
Double **startCol** [INPUT, MANDATORY, default=no default value]
Double **endRow** [INPUT, MANDATORY, default=no default value]
Double **endCol** [INPUT, MANDATORY, default=no default value]
Double **slitWidth** [INPUT, OPTIONAL, default=1.0]
Double **aspectRatio** [INPUT, OPTIONAL, default=1.0]
SimpleImage **pvMap** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

SpectralSimpleCube cube [INPUT, MANDATORY, default=no default value]
The input cube

Double reference [INPUT, MANDATORY, default=no default value]

The reference wavelength/frequency/wavenumber, which corresponds with velocity = 0 km/s

Double startRow [INPUT, MANDATORY, default=no default value]

The start row of the slit (in pixel coordinates)

Double startCol [INPUT, MANDATORY, default=no default value]

The start column of the slit (in pixel coordinates)

Double endRow [INPUT, MANDATORY, default=no default value]

The end row of the slit (in pixel coordinates)

Double endCol [INPUT, MANDATORY, default=no default value]

The end column of the slit (in pixel coordinates)

Double slitWidth [INPUT, OPTIONAL, default=1.0]

The width of the slit in pixels

Double aspectRatio [INPUT, OPTIONAL, default=1.0]

Aspect ratio (velocity dimension/ spatial dimension) of PV map

SimpleImage pvMap [OUTPUT, MANDATORY, default=no default value]

The Position-Velocity (PV) map

See also

- Developers Manual: herschel.ia.toolbox.cube.ComputePVMapTask

1.73. computeVelocityMap

Full Name:	herschel.ia.toolbox.cube.ComputeVelocityMapTask
Alias:	computeVelocityMap
Type:	Java Task -
Import:	from herschel.ia.toolbox.cube import ComputeVelocityMapTask
Category	Data cubes/Analysis

Description

VelocityPosMapComputeTask computes several velocity related values from a processed cube.

Computes the velocity, velocity dispersion and line flux maps from a continuum/baseline subtracted spectral cube. The baseline subtraction needs to be done beforehand!

Before applying any algorithm the frequency/wavelength is converted to velocity as follows:

frequency to velocity: v = c*(f0-f)/f0

wavelength to velocity: v = c*(lambda-lambda0)/lambda

Currently two algorithms are offered:

1. Gaussian fit: F(v) = Fmax*exp(-0.5*((v-v0)/w)^2)
   a. line intensity: Area under gaussian profile = **Fmax*w*sqrt(2*pi)**
   b. velocity: The fit parameter **v0**
   c. velocity dispersion: The fit parameter **w**
   d. maximum flux: The fit parameter **Fmax**

   The errors are derived inside the Levenberg-Marquardt algorithm.
2. Moments computation:
   a. line intensity (moment 0): M0 = deltaV * SUM(F_i*w_i,i,Vmin,Vmax)
   b. velocity (moment 1): M1 = SUM(v_i*F_i*w_i,i,Vmin,Vmax)/M0
   c. velocity dispersion (moment 2): M2 = SUM((v_i-M1)^2*F_i*w_i,i,Vmin,Vmax)/M0
   d. maximum flux: Just the maximum flux value in the flux array

   The weights are based on the line strength. See the technical note in the developer manual section of the class ComputeVelocityMapTask for details about how these weights are obtained, as well as on how the errors on the moments are computed. One can fine tune this algorithm with the 'alpha' option (default is 1.0).

Additionally, depending on the algorithm the following diagnostic data is added to the velocityMap product:

1. chiSquared: chi^2 of the gaussian fit.

Furthermore, the values of the 'reference' parameter and 'algorithm' parameter are stored in the meta keywords 'zeroVelocityReference' and 'computeVelocityAlgorithm'.

API Summary

Properties
SpectralSimpleCube **simplecube** [INPUT, MANDATORY, default=no default value]
Double **reference** [INPUT, MANDATORY, default=no default value]
Enum **velAlgorithm** [INPUT, OPTIONAL, default=GAUSSIAN_FIT]
Boolean **isEmission** [INPUT, OPTIONAL, default=true]
Double **alpha** [INPUT, OPTIONAL, default=1.0]
SimpleImage **velocityMap** [OUTPUT, MANDATORY, default=no default value]
SimpleImage **dispersionMap** [OUTPUT, MANDATORY, default=no default value]
SimpleImage **lineIntensityMap** [OUTPUT, MANDATORY, default=no default value]
SimpleImage **maxFluxMap** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

SpectralSimpleCube simplecube [INPUT, MANDATORY, default=no default value]

A baseline/continuum subtracted spectral cube.

Double reference [INPUT, MANDATORY, default=no default value]

The reference wavelength / wavenumber / frequency, which corresponds to v = 0

Enum velAlgorithm [INPUT, OPTIONAL, default=GAUSSIAN_FIT]

Algorithm to use to compute the map, MOMENTS or GAUSSIAN_FIT.

Boolean isEmission [INPUT, OPTIONAL, default=true]

Look for emission (true) or absorption (false) line

Double alpha [INPUT, OPTIONAL, default=1.0]

Expert option to fine tune the line strength weight computation for the moment algorithm

SimpleImage velocityMap [OUTPUT, MANDATORY, default=no default value]

The velocity map

SimpleImage dispersionMap [OUTPUT, MANDATORY, default=no default value]

The velocity dispersion map

SimpleImage lineIntensityMap [OUTPUT, MANDATORY, default=no default value]

The line intensity map

SimpleImage maxFluxMap [OUTPUT, MANDATORY, default=no default value]
A map with the simple line peak values

See also

- Developers Manual: herschel.ia.toolbox.cube.ComputeVelocityMapTask

1.74. CONCATENATE

Full Name:	herschel.ia.numeric.toolbox.basic.Concatenate
Alias:	CONCATENATE
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Concatenate
Category	Arrays and datasets/Manipulation

Description

Concatenates an array of rank greater than one into an array of rank 1 of the same type.

The elements are concatenated starting from the highest dimension.

Example

Example 1: Apply CONCATENATE to an Int1d

```
x=Double2d([ [1,2,3,4],[5,6,7,8] ])
print CONCATENATE(x) # [1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0]
x=Int3d([ [ [1,2],[3,4] ], [ [5,6],[7,8] ], [ [9,10],[11,12] ] ])
print CONCATENATE(x) # [1,2,3,4,5,6,7,8,9,10,11,12]
```

API Summary

Jython Syntax
<y>=CONCATENATE(<x>)

Properties
Array **x** [INPUT, MANDATORY, default=no default value]
Array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Array x [INPUT, MANDATORY, default=no default value]

The input array, of rank greater than one.

Array y [OUTPUT, MANDATORY, default=no default value]

The one-dimensional array resulting from concatenating the input array elements.

See also

- RESHAPE
- Developers Manual: herschel.ia.numeric.toolbox.basic.Concatenate

1.75. Condense

Full Name:	herschel.ia.numeric.toolbox.basic.Condense
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Condense
Category	Arrays and datasets/Reduction

Description

Condenses an array along a given dimension by applying a function to groups of array items.

Each item of the output array is the result of applying a function (such as SUM, COS and so on) to a group of elements of the original array, along the specified dimension. The `chunkSize` argument specifies how many items of the input array are used as input for each item of the output array.

There are two modes of usage, controlled by the `useBoxCar` parameter:

useBoxCar=False (default mode): Condenses the array using groups of elements of specified size.

useBoxCar=True (boxcar mode): Runs a boxcar window of a specified size over the array.

In the default mode, the output array will have _d)) but you can split the original array into chunks. You will obtain an array with the number of chunks that you have specified (the number of items of the returned array is truncated if it is required, see example). Nevertheless, if you use 'boxcar' mode, chunks are created based on the current position for each item of the returned array (see 'mode' parameter and the example).

Example

Example 1: Using the SUM function:

```
x = RESHAPE(Int1d.range(4*2),[4,2])
print x
# [
# [0,1],
# [2,3],
# [4,5],
# [6,7]
# ]
# print Condense(0,1,SUM)(x) # Mode = Default
# java.lang.IllegalArgumentException: Chunk needs to be bigger than 1!
print Condense(0,2,SUM)(x) # Mode = Default
# [
# [2,4],
# [10,12]
# ]
# Get chunks of two items along dimension 0:
# First chunk: SUM(Int2d([[0,1],[2,3]]),0) = [2,4]
# Second chunk: SUM(Int2d([[4,5],[6,7]]),0) = [10,12]
print Condense(0,3,SUM)(x) # Mode = Default
# [
# [6,9]
# ]
# Get chunks of three items along dimension 0:
# First chunk: SUM(Int2d([[0,1],[2,3],[4,5]]),0) = [6,9]
# Second chunk: [6,7] skipped
print Condense(0,4,SUM)(x) # Mode = Default
# [
# [12,16]
# ]
# Get chunks of four items along dimension 0:
```

Example 1: Using the SUM function:

```
# First chunk: SUM(Int2d([[0,1],[2,3],[4,5],[6,7]]),0) = [12,16]
print Condense(0,2,SUM,True)(x)
# [
# [2,4],
# [6,8],
# [10,12],
# [6,7]
# ]
# Get chunks of two items along dimension 0 =&gt; one item after current
 position:
# First chunk: SUM(Int2d([[0,1],[2,3]]),0) = [2,4]
# Second chunk: SUM(Int2d([[2,3],[4,5]]),0) = [6,8]
# Third chunk: SUM(Int2d([[4,5],[6,7]]),0) = [10,12]
# Fourth chunk: SUM(Int2d([[6,7]]),0) = [10,12]
print Condense(0,3,SUM,True)(Int2d([[0,1],[2,3],[4,5],[6,7],[8,9],[10,11],
[12,13]]))
# [
# [2,4],
# [6,9],
# [12,15],
# [18,21],
# [24,27],
# [30,33],
# [22,24]
# ]
# Get chunks of three items along dimension 0 =&gt; one item before current
 position, one item
# after current position:
# First   chunk: SUM (Int2d([[0,1],[2,3]]),0)            = [2,4]
# Second  chunk: SUM (Int2d([[0,1],[2,3],[4,5]]),0)      = [6,9]
# Third   chunk: SUM (Int2d([[2,3],[4,5],[6,7]]),0)      = [12,15]
# Fourth  chunk: SUM (Int2d([[4,5],[6,7],[8,9]]),0)      = [18,21]
# Fifth   chunk: SUM (Int2d([[6,7],[8,9],[10,11]]),0)    = [24,27]
# Sixth   chunk: SUM (Int2d([[8,9],[10,11],[12,13]]),0)  = [30,33]
# Seventh chunk: SUM (Int2d([[10,11],[12,13]]),0)        = [22,24]
print Condense(0,4,SUM,True)(Int2d([[0,1],[2,3],[4,5],[6,7],[8,9],[10,11],
[12,13]]))
# [
# [6,9],
# [12,16],
# [20,24],
# [28,32],
# [36,40],
# [18,20],
# [22,24]
# ]
# Get chunks of four items along dimension 0 =&gt; one item before current
 position, two items
# after current position:
# First   chunk: SUM (Int2d([[0,1],[2,3],[4,5]]),0)          = [6,9]
# Second  chunk: SUM (Int2d([[0,1],[2,3],[4,5],[6,7]]),0)    = [12,16]
# Third   chunk: SUM (Int2d([[2,3],[4,5],[6,7],[8,9]]),0)    = [20,24]
# Fourth  chunk: SUM (Int2d([[4,5],[6,7],[8,9],[10,11]]),0)  = [28,32]
# Fifth   chunk: SUM (Int2d([[6,7],[8,9],[10,11],[12,13]]),0) = [36,40]
# Sixth   chunk: SUM (Int2d([[8,9],[10,11]]),0)              = [18,20]
# Seventh chunk: SUM (Int2d([[10,11],[12,13]]),0)            = [22,24]
```

API Summary

Jython Syntax
`outArray = Condense ( alongDimension, chunkSize, function [,use-BoxCar])(inArray)`

Properties
`Array` **`inArray`** `[INPUT, MANDATORY, default=no default value]`

Properties
Integer **alongDimension** [INPUT, MANDATORY, default=no default value]
Integer **chunkSize** [INPUT, MANDATORY, default=no default value]
ArrayReductor|ArrayToNumber **function** [INPUT, MANDATORY, default=no default value]
Boolean **useBoxCar** [INPUT, OPTIONAL, default=default = false]
Array **outArray** [OUTPUT, MANDATORY, default=no default value]

Limitations

Reducing an array using ArrayToNumber apply(function, dim) can have undesired results. Unlike the array reductors, all results are expressed in:

```
f: Int2d -> Double1d
f: Long4d -> Double3d
f: Double5d -> Double4d
...
```

The following examples show what can go wrong

- Using the Condense function with any ArrayToNumber (MEAN, MEDIAN, STDEV, QRMS) on a non-double input.
- Asking an image to count unique values per row would return a Double1d

API details

Properties

Array inArray [INPUT, MANDATORY, default=no default value]

Input array.

Integer alongDimension [INPUT, MANDATORY, default=no default value]

Dimension along which to apply the function. Starts from zero.

Integer chunkSize [INPUT, MANDATORY, default=no default value]

Number of elements to pass to the function for each output array element. Must be greater than one.

ArrayReductor|ArrayToNumber function [INPUT, MANDATORY, default=no default value]

The function to apply.

Boolean useBoxCar [INPUT, OPTIONAL, default=default = false]

The execution mode.

If `true`', the mode is set to `boxcar`. If `false`', the mode is set to `default`.

- Default: the array is split into arrays of size equal to `chunkSize`, along the dimension specified by `alongDimension`. Any remaining items that cannot fit in a chunk are ignored.

Boolean useBoxCar [INPUT, OPTIONAL, default=default = false]

- Boxcar: chunks are created using a "box" centered on the current item. The process goes item by item along the specified dimension:
  - Odd chunk size: the chunk is made by the item in the current position, (chunkSize/2) item(s) before the current position and (chunkSize/2) item(s) after the current position. The division is an integer one.
  - Even chunk size: the chunk is made by the item in the current position, ((chunkSize/2)-1) item(s) before the current position and (chunkSize/2) item(s) after the current position. The division is an integer one.

Example: for chunkSize = 3, one item before the current position, the current item and one item after the current position. For chunkSize = 4, one item before the current position, the current item and two items after the current position.

Array outArray [OUTPUT, MANDATORY, default=no default value]

The result array.

See also

- Developers Manual: herschel.ia.numeric.toolbox.basic.Condense

1.76. ConjugateGradientFitter

Full Name:	herschel.ia.numeric.toolbox.fit.ConjugateGradientFitter
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import ConjugateGradientFitter
Category	Mathematics/Fitting

Description

Non-linear fitter using the conjugate gradient method.

This class is a wrapper around the ConjugateGradientMethod which integrates it into the Fitter concept.

The ConjugateGradientFitter (CGF) does not use matrix inversions to iterate to the solution (as LevenbergMarquardt). It calculates the gradient to the (local) ChiSq function and proceeds along that direction until it encounters a minimum. From that new position it iterates further.

This behaviour makes it fit for solving problems with many (>10 or so) parameters.

For documentation on the various options see Numerical Recipes Cpt 10.6.

Example

Example 1: ConjugateGradientFitter (for non-linear models)

```
# assume x and y are Double1d data arrays.
x = Double1d.range(100) / 10
y = Double1d.range(100) / 122                    # make slope
rg = RandomGauss( seed=12345L )                    # Gaussian random number
 generator
y += rg( Double1d(100) ) * 0.2                     # add noise
y[Range(9,12)] += Double1d([5,10,7])                  # make some peak
gauss = GaussModel( )                            # Gaussian
gauss += PolynomialModel( 1 )                    # add linear background
print gauss.getNumberOfParameters()                # 5 (= 3 for Gauss + 2 for
 line)
cgfit = ConjugateGradientFitter( x, gauss )
param = cgfit.fit( y )
print param.length()                             # 5
stdev = cgfit.getStandardDeviation()             # stdevs on the parameters
chisq = cgfit.getChiSquared()
scale = cgfit.getScale()                         # noise scale
yfit  = cgfit.getResult()                        # fitted values
yband = cgfit.monteCarloError()                    # 1 sigma confidence region
```

Limitations

1. CGF is **not** guaranteed to find the global minimum.

2. CGF does **not** work with fixed parameters or limits

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.ConjugateGradientFitter

1.77. ConstantModel

Full Name:	herschel.ia.numeric.toolbox.fit.ConstantModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import ConstantModel
Category	Mathematics/Fitting

Description

ConstantModel is an extension of NullModel. It does not have any parameters.

f(x:p) = f(x)

As such it is irrelevant whether it is linear or not. It has 0 params and returns a 0 for its partials.

ConstantModel, by default, returns a constant (= 0) for its result. It can however return any fixed form that an AbstractModel can provide.

This might all seem quite irrelevant for fitting. And indeed no parameters can be fitted to these models, no standard deviations can be calculated, but it is possible to calculate the evidence for these models and compare them with more complicated models to decide whether there is any evidence for some structure at all.

It can also be used when some constant is needed in a compound model.

See example

Example

Example 1: ConstantModel

```
model = ConstantModel()
model.setValue( 1.0 )
model.addModel( ExpModel() )         # make a model that decays to 1.0
# Example 2
model = ConstantModel()
model.setFixedModel( SineModel() )
fixpar = Double1d( [1.0,0.0,5.0] )
model.setFixedParameters( fixpar )    # always returning a cosine of frequency
 5
```

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.ConstantModel

1.78. ContainerModel

Full Name:	herschel.ia.numeric.toolbox.fit.ContainerModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import ContainerModel
Category	Mathematics/Fitting

Description

ContainerModel is a AbstractModel that contains another AbstractModel.

Its results are exactly the same as the results of the contained model.

Warning: ContainerModel is about rather advanced model building.

There are 2 reasons for its existence.

1. When the contained model is a compound model (chain of models), the ContainerModel make a single unit out of it. It acts as a pair of brackets in another chain of models. Since compound models can be joined by operations other than addition (there is also subtract, multiply and divide) brackets are needed to distinguish m1 * (m2 + m3) from m1 * m2 + m3. The ContainerModel provides the brackets.

2. A ContainerModel can (permanently) reduce its number of parameters, i.e. act like it has less parameters than the contained model(s). The missing parameters are permanently fixed. It also has applications in compound models. E.g. when you have a model chain m1 * m2 and both models have an amplitude-like parameter, one of them need to be fixed, otherwise the compound model is degenerate.

See also

- PriorList
- UniformPrior
- NestedSampler
- Developers Manual: herschel.ia.numeric.toolbox.fit.ContainerModel

1.79. ContinuousWavelet

Full Name:	herschel.ia.numeric.toolbox.wavelet.ContinuousWavelet
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.wavelet import ContinuousWavelet

Description

Continuous Wavelet

Performs a continuous wavelet transform using an internal wavelet and an internal algorithm.

Examples

Example 1: Import statements

```
from herschel.ia.numeric.toolbox.wavelet import ContinuousWavelet
from herschel.ia.numeric.toolbox.wavelet.wutil import WBorder
```

Example 2: Short example with one-dimensional signal

```
from herschel.ia.numeric.toolbox.wavelet.wdemo import WSigGenerator
signal1d = WSigGenerator().getPredefinedRealFunction1()
from herschel.ia.numeric.toolbox.wavelet import ContinuousWavelet
algo   = ContinuousWavelet("MexicanHat") # Define the algorithm and wavelet to
 use.
coefs  = algo.decompose(signal1d)        # Decompose our one-dimensional
 signal.
res    = algo.synthesis(coefs)           # rebuild the signal
```

Example 3: One-dimensional signal

```
from herschel.ia.numeric.toolbox.wavelet.wdemo import WSigGenerator
signal1d = WSigGenerator().getPredefinedRealFunction1()
from herschel.ia.numeric.toolbox.wavelet import ContinuousWavelet
algo     = ContinuousWavelet("MexicanHat") # Define the algorithm and wavelet
 to use.
coefs    = algo.decompose(signal1d)        # Decompose our one-dimensional
 signal.
print coefs.depth                  # Show decomposition depth (voice number *
 octave number).
algoDetail = algo.getAlgorithm()   # get details of the algorithm
print algoDetail.getNVoice()       # get number of voices per octave
octave = 2                         # define octave to show
voice  = 10                        # define voice to show
data   = coefs.getNodeForOctaveAndVoice(octave,voice)   # get data found in the
 octave 2 and voice 10
data.multiply(0.25)                # change data - see also (subtract, divide,
 add and reset)
res    = algo.synthesis(coefs)     # rebuild the signal
res.subtract(signal1d)             # compare the result with original signal
print MIN(res),MAX(res)            # print miscellaneous information on the
 difference
print STDDEV(res), MEAN(res)       # print miscellaneous information on the
 difference
plot   = PlotXY(res)               # display the difference
```

Example 4: With border management

```
from herschel.ia.numeric.toolbox.wavelet.wdemo import WSigGenerator
signal1d = WSigGenerator().getPredefinedRealFunction1()
from herschel.ia.numeric.toolbox.wavelet import ContinuousWavelet
```

Example 4: With border management

```
from herschel.ia.numeric.toolbox.wavelet.wutil import WBorder
algo  = ContinuousWavelet("MexicanHat")           # Define the algorithm and
 wavelet to use.
coefs = algo.decompose(signal1d,WBorder.SYMMETRIC) # SYMMETRIC border
 management selected
res   = algo.synthesis(coefs)                     # rebuild the signal
```

Example 5: Wavelet Transform Modulo Maxima Line processing with one-dimensional signal

```
from herschel.ia.numeric.toolbox.wavelet.wdemo import WSigGenerator
signal1d = WSigGenerator().getPredefinedRealFunction1()
from herschel.ia.numeric.toolbox.wavelet import ContinuousWavelet
algo  = ContinuousWavelet("MexicanHat")    # Define the algorithm and wavelet
 to use.
algo.activatesWTMML()                      # activate wtmml algorithm
coefs = algo.decompose(signal1d)           # decompose and build MML map
Display(coefs.getWTMML())                  # display the map
algo.deactivatesWTMML()                    # deactivates WTMML function
```

Example 6: Another example

```
from herschel.ia.numeric.toolbox.wavelet import ContinuousWavelet
algo  = ContinuousWavelet("MexicanHat")        # Define the algorithm and
 wavelet to use.
wavelet = algo.wavelet                        # get the wavelet object
print wavelet.name                             # the name of the wavelet
print wavelet.family                           # the wavelet family
```

Example 7: Yet Another example

```
from herschel.ia.numeric.toolbox.wavelet.wdemo import WSigGenerator
signal1d = WSigGenerator().getPredefinedRealFunction1()
from herschel.ia.numeric.toolbox.wavelet import ContinuousWavelet
algo  = ContinuousWavelet("MexicanHat")        # Define the algorithm and
 wavelet to use.
scale   = 100                                  # scale of the wavelet
coefs = algo.decompose(signal1d)               # decompose and build MML map
wavelet = coefs.wavelet                        # get the wavelet object
wdata   = wavelet.getData(scale)               # wdata contains the wavelet at
 scale 100
PlotXY(wdata.real)                             # show the real part of the
 wavelet at scale 100
PlotXY(wavelet.getDerivativeData(scale).real)  # plot derivative wavelet
 function at scale 100
PlotXY(wavelet.getScalingData(scale).real)     # plot scaling function at scale
 100
```

See also

- Developers Manual: herschel.ia.numeric.toolbox.wavelet.ContinuousWavelet

History

- `2011-01-20 - initial`: version

1.80. Contour

Full Name:	herschel.ia.dataset.image.Contour
Type:	Java Class -
Import:	from herschel.ia.dataset.image import Contour
Category	Images/Display

Description

A class to deal with Contours.

API Summary

Constructor
`Contour (Double2d contourPoints)`
This constructor creates a new Contour with the given contour points.

Methods
***`int`* `getNbOfContourPoints`**
Returns the number of contour points for this Contour.
***`double[]`* `convert (Wcs wcsOld, Wcs wcsNew)`**
Conversion from one Wcs to another.

API Details

Constructor

`Contour (Double2d contourPoints)`

This constructor creates a new Contour with the given contour points.

This constructor creates a new Contour and sets the data (contour points) for the new Contour to the given data (contour points).

Argument

`Double2d` **`contourPoints`** `[INPUT, MANDATORY, default=no default value]`

The data (contour points) to set as the data (contour points) of the new Contour, as Double2d

Error

`IllegalArgumentException`

For each of the given contour points for the new Contour, the x- and y-pixel-coordinate must be given.

Methods

***`int`* `getNbOfContourPoints`**

Returns the number of contour points for this Contour.

Return

***int* getNbOfContourPoints**

int

Returns the number of contour points for this Contour.

***double[]* convert (Wcs wcsOld, Wcs wcsNew)**

Conversion from one Wcs to another.

Converts the pixel coordinates of the contour points of this Contour from one Wcs to pixel coordinates to pixel coordinates in another Wcs.

Arguments

`Wcs` **`wcsOld`** `[INPUT, MANDATORY, default=no default value]`

The old Wcs, as Wcs

`Wcs` **`wcsNew`** `[INPUT, MANDATORY, default=no default value]`

The new Wcs, as Wcs

Return

double[]

The converted contours.

Error

`IllegalArgumentException`

If one of the given Wcs's is invalid.

See also

- Developers Manual: herschel.ia.dataset.image.Contour

1.81. contour

Full Name:	herschel.ia.toolbox.image.ContourTask
Alias:	contour
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import ContourTask
Category	Images/Display

Description

This task generates the contours for one given contour value.

This task generates the contours for one given contour value and is also called internally within automaticContour and manualContour. The output product is an ImageContour product, which holds the generated contours. This product can be dragged over an image explorer for visual inspection. The user can also specify in this task which color should be used for visualization.

Example

Example 1: This is an example of how you can use contour :

```
from java.awt.Color import GREEN
contours = contour(image = myImage, value = 3.7, color = GREEN)
contours = contour(image = myImage, value = 3.7)
```

API Summary

Properties
Image **image** [INPUT, MANDATORY, default=None]
Double **value** [INPUT, MANDATORY, default=None]
Color **color** [INPUT, OPTIONAL, default=BLUE]
ImageContour **contours** [OUTPUT, MANDATORY, default=None]

Miscellaneous

A description of the algorithm can be found in : Regibo, S., Vandenbussche, B. and De Meester, W., 2008, A Java Image Contour Algorithm for Herschel DP, Astronomical Data Analysis Software and Systems (ADASS) XVII

API details

Properties

Image image [INPUT, MANDATORY, default=None]

This is the input image.

Double value [INPUT, MANDATORY, default=None]

This is the contour value.

Color color [INPUT, OPTIONAL, default=BLUE]

This is the color to visualize the contours.

ImageContour contours [OUTPUT, MANDATORY, default=None]
These are the resulting contours.

See also

- Developers Manual: herschel.ia.toolbox.image.ContourTask

1.82. ContourLevel

Full Name:	herschel.ia.dataset.image.ContourLevel
Type:	Java Class -
Import:	from herschel.ia.dataset.image import ContourLevel
Category	Images/Display

Description

A class to deal with ContourLevels.

API Summary

Constructors
ContourLevel (double contourValue) Creation of a new ContourLevel.
ContourLevel (double contourValue, Color color) Creation of a new ContourLevel.

Methods
setColor (Color color) Sets the color for this ContourLevel.
Color **getColor** Returns the color for this ContourLevel.
addContour (Contour contour) Adds the given Contour to this ContourLevel.

API Details

Constructors

ContourLevel (double contourValue)

Creation of a new ContourLevel.

This constructor creates a new ContourLevel for the given contour value.

Argument

double **contourValue** [INPUT, MANDATORY, default=no default value]

The contour value for which a new ContourLevel should be created, as double

ContourLevel (double contourValue, Color color)

Creation of a new ContourLevel.

This constructor creates a new ContourLevel for the given contour value and with the given color.

Arguments

double **contourValue** [INPUT, MANDATORY, default=no default value]

The contour value for which a new ContourLevel should be created, as double

ContourLevel (double contourValue, Color color)

`Color` **color** `[INPUT, MANDATORY, default=no default value]`

The color in which to display the contours, as Color

Methods

setColor (Color color)

Sets the color for this ContourLevel.

Sets the given color as the color for this ContourLevel.

Argument

`Color` **color** `[INPUT, MANDATORY, default=no default value]`

The color to set as the color for this ContourLevel, as Color

Color **getColor**

Returns the color for this ContourLevel.

Return

Color

Returns the color for this ContourLevel.

addContour (Contour contour)

Adds the given Contour to this ContourLevel.

Argument

`Contour` **contour** `[INPUT, MANDATORY, default=no default value]`

The Contour to add to this ContourLevel, as Contour

See also

- Developers Manual: herschel.ia.dataset.image.ContourLevel

1.83. convertAngles

Full Name:	herschel.ia.toolbox.util.ConvertAnglesTask
Alias:	convertAngles
Type:	Java Task -
Import:	from herschel.ia.toolbox.util import ConvertAnglesTask
Category	Arrays and datasets/Manipulation

Description

Converts angles between numeric and textual representations.

It can be used with String1d, Double1d or TableDatasets that hold angle data. This task is provided because AsciiTableReader will not read String formatted angle data directly into numeric format, so those columns need to be post-processed with this task if numeric data is desired. The task will also do the inverse conversion (from numeric data into string representation).

Examples

Example 1: Convert table column to numeric degrees, replacing the data

```
t = TableDataset(description = "table to convert")
t["one"] = Column(String1d(["12h", "6h", "13h00m00.05s"]))
convertAngles(data=t, columnId="one", overwrite=True)
print t["one"]
# {description="", data=[180.0,90.0,195.00020833333335], unit=null}
```

Example 2: Convert table column data to numeric radians

```
t = TableDataset(description = "table to convert")
t["one"] = Column(String1d(["12h", "6h", "13h00m00.05s"]))
converted = convertAngles(data=t, columnId="one", unit="RADIANS")
print converted
# [3.141592653589793,1.5707963267948966,3.403395677491551]
```

Example 3: Convert String1d degrees to numeric hours and back (reformat)

```
t = String1d(["0", "30", "60", "90", "360"])
converted = convertAngles(data=t, format="DEGREES", unit="HOURS")
print converted
# [0.0,1.9999999999999998,3.9999999999999996,6.0,24.0]
back = convertAngles(data=converted, format="DEGREES", unit="HOURS",
 reverse=True)
print back
# ["  0d 0m 0.000s"," 30d 0m 0.000s"," 60d 0m 0.000s"," 90d 0m 0.000s"," 360d
 0m 0.000s"]
```

Example 4: Convert table column from string hours to string degrees (in 2 steps)

```
t = TableDataset(description = "table to convert")
t["one"] = Column(String1d(["12h", "6h", "13h00m00.005s"]))
numeric = convertAngles(data=t, columnId="one", format="HOURS", unit="RADIANS")
print numeric
# [3.141592653589793,1.5707963267948966,3.4033924049992033]
t["one"].data = convertAngles(data=numeric, format="DEGREES", unit="RADIANS",
 reverse=True)
print t["one"]
# {description="", data=[" 180d 0m 0.000s"," 90d 0m 0.000s"," 195d 0m 0.075s"],
 unit=null}
```

API Summary

Jython Syntax
converted= convertAngles(<data> [, <columnId>="0", <format>="DEGREES", <unit>="DEGREES", <overwrite>=False,
<reverse>= False])

Properties
Object **data** [INPUT, MANDATORY, default=no default value]
String **columnId** [INPUT, OPTIONAL, default="0" (first column)]
String **format** [INPUT, OPTIONAL, default="DEGREES"]
String **unit** [INPUT, OPTIONAL, default="DEGREES"]
Boolean **overwrite** [INPUT, OPTIONAL, default=False]
Boolean **reverse** [INPUT, OPTIONAL, default=False]
AbstractArray1dData **converted** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Object data [INPUT, MANDATORY, default=no default value]

A String1d (Double1d for reverse) or TableDataset with angle data to be converted (will not be modified unless overwrite is True)

String columnId [INPUT, OPTIONAL, default="0" (first column)]

The ID of the column to sort the table by. If it can be interpreted as a valid index it will, otherwise it will be a column name. Ignored unless data is a TableDataset

String format [INPUT, OPTIONAL, default="DEGREES"]

String with the default units of the string representation ("DEGREES" (default), "HOURS") (only used if the data has no units markers). With reverse= True, "DEGREES" is XXd YYm ZZ.Zs and "HOURS" is XXh YYm ZZ.Zs

String unit [INPUT, OPTIONAL, default="DEGREES"]

The unit for the angles in numeric representation ("DEGREES" (default), "HOURS", "RADIANS")

Boolean overwrite [INPUT, OPTIONAL, default=False]

If true, it will modify data (the passed table) Ignored unless data is a TableDataset

Boolean reverse [INPUT, OPTIONAL, default=False]

If true convert from numbers to strings (default is strings to numbers)

AbstractArray1dData converted [OUTPUT, MANDATORY, default=no default value]

The column data in numerical format (string format if reverse)

See also

- asciiTableReader
- asciiTableWriter
- Developers Manual: herschel.ia.toolbox.util.ConvertAnglesTask

History

- `2012-12-20 - JDS`: Initial prototype
- `2013-06-26 - JDS`: Release

1.84. convertImageUnit

Full Name:	herschel.ia.toolbox.image.ConvertImageUnitTask
Alias:	convertImageUnit
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import ConvertImageUnitTask
Category	Images/Analysis

Description

This is a task to convert the unit of an image from one surface brightness unit to another one.

This is a task to convert the unit of an image from one surface brightness unit to another one. You must give the new surface brightness unit as input, and in some cases also the beam area is required (f.i. for conversion from Jy/beam to Jy/pixel).

Example

Example 1: This is an example of how to use the convertImageUnit :

```
myBeamArea = (Math.PI * (18.1**2))/(4 * LOG(2))
converted = convertImageUnit(image = myImage, newUnit = "Jy/pixel", beamArea =
 myBeamArea)
```

API Summary

Properties
Image **image** [INPUT, MANDATORY, default=None]
Unit **newUnit** [INPUT, MANDATORY, default=None]
float **beamArea** [INPUT, OPTIONAL, default=0.0]
Image **convertedImage** [OUTPUT, MANDATORY, default=None]

API details

Properties

Image image [INPUT, MANDATORY, default=None]

This is the input image.

Unit newUnit [INPUT, MANDATORY, default=None]

This is the surface brightness unit to convert the image to.

float beamArea [INPUT, OPTIONAL, default=0.0]

This is is the beam area in square arcsec, which is required for some unit conversion (f.i. from Jy/beam to Jy/pixel).

Image convertedImage [OUTPUT, MANDATORY, default=None]

This is the resulting converted image.

See also

- Developers Manual: herschel.ia.toolbox.image.ConvertImageUnitTask

1.85. convertUnits

Full Name:	herschel.ia.toolbox.util.ConvertUnitsTask
Alias:	convertUnits
Type:	Java Task -
Import:	from herschel.ia.toolbox.util import ConvertUnitsTask
Category	Arrays and datasets/Manipulation

Description

Converts numeric data expressed in one unit, in terms of another unit.

Example

Example 1: Convert gigahertz to megahertz

```
dataGHz = Double1d.range(3)                                                  #
 [1.0, 2.0, 3.0]
dataMHz = convertUnits(data=dataGHz, sourceUnit="GHz", targetUnit="MHz") #
 [1000.0, 2000.0, 3000.0]
```

API Summary

Jython Syntax

```
convertedData = convertUnits(&lt;data&gt;, &lt;sourceUnit&gt;,
&lt;targetUnit&gt;
[, &lt;quantity&gt;, &lt;referenceUnit&gt;, &lt;reference&gt;,
&lt;overwrite&gt;=False, &lt;verbose&gt;=False])
```

Properties

Object **data** [INPUT, MANDATORY, default=no default value]
String **sourceUnit** [INPUT, OPTIONAL, default=no default value]
String **targetUnit** [INPUT, MANDATORY, default=no default value]
String **quantity** [INPUT, OPTIONAL, default=no default value]
String **referenceUnit** [INPUT, OPTIONAL, default=no default value]
Object **reference** [INPUT, OPTIONAL, default=no default value]
Boolean **overwrite** [INPUT, OPTIONAL, default=False]
Object **convertedData** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Object data [INPUT, MANDATORY, default=no default value]

The data to convert: either a Double, a DoubleArray, a DoubleParameter or a DataWrapper holding a DoubleArray. If it is not a Double and overwrite is True, the data will be modified

String sourceUnit [INPUT, OPTIONAL, default=no default value]

The name of the unit in which the input data is expressed; must be omitted if the data provides the source unit

String targetUnit [INPUT, MANDATORY, default=no default value]

The name of the unit in which the input data will be expressed

String quantity [INPUT, OPTIONAL, default=no default value]

The name of the unit in which the input data will be expressed

String referenceUnit [INPUT, OPTIONAL, default=no default value]

The unit in which the reference value will be expressed; required when converting between incompatible flux density units

Object reference [INPUT, OPTIONAL, default=no default value]

The reference value(s) to be used for converting between incompatible flux density units. It must match the type and dimensions of the data parameter.

Boolean overwrite [INPUT, OPTIONAL, default=False]

If True, the input data will be modified; if False, a copy array will be used for the conversion. It is ignored if the input data is a Double

Object convertedData [OUTPUT, MANDATORY, default=no default value]

The data in numerical format, either a Double or a DoubleXd (depending on the input)

See also

- Developers Manual: herschel.ia.toolbox.util.ConvertUnitsTask

History

- `2013-03-04 - JSS`: Initial prototype.
- `2013-09-18 - JSS`: Require reference to be an array if the data is an array.

1.86. convertWavescale

Full Name:	herschel.ia.toolbox.spectrum.ConvertWavescaleTask
Alias:	convertWavescale
Type:	Java Task -
Import:	from herschel.ia.toolbox.spectrum import ConvertWavescaleTask
Category	Spectra/Analysis

Description

Task for transforming the wavescale between frequency (w), velocity (v), wavelength (l) and wavenumber (k).

On one hand, the numerical values are adjusted according to the rules

- `k = w / c = 1 / l`
- `w = w0 * (1 - v/c); k = k0 * (1 - v/c); l = l0 / (1 - v/c)`

where `c` is the speed of light. Furthermore, some meta data is updated:

- wave unit
- wave description
- wave name (where applicable - not all data structures may have this property)
- reference frequency / wavelength or wavenumber used for the transformation to or from the velocity scale.

The wavename mentioned above is changed to:

- "frequency" when transforming to a frequency scale
- "velocity" when transforming to a velocity scale
- "wavelength" when transforming to a wavelength scale
- "wavenumber" when transforming to a wavenumber scale

When transforming to a velocity scale do not forget to specify a reference.

Example

Example 1: global spectra # defined elsewhere

```
# assume data with wave scale in units MHz - prepare a new spectrum data object
 with transformed unit and
# keep the input unchanged:
result = convertWavescale(ds=spectra, to="1/s", overwrite=False)
result = convertWavescale(ds=spectra, to="Hz", overwrite=False)
result = convertWavescale(ds=spectra, to="m-1", overwrite=False)
result = convertWavescale(ds=spectra, to="micrometer", overwrite=False)
result = convertWavescale(ds=spectra, to="mm", overwrite=False)
result = convertWavescale(ds=spectra, to="km/s", reference=1153.0,
 overwrite=False)
result = convertWavescale(ds=spectra, to="km s-1", reference=1153.0,
 overwrite=False)
# without setting the overwriting parameter (and setting it to false) the
 wavescale unit is tranformed
```

Example 1: global spectra # defined elsewhere

```
# directly for the input data:
convertWavescale(ds=spectra, to="GHz")
convertWavescale(ds=spectra, to="MHz")
convertWavescale(ds=spectra, to="km s-1", reference=1153.0,
 referenceUnit="GHz")
convertWavescale(ds=spectra, to="MHz")
convertWavescale(ds=spectra, to="1/mm")
convertWavescale(ds=spectra, to="m s-1", reference=3.8465)
convertWavescale(ds=spectra, to="1/m")
convertWavescale(ds=spectra, to="mm")
convertWavescale(ds=spectra, to="m s-1", reference=260.0,
 referenceUnit="micrometer")
convertWavescale(ds=spectra, to="micrometer")
convertWavescale(ds=spectra, to="GHz")
```

API Summary

Properties
SpectrumContainer **ds** [INPUT, MANDATORY, default=no default value.]
String **to** [INPUT, MANDATORY, default=no default value.]
Double **reference** [INPUT, OPTIONAL, default=no default value.]
String **referenceUnit** [INPUT, OPTIONAL, default=no default value.]
Boolean **overwrite** [INPUT, OPTIONAL, default=False.]
SpectrumContainer **result** [OUTPUT, OPTIONAL, default=no default value]

API details

Properties

SpectrumContainer ds [INPUT, MANDATORY, default=no default value.]

Spectra to be converted to a velocity scale. Examples of `SpectrumContainer` are `Spectrum1d, Spectrum2d, SpectralSimpleCube`.

String to [INPUT, MANDATORY, default=no default value.]

Unit the given wave scale should be transformed to. Possible units are

- when transforming to frequency: "Hz" (or "s-1"), "kHz", "MHz", "GHz", "THz"
- when transforming to wavelength: "angstrom", "micrometer", "mm" (millimeter), "cm" (centimeter), "m", "km", "ua" (astronomical units)
- when transforming to wavenumber: "m-1" (or "1/m"), "cm-1" (or "1/cm"), "mm-1" (or "1/mm")
- when transforming to velocity: "m s-1" (or "m/s"), "km s-1" (or "km/s"), "km h-1"(or "km/h")

Double reference [INPUT, OPTIONAL, default=no default value.]

Reference frequency, wavelength or wavenumber used only when transforming to the velocity scale. It defines where the velocity scale is centered at. Note that the reference unit needs to be expressed as a frequency, wavelength or wavenumber if the spectra are expressed at a fre-

Double reference [INPUT, OPTIONAL, default=no default value.]

quency, wavelength or wavenumber wavescale, respectively. Specify a reference unit if the reference is expressed in a different unit that is consistent with the original wave scale. Example: If the original wavescale is in MHz but the reference is specified in GHz, set the referenceUnit to 'GHz'. But, it is not possible to have the original wavescale in MHz and specify the reference in a wavelength unit.

String referenceUnit [INPUT, OPTIONAL, default=no default value.]

Unit of the reference frequency or reference wavescale. The same units are applicable as listed with the parameter "to" (except for the velocity units that do not apply here). If no reference unit is specified the reference is assumed to be in the same unit as the wave scale the spectra are at.

Boolean overwrite [INPUT, OPTIONAL, default=False.]

Specify whether the input data container is reused - if true the values found therein are overwritten if false a new container is returned and the original spectrum data are not modified.

SpectrumContainer result [OUTPUT, OPTIONAL, default=no default value]

Result object containing the results of the operation applied.

See also

- SpectrumContainer
- Developers Manual: herschel.ia.toolbox.spectrum.ConvertWavescaleTask

History

- `2011-08-08 - melchior`: renamed from ConvertWavescale

1.87. Convolution

Full Name:	herschel.ia.numeric.toolbox.filter.Convolution
Alias:	Convolution
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.filter import Convolution
Category	Mathematics/Signal processing

Description

This class implements the convolution of a one- or two-dimensional array with a one-dimensional convolution kernel.

You can choose the behaviour at the edges with the *edge* parameter :

- edge = Convolution.ZEROES = 0 : sets the input array values to zero beyond the edges (default behavior)
- edge = Convolution.CIRCULAR = 1 : wraps around the edges (circular convolution) and continues at the other side of the input array
- edge = Convolution.REPEAT = 2 : repeats the values beyond the edges of the input array
- edge = Convolution.CUTOFF = 3 : Cut off the parts of the filter that go beyond the input array

With the *center* parameter, you can also indicate whether you want to align the convolution kernel with the input array. Set it to true (which is the default value) if you want alignment, or to false when you don't want alignment.

Example

Example 1: Apply convolution filter on an Int1d x with a kernel k:

```
x = Int1d([1,3,2,4,6,5])
k = Double1d([1,3,2]) #The kernel has odd number of elements
f = Convolution(k) # center=true and edge = 0 (default)
print f(x) # [6.0,13.0,16.0,22.0,31.0,27.0]
f = Convolution(k,center=0) # center=false, edge = 0 (default)
print f(x) # [1.0,6.0,13.0,16.0,22.0,31.0]
f=Convolution(k,edge=1) # center is true and edge is circular
print f(x) # [16.0,13.0,16.0,22.0,31.0,28.0]
f=Convolution(k, edge=2) # center=true, edge is repeat
print f(x) # [8.0,13.0,16.0,22.0,31.0,32.0]
f=Convolution(k, edge=3) # center=true, edge is cutoff
print f(x) # [9.0,13.0,16.0,22.0,31.0,32.4]
```

API Summary

Constructors
Convolution (Double1d kernel) Construction of a new Convolution.
Convolution (Double1d kernel) Construction of a new Convolution.
Convolution (Double1d kernel, boolean center, Convolution.ZEROES|CIRCULAR|REPEAT|CUTOFF edge)

Constructors
Construction of a new Convolution.
Convolution (double[] kernel, boolean center, Convolution.ZEROES|CIRCULAR|REPEAT|CUTOFF edge)
Construction of a new Convolution.

Methods
Double1d **getKernel**
Returns the convolution kernel.
boolean **getCenter**
Returns the alignment behaviour: center
int **getEdge**
Returns the behaviour at the edges.
setCenter (boolean center)
Sets the alignment behaviour parameter: center
setEdge (Convolution.ZEROES|CIRCULAR|REPEAT|CUTOFF edge)
Setting the behaviour at the edges for this Convolution.
Double1d **mutate** (Double1d array)
Applies the convolution to a one-dimensional array
Double2d **mutate** (Double2d array)
Applies the convolution to a two-dimensional array

API Details

Constructors

Convolution (Double1d kernel)

Construction of a new Convolution.

A new Convolution is constructed with the given convolution kernel, and default values for the other parameters :

- center = true : convolution kernel not aligned with the input array
- edge = Convolution.ZEROES : set the input array values to zero beyond the edges

Argument

Double1d **kernel** [INPUT, MANDATORY, default=None]

The convolution kernel in the form of a Double1d. When the *center* parameter is set to true, this kernel should have an off number of elements.

Convolution (Double1d kernel)

Construction of a new Convolution.

A new Convolution is constructed with the given convolution kernel, and default values for the other parameters :

- center = true : convolution kernel not aligned with the input array
- edge = Convolution.ZEROES : set the input array values to zero beyond the edges

Argument

Convolution (Double1d kernel)

`Double1d` **`kernel`** `[INPUT, MANDATORY, default=None]`

The convolution kernel in the form of a Double1d. When the *center* parameter is set to true, this kernel should have an off number of elements.

Convolution (Double1d kernel, boolean center, Convolution.ZEROES|CIRCULAR|REPEAT|CUTOFF edge)

Construction of a new Convolution.

A new Convolution is constructed with the given convolution kernel. Also the behaviour at the edges and whether or not to align the convolution kernel with the input array are given.

Arguments

`Double1d` **`kernel`** `[INPUT, MANDATORY, default=None]`

The convolution kernel in the form of a Double1d. When the *center* parameter is set to true, this kernel should have an off number of elements.

`boolean` **`center`** `[INPUT, MANDATORY, default=true]`

Whether or not to align the convolution kernel with the input array.

`Convolution.ZEROES|CIRCULAR|REPEAT|CUTOFF` **`edge`** `[INPUT, MANDATORY, default=Convolution.ZEROES]`

Specifies the behaviour at the edges :

- edge = Convolution.ZEROES = 0 : sets the input array values to zero beyond the edges (default behavior)
- edge = Convolution.CIRCULAR = 1 : wraps around the edges (circular convolution) and continues at the other side of the input array
- edge = Convolution.REPEAT = 2 : repeat the values beyond the edges of the input array
- edge = Convolution.CUTOFF = 3 : Cut off the parts of the filter that go beyond the input array

Convolution (double[] kernel, boolean center, Convolution.ZEROES|CIRCULAR|REPEAT|CUTOFF edge)

Construction of a new Convolution.

A new Convolution is constructed with the given convolution kernel. Also the behaviour at the edges and whether or not to align the convolution kernel with the input array are given.

Arguments

`double[]` **`kernel`** `[INPUT, MANDATORY, default=None]`

The convolution kernel in the form of a double[]. When the *center* parameter is set to true, this kernel should have an off number of elements.

`boolean` **`center`** `[INPUT, MANDATORY, default=true]`

Whether or not to align the convolution kernel with the input array.

`Convolution.ZEROES|CIRCULAR|REPEAT|CUTOFF` **`edge`** `[INPUT, MANDATORY, default=Convolution.ZEROES]`

Specifies the behaviour at the edges :

- edge = Convolution.ZEROES = 0 : sets the input array values to zero beyond the edges (default behavior)
- edge = Convolution.CIRCULAR = 1 : wraps around the edges (circular convolution) and continues at the other side of the input array

Convolution (double[] kernel, boolean center, Convolution.ZEROES|CIRCULAR|REPEAT|CUTOFF edge)

- edge = Convolution.REPEAT = 2 : repeat the values beyond the edges of the input array
- edge = Convolution.CUTOFF = 3 : Cut off the parts of the filter that go beyond the input array

Methods

***Double1d* getKernel**

Returns the convolution kernel.

Returns the convolution kernel for this Convolution.

Return

Double1d

The convolution kernel for this Convolution.

***boolean* getCenter**

Returns the alignment behaviour: center

Returns the alignment behaviour of the convolution kernel with the input data for this Convolution :

- center = true : the convolution kernel will be aligned with the input array
- center = false : the convolution kernel will not be aligned with the input array

Return

boolean

The alignment behaviour of the convolution kernel with the input array for this Convolution.

***int* getEdge**

Returns the behaviour at the edges.

Returns the behaviour at the edges for this Convolution :

- edge = Convolution.ZEROES = 0 : sets the input array values to zero beyond the edges (default behavior)
- edge = Convolution.CIRCULAR = 1 : wraps around the edges (circular convolution) and continues at the other side of the input array
- edge = Convolution.REPEAT = 2 : repeats the values beyond the edges of the input array
- edge = Convolution.CUTOFF = 3 : Cut off the parts of the filter that go beyond the input array

Return

int

The behaviour at the edges for this Convolution.

setCenter (boolean center)

Sets the alignment behaviour parameter: center

Sets the alignment behaviour of the convolution kernel with the input array for this Convolution to the given behaviour :

setCenter (boolean center)

- center = true : the convolution kernel will be aligned with the input array
- center = false : the convolution kernel will not be aligned with the input array

Argument

`boolean` **`center`** `[INPUT, MANDATORY, default=true]`

Whether or not to align the convolution kernel with the input array.

setEdge (Convolution.ZEROES|CIRCULAR|REPEAT|CUTOFF edge)

Setting the behaviour at the edges for this Convolution.

Sets the behaviour at the edges for this Convolution to the given behaviour :

- edge = Convolution.ZEROES = 0 : sets the input array values to zero beyond the edges (default behavior)
- edge = Convolution.CIRCULAR = 1 : wraps around the edges (circular convolution) and continues at the other side of the input array
- edge = Convolution.REPEAT = 2 : repeats the values beyond the edges of the input array
- edge = Convolution.CUTOFF = 3 : Cut off the parts of the filter that go beyond the input array

Argument

`Convolution.ZEROES|CIRCULAR|REPEAT|CUTOFF` **`edge`** `[INPUT, MANDATORY, default=Convolution.ZEROES]`

Specifies the behaviour at the edges :

- edge = Convolution.ZEROES = 0 : sets the input array values to zero beyond the edges (default behavior)
- edge = Convolution.CIRCULAR = 1 : wraps around the edges (circular convolution) and continues at the other side of the input array
- edge = Convolution.REPEAT = 2 : repeat the values beyond the edges of the input array
- edge = Convolution.CUTOFF = 3 : Cut off the parts of the filter that go beyond the input array

Double1d **mutate (Double1d array)**

Applies the convolution to a one-dimensional array

Applies the convolution to a one-dimensional array with the parameters held by this Convolution.

Argument

`Double1d` **`array`** `[INPUT, MANDATORY, default=no default value]`

The array to which to apply the convolution, as Double1d

Return

Double1d

The result of the convolution (a one-dimensional array).

Double2d **mutate (Double2d array)**

Applies the convolution to a two-dimensional array

Applies the convolution to a two-dimensional array with the parameters held by this Convolution.

***Double2d* mutate (Double2d array)**

Argument

`Double2d` **`array`** `[INPUT, MANDATORY, default=no default value]`

The array to which to apply the convolution, as Double2d

Return

Double2d

The result of the convolution (a two-dimensional array).

See also

- BoxCarFilter
- GaussianFilter
- Developers Manual: herschel.ia.numeric.toolbox.filter.Convolution

History

- `2008-04-23 - AS`: Modify default for center parameter; default is center=True.
- `2010-12-14 - AS`: Add URM category; correct URM example.
- `2011-09-19 - New`: algorithm for proper convolution calculation (HCSS 14114)

1.88. CoordsTranslator

Full Name:	herschel.ia.dataset.image.wcs.CoordsTranslator
Type:	Java Class -
Import:	from herschel.ia.dataset.image.wcs import CoordsTranslator

Description

This is a class that is used to convert sky coordinates in one coordinate system to sky coordinates in another coordinate grid.

API Summary

Constructors
CoordsTranslator (Wcs from, Wcs to) Constructs a new CoordsTranslator, that can be used to convert sky coordinates one one
CoordsTranslator (Wcs.Type from, Wcs.Type to) Constructs a new CoordsTranslator, that can be used to convert sky coordinates one one

Methods
double[] **translate** (double[] before) Translation of sky coordinates.
Quaternion **getQuaternion** Returns the quaternion for this CoordsTranslator, that is used for coordinate conversions between equatorial, galactic, and ecliptic coordinate systems.
boolean **isSame** Checks whether the original and the new coordinate system for this CoordsTranslator are the same.

API Details

Constructors

CoordsTranslator (Wcs from, Wcs to)

Constructs a new CoordsTranslator, that can be used to convert sky coordinates one one

coordinate system, to sky coordinates in another coordinate system.

Arguments

`Wcs` **`from`** `[INPUT, MANDATORY, default=no default value]`

The original coordinate system, as Wcs

`Wcs` **`to`** `[INPUT, MANDATORY, default=no default value]`

The new coordinate system, as Wcs

CoordsTranslator (Wcs.Type from, Wcs.Type to)

Constructs a new CoordsTranslator, that can be used to convert sky coordinates one one

coordinate system, to sky coordinates in another coordinate system.

`CoordsTranslator (Wcs.Type from, Wcs.Type to)`

Arguments

`Wcs.Type` **`from`** `[INPUT, MANDATORY, default=no default value]`

The original coordinate system, as Wcs.Type

`Wcs.Type` **`to`** `[INPUT, MANDATORY, default=no default value]`

The new coordinate system, as Wcs.Type

Methods

`double[]` **`translate (double[] before)`**

Translation of sky coordinates.

Translates the given sky coordinates in the original coordinate system for this CoordsTranslator to sky coordinates in the new coordinate system for this CoordsTranslator.

Argument

`double[]` **`before`** `[INPUT, MANDATORY, default=no default value]`

The sky coordinates in the original coordinate system for this CoordsTranslator, as double[]

Return

double[]

The given sky coordinates translated to the new coordinate system for this CoordsTranslator.

`Quaternion` **`getQuaternion`**

Returns the quaternion for this CoordsTranslator, that is used for coordinate conversions between equatorial, galactic, and ecliptic coordinate systems.

Return

Quaternion

The quaternion for this CoordsTranslator, that is used for coordinate conversions between equatorial, galactic, and ecliptic coordinate systems.

`boolean` **`isSame`**

Checks whether the original and the new coordinate system for this CoordsTranslator are the same.

Return

boolean

True if the original and the new coordinate system for this CoordsTranslator are the same; false otherwise.

See also

- Developers Manual: herschel.ia.dataset.image.wcs.CoordsTranslator

1.89. Correlate

Full Name:	herschel.ia.numeric.toolbox.basic.Correlate
Alias:	Correlate
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Correlate
Category	Mathematics/Statistics

Description

Returns the linear Pearson correlation coefficient of the input arrays.

If one array is longer than the other, only the values up to the length of the shortest array are taken into account.

Example

Example 1: Correlation of two Long1d arrays

```
x = Long1d([65, 63, 67, 64, 68, 62, 70, 66, 68, 67, 69, 71])
y = Long1d([68, 66, 68, 65, 69, 66, 68, 65, 71, 67, 68, 70])
print Correlate(x)(y) # 0.7026516450800809
```

API Summary

Jython Syntax

```
&lt;r&gt;=Correlate(&lt;x&gt;)(&lt;y&gt;)
```

Properties

1-D array **x** [INPUT, MANDATORY, default=no default value]

1-D array **y** [INPUT, MANDATORY, default=no default value]

Double **r** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

1-D array x [INPUT, MANDATORY, default=no default value]

First array to be correlated. Integer arrays are implicitly transformed to double arrays.

1-D array y [INPUT, MANDATORY, default=no default value]

Second array to be correlated. Integer arrays are implicitly transformed to double arrays.

Double r [OUTPUT, MANDATORY, default=no default value]

The linear Pearson correlation coefficient of the input arrays.

See also

- CorrelateMatrix

- Developers Manual: herschel.ia.numeric.toolbox.basic.Correlate

1.90. CorrelateMatrix

Full Name:	herschel.ia.numeric.toolbox.basic.CorrelateMatrix
Alias:	CorrelateMatrix
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import CorrelateMatrix
Category	Mathematics/Statistics

Description

Returns the linear Pearson correlation coefficients of the input *m x n* matrix.

The result is an *n x n* matrix with each *(i, j)* value equal to the correlation coefficient of the *i* and *j* columns of the original matrix.

Example

Example 1: Correlation of a matrix of integers

```
m = Int2d([ [65, 67], [64, 68], [67, 71] ])
print CorrelateMatrix()(m)
# [ [1.0, 0.8386278693775344], [0.8386278693775344, 1.0] ]
```

API Summary

Jython Syntax

```
&lt;r&gt;=CorrelateMatrix()(&lt;m&gt;)
```

Properties

2-D array (m rows x n columns) m [INPUT, MANDATORY, default=no default value]

Double2d r [OUTPUT, MANDATORY, default=no default value]

API details

Properties

2-D array (m rows x n columns) m [INPUT, MANDATORY, default=no default value]

The matrix to be correlated, with *m* rows and *n* columns. Integer values are implicitly transformed to double values.

Double2d r [OUTPUT, MANDATORY, default=no default value]

The *n x n* matrix with the linear Pearson correlation coefficients of the input matrix.

See also

- Correlate
- Developers Manual: herschel.ia.numeric.toolbox.basic.CorrelateMatrix

1.91. COS

Full Name:	herschel.ia.numeric.toolbox.basic.Cos
Alias:	COS
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Cos
Category	Mathematics/Trigonometry

Description

Returns the cosine of a number or array.

If the input is an array, the output is an array of the same type and size, with cosine values instead of the original elements. For complex numbers, the cosine is computed for the real and imaginary part.

Example

Example 1: Applying COS to a Float1d

```
x = Float1d([0,0.5])
print  COS(x) # [1.0,0.87758255]
```

API Summary

Jython Syntax
<y> = COS(<x>)

Properties
Number or array **x** [INPUT, MANDATORY, default=no default value]
Number or array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Number or array x [INPUT, MANDATORY, default=no default value]

The angle or angles of which to compute the cosine, in radians.

Number or array y [OUTPUT, MANDATORY, default=no default value]

The cosine value or values.

See also

- ARCCOS
- COSH
- SIN
- TAN

- Developers Manual: herschel.ia.numeric.toolbox.basic.Cos

1.92. COSH

Full Name:	herschel.ia.numeric.toolbox.basic.CosH
Alias:	COSH
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import CosH
Category	Mathematics/Trigonometry

Description

Returns the hyperbolic cosine of a number or array.

If the input is an array, the output is an array of the same type and size, with hyperbolic cosine values instead of the original elements. For complex numbers, the hyperbolic cosine is computed for the real and imaginary part.

API Summary

Jython Syntax
`<y> = COSH(<x>)`

Properties
`Number or array x [INPUT, MANDATORY, default=no default value]`
`Number or array y [OUTPUT, MANDATORY, default=no default value]`

API details

Properties

`Number or array x [INPUT, MANDATORY, default=no default value]`
The angle or angles of which to compute the hyperbolic cosine, in radians.

`Number or array y [OUTPUT, MANDATORY, default=no default value]`
The hyperbolic cosine value or values.

See also

- COS
- ARCCOS
- Developers Manual: herschel.ia.numeric.toolbox.basic.CosH

1.93. Covariance

Full Name:	herschel.ia.numeric.toolbox.stat.Covariance
Alias:	Covariance
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.stat import Covariance
Category	Mathematics/Statistics

Description

Yields the covariance between two random variables/vectors x and y with finite second moments

If one vector is longer than the other, only the values up to the length of the shorter vector will be taken into account.

Example

Example 1: Covariance of two vectors of long

```
x = Long1d([65, 63, 67, 64, 68, 62, 70, 66, 68, 67, 69, 71])
y = Long1d([68, 66, 68, 65, 69, 66, 68, 65, 71, 67, 68, 70])
print Covariance(x)(y) # 3.361111111111112
```

API Summary

Jython Syntax
<r>=Covariance(<x>)(<y>)

Properties
any ordered 1d array **x** [INPUT, MANDATORY, default=no default value]
any ordered 1d array **y** [INPUT, MANDATORY, default=no default value]
double **r** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

any ordered 1d array x [INPUT, MANDATORY, default=no default value]

May be an integral or floating-point array; the former implicitly transformed to a double array.

any ordered 1d array y [INPUT, MANDATORY, default=no default value]

May be an integral or floating-point array; the former implicitly transformed to a double array.

double r [OUTPUT, MANDATORY, default=no default value]

Returns the covariance between the input arrays with finite second moments..

See also

- CovarianceMatrix
- Developers Manual: herschel.ia.numeric.toolbox.stat.Covariance

1.94. CovarianceMatrix

Full Name:	herschel.ia.numeric.toolbox.stat.CovarianceMatrix
Alias:	CovarianceMatrix
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.stat import CovarianceMatrix
Category	Mathematics/Statistics

Description

Yields the covariance matrix of the input M x N matrix

The result is a N x N matrix with each i, j value equal to the covariance of the ith and jth columns of the original matrix.

Example

Example 1: Apply Covariance of a matrix of integers

```
m = Int2d([ [65, 67], [64, 68], [67, 71] ])
print CovarianceMatrix()(m)
# [[1.5555555555555556,1.777777777777777],
[1.777777777777777,2.8888888888888893]]
```

API Summary

Jython Syntax
<r>=CovarianceMatrix()(<m>)

Properties
any ordered 2d array (M rows x N columns) m [INPUT, MANDATORY, default=no default value]
Double2d r [OUTPUT, MANDATORY, default=no default value]

API details

Properties

any ordered 2d array (M rows x N columns) m [INPUT, MANDATORY, default=no default value]

May be an integral or floating-point array; the former implicitly transformed to a double array.

Double2d r [OUTPUT, MANDATORY, default=no default value]

Returns the N x N matrix with the covariances of the input matrix.

See also

- Covariance
- Developers Manual: herschel.ia.numeric.toolbox.stat.CovarianceMatrix

1.95. createRgbImage

Full Name:	herschel.ia.toolbox.image.CreateRgbImageTask
Alias:	createRgbImage
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import CreateRgbImageTask
Category	Images/Analysis

Description

This is a task to combine three images to an RGB-image.

These three images should cover more or less the same area on the sky, but in different filters. If they have a different Wcs, or they have the same Wcs but different dimensions, they will be regridded onto the smallest grid before they are combined to an RGB-image. Before combining them, their values are rescaled, such that these are integers between 0 and 255. If not all three input image have a valid Wcs, they can only be combined if they have the same dimensions. If they all have the same Wcs and the same dimensions, this Wcs is also used for the resulting RGB image. Otherwise they are regridded onto the spatial grid of the image with the least number of pixels. If a Wcs was given as input, that will be used, if valid. How is the rescaling done if you have given the cut levels for all input images yourself? For an image for which you have given the cut levels "lowCut" and "highCut", each intensity value is replaced by (value - lowCut) / (highCut - lowCut) * 255 The values in the scaled image then range between 0 and 255. Possible regridding onto a new output grid is then done in a non-flux-conservative way (in order to guarantee that the values of the regridded scaled image are still between 0 and 255). How is the rescaling done if you have not given the cut levels yourself, by a percentage and a scaling factor? For each image, the cut levels are calculated, such that the given percentage of values is included in this range. We then multiply the intensity values in the image with the given scaling factor. We then apply the procedure as described above for the multiplied image and the calculated cut levels.

Examples

Example 1: This is an example of how you can use the createRgbImage by specifying the cut levels yourself :

```
rgb = createRgbImage(red = myRedImage, green = myGreenImage, blue =
 myBlueImage, lowBlue = 0.0,
                      highBlue = 50.0, lowGreen = 60.0, highGreen = 120.0,
 lowBlue = 12.0,
                      highGreen = 160.0)
```

Example 2: This is an example of how you can use the createRbImage by specifying the percentage and scaling

```
factors :
rgb = createRgbImage(red = myRedImage, green = myGreenImage, blue =
 myBlueImage,
                      percentage = 98.0, redFactor = 1.3, greenFactor = 1.0,
 blueFactor = 1.6)
rgb = createRgbImage(red = myRedImage, green = myGreenImage, blue =
 myBlueImage,
                      percentage = 98.0, redFactor = 1.3, greenFactor = 1.0,
 blueFactor = 1.6,
                      wcs = myWcs)
```

API Summary

Properties
Image **red** [INPUT, MANDATORY, default=None]
Image **green** [INPUT, MANDATORY, default=None]
Image **blue** [INPUT, MANDATORY, default=None]
Wcs **wcs** [INPUT, OPTIONAL, default=None]
Double **percent** [INPUT, MANDATORY, default=98.0]
Double **lowRed** [INPUT, OPTIONAL, default=None]
Double **highRed** [INPUT, OPTIONAL, default=None]
Double **lowGreen** [INPUT, OPTIONAL, default=None]
Double **highGreen** [INPUT, OPTIONAL, default=None]
Double **lowBlue** [INPUT, OPTIONAL, default=None]
Double **highBlue** [INPUT, OPTIONAL, default=None]
Double **redFactor** [INPUT, OPTIONAL, default=None]
Double **greenFactor** [INPUT, OPTIONAL, default=None]
Double **blueFactor** [INPUT, OPTIONAL, default=None]
RgbSimpleImage **rgb** [OUTPUT, MANDATORY, default=None]

API details

Properties

Image red [INPUT, MANDATORY, default=None]

This is the red input image.

Image green [INPUT, MANDATORY, default=None]

This is the green input image.

Image blue [INPUT, MANDATORY, default=None]

This is the blue input image.

Wcs wcs [INPUT, OPTIONAL, default=None]

This is the optional Wcs for the resulting RGB-image.

Double percent [INPUT, MANDATORY, default=98.0]

This is the percentage of values to include.

Double lowRed [INPUT, OPTIONAL, default=None]

This is the low cut level for the red image.

Double highRed [INPUT, OPTIONAL, default=None]

This is the high cut level for the red image.

Double lowGreen [INPUT, OPTIONAL, default=None]

This is the low cut level for the green image.

Double highGreen [INPUT, OPTIONAL, default=None]

This is the high cut level for the green image.

Double lowBlue [INPUT, OPTIONAL, default=None]

This is the low cut level for the blue image.

Double highBlue [INPUT, OPTIONAL, default=None]

This is the high cut level for the blue image.

Double redFactor [INPUT, OPTIONAL, default=None]

This is the factor with which to multiply the intensity values in the red image. Should be left to 1.0 if you specify the cut levels yourself.

Double greenFactor [INPUT, OPTIONAL, default=None]

This is the factor with which to multiply the intensity values in the green image. Should be left to 1.0 if you specify the cut levels yourself.

Double blueFactor [INPUT, OPTIONAL, default=None]

This is the factor with which to multiply the intensity values in the blue image. Should be left to 1.0 if you specify the cut levels yourself.

RgbSimpleImage rgb [OUTPUT, MANDATORY, default=None]

This is the resulting RGB-image.

See also

- Developers Manual: herschel.ia.toolbox.image.CreateRgbImageTask

1.96. cropCube

Full Name:	herschel.ia.toolbox.cube.CropCubeTask
Alias:	cropCube
Type:	Java Task -
Import:	from herschel.ia.toolbox.cube import CropCubeTask
Category	Data cubes/Analysis

Description

Extracts a sub-cube from the provided Cube.

Crops the cube in the spectral and/or spatial domains. If provided as inputs, the following metadata are added to the output cube: colMin, colMax, rowMin, rowMax.

API Summary

Properties
SimpleCube **cube** [INPUT, MANDATORY, default=no default value]
Integer **startSpecIndex** [INPUT, OPTIONAL, default=0]
Integer **endSpecIndex** [INPUT, OPTIONAL, default=depth-1]
Double **startWave** [INPUT, OPTIONAL, default=no default]
Double **endWave** [INPUT, OPTIONAL, default=no default]
Integer **colMin** [INPUT, OPTIONAL, default=0]
Integer **colMax** [INPUT, OPTIONAL, default=width-1]
Integer **rowMin** [INPUT, OPTIONAL, default=0]
Integer **rowMax** [INPUT, OPTIONAL, default=height-1]
SimpleCube **croppedCube** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

SimpleCube cube [INPUT, MANDATORY, default=no default value]

The Cube from which we want to extract a smaller part

Integer startSpecIndex [INPUT, OPTIONAL, default=0]

The first spectral index. Ignored when startWave is provided.

Integer endSpecIndex [INPUT, OPTIONAL, default=depth-1]

The last spectral index. Ignored when endWave is provided.

Double startWave [INPUT, OPTIONAL, default=no default]

The start wavelength/frequencey of the spectral range.

Double endWave [INPUT, OPTIONAL, default=no default]

The end wavelength/frequencey of the spectral range.

Integer colMin [INPUT, OPTIONAL, default=0]

the minimum column value for the spatial extraction

Integer colMax [INPUT, OPTIONAL, default=width-1]

the maximum column value for the spatial extraction

Integer rowMin [INPUT, OPTIONAL, default=0]

the minimum row value for the spatial extraction

Integer rowMax [INPUT, OPTIONAL, default=height-1]

the maximum row value for the spatial extraction

SimpleCube croppedCube [OUTPUT, MANDATORY, default=no default value]

Output cube (same type of product as input cube)

See also

- Developers Manual: herschel.ia.toolbox.cube.CropCubeTask

1.97. crop

Full Name:	herschel.ia.toolbox.image.CropTask
Alias:	crop
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import CropTask
Category	Images/Analysis

Description

This is a task to crop an image to a rectangular region.

This is a task to crop an image to a rectangular region. You must specify the rows and column, between which to crop the image.

Example

Example 1: This is how you can crop an image between row 11 and 240, and column 240 and 300

```
cropped = crop(image = myImage, row1 = 11, row2 = 55, column1 = 240, column2 =
 300)
```

API Summary

Jython Syntax

```
crop(image, 11, 240, 55, 300)
```

Properties

`Image image [INPUT, MANDATORY, default=None]`
`int row1 [INPUT, OPTIONAL, default=0]`
`int column1 [INPUT, OPTIONAL, default=0]`
`int row2 [INPUT, OPTIONAL, default=height]`
`int column2 [INPUT, OPTIONAL, default=width]`
`Image croppedImage [OUTPUT, MANDATORY, default=None]`

API details

Properties

`Image image [INPUT, MANDATORY, default=None]`

This is the input image.

`int row1 [INPUT, OPTIONAL, default=0]`

This is the minimum row of the rectangle to which to crop the image. If this is not specified, 0 is used.

`int column1 [INPUT, OPTIONAL, default=0]`

This is the minimum column of the rectangle to which to crop the image. If this is not specified, 0 is used.

int row2 [INPUT, OPTIONAL, default=height]

This is the maximum row of the rectangle to which to crop the image. If this is not specified, the height of the image is used.

int column2 [INPUT, OPTIONAL, default=width]

This is the maximum column to which to crop the image. If this is not specified, the width of image is used.

Image croppedImage [OUTPUT, MANDATORY, default=None]

This is the resulting cropped image.

See also

- Developers Manual: herschel.ia.toolbox.image.CropTask

1.98. crossCorrelation

Full Name:	herschel.ia.toolbox.image.CrossCorrelationTask
Alias:	crossCorrelation
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import CrossCorrelationTask
Category	Images/Analysis

Description

This is a task to calculate the linear Pearson correlation coefficient of two images.

This is a task to calculate the linear Pearson correlation coefficient of two images. We start from two M x N images (M = number of rows, N = number of columns) and calculate the N x N linear Pearson coeffient matrix . The i'th row and j'th column element corresponds to the correlation of the i'th and j'th columns of the M x N matrices.

Example

Example 1: This is an example of how to use the crossCorrelation :

```
correlation = crossCorrelation(image1 = myImage1, image2 = myImage2)
```

API Summary

Properties
Image **image1** [INPUT, MANDATORY, default=None]
Image **image2** [INPUT, MANDATORY, default=None]
Double2d **correlation** [OUTPUT, MANDATORY, default=None]

API details

Properties

Image image1 [INPUT, MANDATORY, default=None]

This is the first image.

Image image2 [INPUT, MANDATORY, default=None]

This is the second image.

Double2d correlation [OUTPUT, MANDATORY, default=None]

This is the matrix with the linear Pearson corelation coefficients.

See also

- Developers Manual: herschel.ia.toolbox.image.CrossCorrelationTask

1.99. CubicSplineInterpolator

Full Name:	herschel.ia.numeric.toolbox.interp.CubicSplineInterpolator
Alias:	CubicSplineInterpolator
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.interp import CubicSplineInterpolator
Category	Mathematics/Interpolation

Description

Given a set of knots (x,y), this function computes values at arbitrary positions by use of a cubic spline.

It assumes that the second derivatives are zero at the end points (i.e. a natural spline). This can only be applied to numeric arrays of rank 1."

Example

Example 1: Create and apply a CubicSplineInterpolator on a Double1d

```
# create some knots
x=Double1d.range(10) # [0.0,1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0]
f=CubicSplineInterpolator(x,SQUARE(x))
u=Float1d([1,1.1,2,2.6,3,3.9])
print f(u) # [1.0,1.1970944,4.0,6.7656145,9.0,15.209593]
```

API Summary

Jython Syntax

```
&lt;f&gt;=CubicSplineInterpolator(&lt;x&gt;,&lt;y&gt;[,allowEx-
trapolation])
```

Properties

Double1d **x** [INPUT, MANDATORY, default=no default value]

Double1d **y** [INPUT, OPTIONAL, default=false]

boolean **allowExtrapolation** [INPUT, OPTIONAL, default=false]

API details

Properties

Double1d x [INPUT, MANDATORY, default=no default value]

The knots are Double1d

Double1d y [INPUT, OPTIONAL, default=false]

The knots are Double1d

boolean allowExtrapolation [INPUT, OPTIONAL, default=false]

Extrapolation is not allowed by default. Boolean.TRUE allows extrapolation.

See also

- LinearInterpolator
- NearestNeighborInterpolator
- Developers Manual: herschel.ia.numeric.toolbox.interp.CubicSplineInterpolator

History

- `2009-02-26 - LZ`: Changed the example output with the real time result

1.100. cutLevels

Full Name:	herschel.ia.toolbox.image.CutLevelsTask
Alias:	cutLevels
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import CutLevelsTask
Category	Images/Analysis

Description

This is a task to determine the cut levels of an image.

This is a task to determine the cut levels of an image. These can calculated using two methods :

- PERCENT : A certain percentage of the pixels is calculated.
- MEDIAN_FILTER : A window of 7 pixels is taken. From this window, the median is taken. This median is then compared with the minimum and maximum value of the image. 21 pixels are skipped in the column dimension and 3 rows are skipped.

The result is returned as an array of doubles. The first value is the minimum value and the second value is the maximum value.

Examples

Example 1: This is how you can calculate the cut levels of an image, where 95% of the intensity values are included:

```
levels = cutLevels(image = myImage, method = CutLevelsTask.PERCENT, percent =
 95.0)
```

Example 2: This is how you can calculate the cut levels of an image, using the median filter:

```
levels = cutLevels(image = myImage, method = CutLevelsTask.MEDIAN_FILTER)
```

API Summary

Jython Syntax

```
cutLevels(image, CutLevels.PERCENT, 95.0)
```

Properties

```
Image image [INPUT, MANDATORY, default=None]
int method [INPUT, OPTIONAL, default=CutLevels.MEDIAN_FILTER]
double percent [INPUT, OPTIONAL, default=99.5]
double[] cutLevels [OUTPUT, MANDATORY, default=None]
```

API details

Properties

`Image image [INPUT, MANDATORY, default=None]`

This is the input image.

int method [INPUT, OPTIONAL, default=CutLevels.MEDIAN_FILTER]

This is the method to use to calculte the cut levels of the input image. Possible values are CutLevels.PERCENT and CutLevels.MEDIAN_FILTER.

double percent [INPUT, OPTIONAL, default=99.5]

This is the percentage of pixels to use in the calculation of the cut levels.

double[] cutLevels [OUTPUT, MANDATORY, default=None]

The minimum and maximum cut level.

See also

- Developers Manual: herschel.ia.toolbox.image.CutLevelsTask

1.101. CWavelet

Full Name:	herschel.ia.numeric.toolbox.wavelet.wlibrary.CWavelet
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.wavelet.wlibrary import CWavelet

Description

A continuous wavelet

Examples

Example 1: Loading continuous wavelets implicitly

```
from herschel.ia.numeric.toolbox.wavelet.wlibrary import WaveletLoader
loader = WaveletLoader()
# If more than one type of wavelet use the given name, then use the continuous
 wavelet
loader.setCwtPreference()
wavelet = loader.get("MexicanHat")
```

Example 2: Loading continuous wavelets explicitly

```
from herschel.ia.numeric.toolbox.wavelet.wlibrary import WaveletLoader
wavelet = WaveletLoader.loadContinuousWavelet("MexicanHat")
```

Example 3: Creating custom continuous wavelets in Jython

```
from herschel.ia.numeric.toolbox.wavelet import ContinuousWavelet
from herschel.ia.numeric.toolbox.wavelet.wlibrary import CWavelet
# To use a custom continuous wavelet, the CWavelet class should be extended
class CustomCWavelet(CWavelet):
    '''A custom CWavelet subclass'''
    def __init__(self):
        '''Initializes the continuous wavelet'''
        CWavelet.__init__(self, "Your class's name")
        # Initialize the admissibility factor
        self.setAdmissibilityFactor(self.newAdmissibilityFactor());
    def get(self, time):
        '''Returns the wavelet function value for the given time (in
 seconds).'''
        return Complex(time)
    def getScaling(self, time):
        '''Returns the scaling function value for the given time (in
 seconds).'''
        return Complex(time)
    def getFt(self, frequency):
        '''Returns the wavelet function value for the given frequency (in
 Hertz).'''
        return Complex(frequency)
    def getScalingFt(self, frequency):
        '''Returns the scaling function value for the given frequency (in
 Hertz).'''
        return Complex(frequency)
    def getDerivative(self, time):
        '''Returns the derivative of the wavelet function for the given time
 (in seconds).'''
        return Complex(1.0)
wavelet = CustomCWavelet()
# Rather than a wavelet name, we can construct a continuous wavelet operation
 using our custom wavelet
algo = ContinuousWavelet(wavelet)
double1d_input_signal = Double1d.range(100)
```

Example 3: Creating custom continuous wavelets in Jython

```
coefficients = algo.decompose(double1d_input_signal);
print coefficients
double1d_output_signal = algo.synthesis(coefficients)
```

Example 4: A custom CWavelet that emulates the existing Haar wavelet implementation

```
from herschel.ia.numeric.toolbox import RealFunction
from herschel.ia.numeric.toolbox.wavelet import ContinuousWavelet
from herschel.ia.numeric.toolbox.wavelet import WException
from herschel.ia.numeric.toolbox.wavelet.wlibrary import CWavelet
from herschel.ia.numeric.toolbox.wavelet.wlibrary import Haar
class HaarWavelet(CWavelet, RealFunction):
    '''A reimplementation of the Haar class as a custom CWavelet subclass'''
    def __init__(self):
        CWavelet.__init__(self, "Haar")
        # Initialize the admissibility factor
        self.setAdmissibilityFactor(self.newAdmissibilityFactor())
    def calc(self, x):
        '''Override RealFunction.calc(double)'''
        return self.get(x).real
    def get(self, time):
        '''Returns the wavelet function value for the given time (in
 seconds).'''
        if -0.5 <= time and time < 0.5:
            if time < 0.0:
                get = Complex(-1.0, 0.0)
            else:
                get = Complex(+1.0, 0.0)
        else:
            get = Complex(0.0, 0.0)
        return get
    def getScaling(self, time):
        '''Returns the scaling function value for the given time (in
 seconds).'''
        if -0.5 <= time and time <= 0.5:
            scaling = Complex(+1.0, 0.0)
        else:
            scaling = Complex(0.0, 0.0)
        return scaling
    def getFt(self, frequency):
        '''Returns the wavelet function value for the given frequency (in
 Hertz).'''
        return Complex(+1.0, 0.0)
    def getScalingFt(self, frequency):
        '''Returns the scaling function value for the given frequency (in
 Hertz).'''
        return None
    def getDerivative(self, time):
        '''Returns the derivative of the wavelet function for the given time
 (in seconds).'''
        raise WException("Haar wavelet function is not derivable");
oldWavelet = Haar()
newWavelet = HaarWavelet()
# The following prints false because the wavelet classes are not the same
print 'The wavelets are equivalent: ', newWavelet == oldWavelet
signal = Double1d.range(100)
oldCoeffs = ContinuousWavelet(oldWavelet).decompose(signal)
newCoeffs = ContinuousWavelet(newWavelet).decompose(signal)
# The following prints false because the tree wavelet classes are not the same
print 'The coefficient trees are equivalent:', oldCoeffs == newCoeffs
# The following prints true because the coefficients are the same
print 'The decomposition coefficients are equivalent:', oldCoeffs.root ==
 newCoeffs.root
```

See also

- Developers Manual: herschel.ia.numeric.toolbox.wavelet.wlibrary.CWavelet

History

- `2010-09-10 - initial`: version

1.102. Cwt

Full Name:	herschel.ia.numeric.toolbox.wavelet.walgo.Cwt
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.wavelet.walgo import Cwt

Description

A continuous wavelet transform algorithm

This decomposition is redundant and is well adapted when we want a screen view of the coefficients.

Wavelet is successively dilated according to the scale in order to create coefficients.

The scale is computed from voice and decade according to the following formula:

- scale = $2^{octave+n*voice}$.
- voice = 1/resolution.

Examples

Example 1: Perform a continuous wavelet transform

```
from herschel.ia.numeric.toolbox.wavelet.wlibrary import WaveletLoader
from herschel.ia.numeric.toolbox.wavelet.walgo import Cwt
from herschel.ia.numeric.toolbox.wavelet.wutil import WBorder
from herschel.ia.numeric.toolbox.wavelet.wdemo import WSigGenerator
sigGen = WSigGenerator()
signal = sigGen.getPredefinedRealFunction1()
wloader = WaveletLoader()
# -- ask to the loader to load a continuous wavelet
wloader.setCwtPreference()
wavelet = wloader.get("MexicanHat")
cwt = Cwt()
coefs = Cwt().decompose(signal,wavelet,WBorder.SYMMETRIC)
Display(coefs.data)
# -- perform synthesis
res  = cwt.synthesis(coefs)
```

Example 2: How to use Wavelet Transform Modulo Maxima Lines (WTMML) mode

```
from herschel.ia.numeric.toolbox.wavelet.wlibrary import WaveletLoader
from herschel.ia.numeric.toolbox.wavelet.walgo    import Cwt
from herschel.ia.numeric.toolbox.wavelet.wutil    import WBorder
from herschel.ia.numeric.toolbox.wavelet.wdemo   import WSigGenerator
# -- creates dummy one-dimensional signal
sigGen  = WSigGenerator()
signal  = sigGen.getPredefinedRealFunction1()
# -- get the loader of wavelet
wloader = WaveletLoader()
# -- ask to the loader to load a continuous wavelet
wloader.setCwtPreference()
# -- load a continuous wavelet, the name is given in parameter
wavelet  = wloader.get("MexicanHat")
# -- activates Wavelet Transform Modulo Maxima Lines (WTMML) mode
cwt = Cwt()
cwt.activatesWTMML()
# -- perform a decomposition
coefs   = Cwt().decompose(signal,wavelet,WBorder.SYMMETRIC)
# -- get and display data
Display(coefs.data)
# -- cancel WTMML mode - useful as synthesis has no meaning in this mode
```

Example 2: How to use Wavelet Transform Modulo Maxima Lines (WTMML) mode

```
cwt.deactivatesWTMML()
```

See also

- Developers Manual: herschel.ia.numeric.toolbox.wavelet.walgo.Cwt

History

- `2010-09-10 - initial`: version

1.103. DataFormatter

Full Name:	herschel.ia.numeric.DataFormatter
Type:	Java Class -
Import:	from herschel.ia.numeric import DataFormatter

Description

Print objects to a String with special attention to array lay out.

The dataformatter classifies objects to be printed as follows:

- Array, or object that can be converted to an array. In this case the first few elements are printed (recursively), preceded by an opening brace, separated by separator strings, and finished off by a closing brace. If the array has more than the maximum number of printable elements an ellipis string is printed.
- If the object is Java primitive type or one of the standard wrapper classes for them, it is printed using {@link java.util.Formatter} with a suitable format.
- Integral types (byte,short,int,long) can also be displayed with an additional "%[flag][width]p" format. It will display the bitpattern; [flag] can be a '0' which pads the result with leading zeroes and [width] it the minimum width of the result. Useful for flags and masks. The bits are ordered (as always) right to left, as a binary number.
- Other objects are printed using `Formatter` and their `toString` method.

The strings used in the array layout and the format specifications can be replaced.

Example

Example 1: DataFormatter

```
F = DataFormatter()
y = Double1d( [1,2,3,4,4,3,2,1] )
print F.p( y )
# [   1.000   2.000   3.000   4.000 ...]
print F.p( y, "%6.2f", 8 )
# [  1.00  2.00  3.00  4.00  4.00  3.00  2.00  1.00]
print F.p( 44, "%p" )
#101100
print F.p( 44, "%4p" )
#101100
print F.p( 44, "%10p" )
#    101100
print F.p( 44, "%010p" )
#0000101100
a = Int1d.range(4) + 40
print F.p( a, "%010p" )
#[0000101000 0000101001 0000101010 0000101011]
b = Int1d.range(6)
print F.p( b, "%p", 8 )
#[0 1 10 11 100 101]
```

Limitations

- The implementation takes shortcuts in analyzing the array dimensions. In irregular structures, the array dimensions might be underestimated. This depends on the setting of the MAXIMUM layout parameter, For example, using defaults settings, `Object[] x = { "a", "b", "c", "d", "e", new Integer[10] }` is seen as 1-dimensional, returning `[ a b c d ... ]`

and `Object[] x = { new Integer[10], "a" }` is seen as 2-dimensional, returning `[ [ null null null null ... ], a ]`

- The default layout settings represent a more-dimensional array as a multi-line string, and try to achieve alignment for the deepest two levels. This works if a single ASCII newline is the line separator and a fixed-width font is used.
- An object "can be converted to array" when the object has an accessible method 'toArray()' that takes no arguments and returns an array.
- Objects of this class are not thread-safe.

See also

- Developers Manual: herschel.ia.numeric.DataFormatter

1.104. decompress

Full Name:	herschel.ia.toolbox.util.DecompressTask
Alias:	decompress
Type:	Java Task -
Import:	from herschel.ia.toolbox.util import DecompressTask
Category	Session utilities

Description

Decompresses archives.

Decompresses a (relative path) compressed archive in a user-chosen directory Two first arguments are mandatory. Supported algorimths are: TAR, ZIP, GZIP (and TAR.GAZ).

Example

Example 1: Untarring in temp directory

```
decompress("./mytar.tar", "/tmp")
```

API Summary

Jython Syntax

```
decompress(&lt;archive&gt;, &lt;dirout&gt; [, &lt;compres-
sion&gt;="Guess", &lt;overwrite&gt;=True])
```

Properties

String **archive** [INPUT, MANDATORY, default=null]
String **dirout** [INPUT, MANDATORY, default=null]
String **compression** [INPUT, OPTIONAL, default="Guess"]
Boolean **overwrite** [INPUT, OPTIONAL, default=True]

Limitations

The archive will not be deleted.

API details

Properties

String archive [INPUT, MANDATORY, default=null]

Path of the archive to be decompressed.

String dirout [INPUT, MANDATORY, default=null]

Directory (that may not exist yet) where we want to decompress the archive.

String compression [INPUT, OPTIONAL, default="Guess"]

Type of compression of the input file. Possible values are:

String compression [INPUT, OPTIONAL, default="Guess"]

- "Guess" (default, find-outs the compression used)
- "ZIP" (archive was compressed with zip)
- "TAR" (archive was packed with tar)
- "GZ" (archive was compressed with gzip)
- "TGZ" (archive was compressed with gzip after being packed with tar)

Boolean overwrite [INPUT, OPTIONAL, default=True]

If set to false, it will not update already existing files.

See also

- compress
- Developers Manual: herschel.ia.toolbox.util.DecompressTask

History

- 2010-05-25 - JDS: first release
- 2010-06-09 - JDS: expanded, renamed to decompress
- 2010-08-16 - JDS: added an overwrite flag

1.105. DERIV

Full Name:	herschel.ia.numeric.toolbox.integr.Deriv
Alias:	DERIV
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.integr import Deriv
Category	Mathematics/Integration

Description

Given a set of knots, this function performs numerical differentiation using 3-point, Lagrangian interpolation.

Y variable to be differentiated.

Example

Example 1: Create Deriv on a Double1d

```
# Create y 1 dimension data  to be differentiated.
from herschel.ia.numeric.toolbox.integr import  *
y=Double1d.range(10) # [0.0,1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0]
result=DERIV(y)
print result # [1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0]
```

API Summary

Jython Syntax
`f=DERIV(y)`

See also

- Developers Manual: herschel.ia.numeric.toolbox.integr.Deriv

History

- `2010-07-20 - AH`: HCSS-6668 Implement DERIVATIVE function

1.106. DETERMINANT

Full Name:	herschel.ia.numeric.toolbox.matrix.MatrixDeterminant
Alias:	DETERMINANT
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.matrix import MatrixDeterminant
Category	Mathematics/Matrices

Description

Yields the determinant of a square matrix.

Example

Example 1: Apply a DETERMINANT to a Double2d matrix

```
A=Double2d([ [1,2],[3,4] ])
print DETERMINANT(A) # -2
```

API Summary

Jython Syntax

```
y=DETERMINANT(x)
```

Properties

`any square matrix x [INPUT, MANDATORY, default=no default value]`

`double y [INPUT, OPTIONAL, default=false]`

Miscellaneous

Does not work for complex matrices.

API details

Properties

`any square matrix x [INPUT, MANDATORY, default=no default value]`

Any square matrix

`double y [INPUT, OPTIONAL, default=false]`

Returns a double

See also

- CubicSplineInterpolator
- LinearInterpolator
- Developers Manual: herschel.ia.numeric.toolbox.matrix.MatrixDeterminant

1.107. dFT2d

Full Name:	herschel.ia.toolbox.image.DFT2dTask
Alias:	dFT2d
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import DFT2dTask
Category	Mathematics/Signal processing

Description

This is a task to calculate the Discrete Fourier Transform and the power spectrum of an image.

This is a task to calculate the Discrete Fourier Transform and the power spectrum of an image.

Example

Example 1: Example of how you can calculate the Discrete Fourier Transform and the power spectrumof an image :

```
transform = dFT2d(image = myImage)
spectrum = dFT2d.spectrum
```

API Summary

Properties
Image **image** [INPUT, MANDATORY, default=None]
Complex2d **transform** [OUTPUT, MANDATORY, default=None]
Double2d **spectrum** [OUTPUT, MANDATORY, default=None]

API details

Properties

Image image [INPUT, MANDATORY, default=None]

This is the input image.

Complex2d transform [OUTPUT, MANDATORY, default=None]

This is the Discrete Fourier Transform.

Double2d spectrum [OUTPUT, MANDATORY, default=None]

This is the power spectrum.

See also

- Developers Manual: herschel.ia.toolbox.image.DFT2dTask

1.108. DiscreteWavelet

Full Name:	herschel.ia.numeric.toolbox.wavelet.DiscreteWavelet
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.wavelet import DiscreteWavelet

Description

Discrete Wavelet

Performs a discrete wavelet transform using an internal wavelet and an internal algorithm.

Examples

Example 1: Import statements

```
from herschel.ia.numeric.toolbox.wavelet import DiscreteWavelet
from herschel.ia.numeric.toolbox.wavelet.wutil import WBorder
```

Example 2: One-dimensional signal

```
from herschel.ia.numeric.toolbox.wavelet import DiscreteWavelet
signal1d = Double1d.range(10000)     # Get a Double1d signal to decompose
algo = DiscreteWavelet("db5")        # Define the algorithm and wavelet to use
coefs = algo.decompose(signal1d)     # Decompose the signal into a coefficients
 tree
print coefs.depth                    # Show the decomposition depth
detail = coefs.getDetailForLevel(1)  # Get the detail node for level 1
approx = coefs.getApproximation()    # Get the approximation node at the level
 equal to the decomposition depth
PlotXY(detail.data)                  # Plot the detail coefficients
PlotXY(approx.data)                  # Plot the approximation coefficients
res = algo.synthesis(coefs)          # Synthesize a new signal from the
 coefficients tree
res -= signal1d                      # Subtract the original signal to compare
 their similarity
print STDDEV(signal1d), STDDEV(res)  # Evaluate the deviation (the synthesized
 signal STDDEV should be close to 0.0)
coefs.reset(1)                       # Reset the level 1 coefficients, setting
 them to 0.0
coefs.multiply(1, 2.50)              # Multiply the level 1 coefficients by
 2.50
coefs.add(1, 4.00)                   # Add 4.00 to the level 1 coefficients
coefs.divide(1, 3.00)                # Divide the level 1 coefficients by 3.00
detail = coefs.getDetailForLevel(1)  # Get the detail node for level 1
print detail.scale                   # Get the scale of the detail coefficients
data = detail.data                   # Get the detail coefficients
data[2] = 4.0                        # Set the coefficient at index 2 to the
 value 4.0
detail.data = data                   # Set the detail coefficients
```

Example 3: Two-dimensional signal

```
from herschel.ia.numeric.toolbox.wavelet import DiscreteWavelet
# Generate an interesting signal to decompose
totalRows = 256
totalCols = 256
signal2d = Double2d(totalRows, totalCols)
for y in range(totalRows):
    for x in range(totalCols):
        signal = 0.0
        signal += COS(x * 0.49 - 3.25) * COS(y * 0.12 - 2.50)
        signal += SIN(x * 0.31 - 1.75) * SIN(y * 0.17 - 1.25)
        signal += COS(x * 0.26 - 0.25) * COS(y * 0.37 + 0.00)
        signal += SIN(x * 0.17 + 1.25) * SIN(y * 0.18 + 1.25)
        signal += COS(x * 0.41 + 2.75) * COS(y * 0.05 + 2.50)
```

Example 3: Two-dimensional signal

```
        signal += SIN(x * 0.07 + 4.25) * SIN(y * 0.22 + 3.75)
        signal2d.set(y, x, signal)
algo = DiscreteWavelet("db5")               # Create a discrete wavelet
 transform configuration that will load and
                                            #     use the discrete wavelet with
 the given name
coefs = algo.decompose(signal2d)            # Decompose our two-dimensional
 signal into a coefficients tree
horizontal = coefs.getHorizontalForLevel(1) # Get the horizontal detail node
 for level 1
vertical = coefs.getVerticalForLevel(1)     # Get the vertical detail node for
 level 1
diagonal = coefs.getDiagonalForLevel(1)     # Get the diagonal detail node for
 level 1
composite = coefs.compose(1)                # Get composition of all detail
 coefficients and the final approximation
                                            #     coefficients from the root
 node to the given level
composite = coefs.compose()                 # Get a nested series of images of
 the coefficient down to the max level
Display(horizontal.data)                    # Display the horizontal detail
 coefficients
Display(vertical.data)                      # Display the vertical detail
 coefficients
Display(diagonal.data)                      # Display the diagonal detail
 coefficients
Display(composite)                          # Display the nested composite
 image
res = algo.synthesis(coefs)                 # Synthesize a two-dimensional
 signal
diff = res - signal2d                       # Subtract for comparison
print STDDEV(signal2d), STDDEV(diff)        # Evaluate the deviation (should be
 minimal for the resulting signal)
print MIN(signal2d), MIN(diff)              # Evaluate the minimum (should be
 minimal for the resulting signal)
print MAX(signal2d), MAX(diff)              # Evaluate the maximum (should be
 minimal for the resulting signal)
# --
coefs = algo.decompose(signal2d)            # Decompose our two-dimensional
 signal into a two-dimensional tree
coefs.getVerticalForLevel(2).reset()        # Reset the vertical detail
 coefficients for level 1
horizontal = coefs.getHorizontalForLevel(1) # Get the horizontal detail node
 for level 1
horizontal.multiply(3.2)                    # Multiply the value 3.2 and the
 coefficients at level 1.
horizontal.add(-5.0)                        # Subtract the value 5.0 and the
 coefficients at level 1.
print horizontal.scale                      # Get the scale of the horizontal
 detail node
diagonal = coefs.getDiagonalForLevel(1)     # Get the diagonal detail
 coefficients for level 1
data = diagonal.data                        # Get the horizontal detail
 coefficients.
data.set(2, 2, 4.0)                         # Set the value 4 at location 2 for
 the level 1.
diagonal.data = data * 0.5                  # Replace diagonal coefficients
 with new ones
# --
coefs.reset(1)                      # Reset coefficients at level 1 to 0.0 for
 all details (horizontal, vertical,
                                    #   diagonal).
coefs.add(1, 7.25)                  # Add the value 7.25 to the coefficients
 for level 1, this operation is
                                    #   done for all details.
coefs.multiply(1, 3.5)              # Multiply the coefficients at level 1 by
 the value 3.5, this operation is
                                    #   done for all details.
coefs.subtract(1, -5.75)            # Subtract the value 5.0 from the
 coefficients at level 1, this operation is
```

Example 3: Two-dimensional signal

```
                                    #   done for all details.
coefs.divide(1, 2.0)                # Divide the coefficients at level 1 by
 the value 2.0, this operation is
                                    #   done for all details.
```

Example 4: Extra information

```
from herschel.ia.numeric.toolbox.wavelet import DiscreteWavelet
signal1d = Double1d(10000)
algo = DiscreteWavelet("db5")
coefs = algo.decompose(signal1d)
print coefs                         # Give information on the wavelet used
print coefs.wavelet                 # Give detail description of the wavelet
 used
print coefs.wavelet.family          # Give the wavelet family
print coefs.wavelet.symmetry        # Give the symmetry of the wavelet
print coefs.wavelet.length          # Give the length of the wavelet
print coefs.wavelet.orthogonality   # Give the orthogonality of the wavelet
print coefs.border                  # Border management use during
 decomposition
# See Wavelet for more information
```

Example 5: Border management

```
from herschel.ia.numeric.toolbox.wavelet import DiscreteWavelet
from herschel.ia.numeric.toolbox.wavelet.wutil import WBorder
signal1d = Double1d(10000)
signal2d = Double2d(100, 100)
algo = DiscreteWavelet("db5")
algo.decompose(signal2d,WBorder.SYMMETRIC)
algo.decompose(signal2d,WBorder.ZERO)
algo.decompose(signal1d,WBorder.SYMMETRIC)
# See WBorder for more information
```

See also

- Developers Manual: herschel.ia.numeric.toolbox.wavelet.DiscreteWavelet

1.109. Display

Full Name:	herschel.ia.gui.image.Display
Alias:	Display
Type:	Java Class -
Import:	from herschel.ia.gui.image import Display
Category	Images/Display

Description

A class to display images.

This class can display a SimpleImage, SimpleCube or any numeric2d or numeric3d object. A status bar, color bar, a zoomed section of the image and an overview of the image are also available.

Examples

Example 1: Display a SimpleImage

```
d = Display(mySimpleImage)
d.close()
```

Example 2: A basic example on how to Display a Double2d

```
im = Double2d(600, 400)
for i in range(600):
    for j in range(400):
        im.set(i, j, i + j)
d = Display(im)
d.close()
```

API Summary

Constructors
Display The standard constructor for Display.
Display (boolean useAsComponent, boolean imageVisible) A constructor to display the image in the background.
Display (boolean useAsComponent) A constructor to use Display as a component.
Display (Image imds) A constructor which displays an Image.
Display (Cube cube) A constructor which displays a Cube.
Display (RgbImage imds) A constructor which displays an RgbImage.
Display (Image imds, pabooleanramType imageVisible) A Display constructor for an Image.
Display (Cube cube, boolean imageVisible) A Display constructor for a Cube.

Constructors
Display (Array2dData data) A constructor to display a Numeric2d.
Display (Array2dData data, boolean imageVisible) A Display constructor for Array2dData.
Display (Array3dData data) A constructor to display a numeric3d.
Display (Array3dData data, boolean imageVisible) A display constructor for 3d data.

Methods
setSize (int x, int y) Adapt the dimensions of the frame.
getSize Returns the dimensions of the frame.
int **getLayerShown** Returns the number of the layer which is shown.
setVisible (boolean imageVisible) Toggles the display visible or invisible.
JPanel **getComposedComponent** Returns a panel with all components of this Display on it.
ImageColorbar **getColorBarComponent** Returns the color bar component.
DivaMainImageDisplay **getComponent** Returns the component where the image is displayed.
ImagePanner **getPannerComponent** Returns the panner component with the overview of the image.
StatusPanel **getStatusPanelComponent** Returns the component which contains the status bar of the image.
setStatusPanelComponent (StatusPanel imageDisplayStatusPanel) Set a new status panel for the display.
ImageZoom **getZoomComponent** Returns the component with the zoomed view.
setImage (Array2dData imageData) Set a new numeric2d image.
setImage (Array3dData imageData) Sets a new numeric3d image.
setImage (Image imds) Sets a new Image to Display.
setImage (Cube cube) Sets a new cube to display.
setImage (RgbImage imds) Sets a new RgbImage to Display.

Methods
addLayer (Layer layer) Adds a new layer to the display.
addLayer (Array2dData imageData) Adds a new layer with numeric2d data.
addLayer (Array3dData imageData) Adds a new layer with numeric3d data.
addLayer (Image imds) Adds a new layer with an Image to the display.
addLayer (RgbImage imds) Adds a new layer with an RgbImage to the display.
addLayer (Cube cube) Adds a new layer with a Cube to the display.
addLayer (SkyMask skyMask) Adds a new layer with a SkyMask to the display.
Layer **getLayer** Returns the shown layer.
Layer **getLayer** (int i) Returns the Layer object.
showLayer (int i) Shows a chosen layer.
removeLayer Removes the current layer.
removeLayer (int i) Removes the given Layer.
updateImage (Array2dData imageData) Updates the layer with numeric2d data.
updateImage (Image imds) Updates the layer with an Image.
updateImage (RgbImage imds) Updates the layer with an Image.
updateImage (Array3dData imageData) Updates the layer with numeric3d data.
updateImage (Cube imds) Updates the layer with a cube.
setBackground (Color bgColor) Sets the background color.
getBackground Returns the background color.
annotationToolbox Fires up the annotation toolbox.
skyMaskToolbox

Methods
Fires up the SkyMask toolbox.
CanvasFigure **addAnnotation** (String text, double row, double column)
Adds a text annotation.
TextAnnotation **addTextAnnotation** (String text, double row, double column, Font font, Color fontColor)
Adds a text annotation at given pixel coordinates.
CanvasFigure **addAnnotationWorldCoordinates** (String text, double ra, double decl)
Adds a text annotation at given sky coordinates.
TextAnnotation **addTextAnnotationWorldCoordinates** (String text, double ra, double decl, Font font, Color fontColor)
Adds a text annotation at given sky coordinates.
CanvasFigure **addGreekAnnotation** (String text, double row, double column)
Adds a Greek annotation at the given pixel coordinates.
TextAnnotation **addGreekTextAnnotation** (String text, double row, double column)
Adds a Greek annotation at the given pixel coordinates.
CanvasFigure **addGreekAnnotationWorldCoordinates** (String text, double ra, double decl)
Adds a Greek annotation at the given world coordinates.
TextAnnotation **addGreekTextAnnotationWorldCoordinates** (String text, double ra, double decl)
Adds a Greek annotation at the given world coordinates.
Font **getAnnotationFont**
Returns the default font for annotations.
Font **getAnnotationFont** (double row, double column)
Returns the font for the specified annotation.
Font **getAnnotationFontWorldCoordinates** (double ra, double dec)
Returns the font for the specified annotation.
Color **getAnnotationFontColor**
Returns the default color for annotations.
Color **getAnnotationFontColor** (double row, double column)
Returns the color of the specified annotation.
Color **getAnnotationFontColorWorldCoordinates** (double ra, double dec)
Returns the color of the specified annotation.
removeAnnotation (double row, double column)
Removes the specified annotation.
removeAnnotationWorldCoordinates (double ra, double dec)
Remove the specified annotation.
removeAnnotation (CanvasFigure fig)
Removes the specified annotation.

Methods
removeAnnotations Removes all annotations.
setAnnotationFont (Font f) Changes the font of all Annotations.
setAnnotationFont (double row, double column, Font f) Changes the font of the specified annotations.
setAnnotationFontWorldCoordinates (double ra, double dec, Font f) Changes the font of the annotations at given sky coordinates.
setAnnotationFont (int size) Changes the font size of all annotations.
setAnnotationFont (double row, double column, int size) Changes the font size of the specified annotations.
setAnnotationFontWorldCoordinates (double ra, double dec, int size) Changes the font size of the specified annotations.
setAnnotationFontColor (Color color) Sets the default color for annotations.
setAnnotationFontColor (double row, double column, Color color) Sets the font color for the specified annotation.
setAnnotationFontColorWorldCoordinates (double ra, double dec, Color color) Sets the font color for the specified annotation.
EllipseAnnotation **addEllipseAnnotation** (double row, double column, double w, double h, float lineWidth, Color color, double positionAngle, boolean fixed) Adds an ellipse.
EllipseAnnotation **addCircleAnnotation** (double row, double column, double radius, float lineWidth, Color color) Adds a circle.
LineAnnotation **addLineAnnotation** (double r1, double c1, double r2, double c2, float lineWidth, Color color, boolean fixed) Adds a line.
CrossAnnotation **addCrossAnnotation** (double row, double column, double size, float width, Color color) Adds a cross.
PolygonAnnotation **addPolygonAnnotation** (double[] coords, float lineWidth, Color color, boolean fixed) Adds a polygon.
PolylineAnnotation **addPolylineAnnotation** (double[] coords, float lineWidth, Color color, boolean fixed) Adds a polyline.
RectangleAnnotation **addRectangleAnnotation** (double row, double column, double w, double h, float lineWidth, Color color, boolean fixed)

Methods
Adds a rectangle.
addCompassAnnotation (double row, double column, double size, float width, Color color) Adds a compass with the north and east directions.
addArcSecsAnnotation (double arcsecs, double row, double column, float lineWidth, Color color) Adds a line which of a requested number of arcseconds to the image.
ImageContourAnnotation **addImageContourAnnotation** (Color imageContour, Color colors, Color minLength) Adds an ImageContour.
ImageContourAnnotation **addImageContourAnnotation** (ImageContour imageContour) Adds an ImageContour.
ImageContourAnnotation **addWcsImageContourAnnotation** (ImageContour imageContour, ArrayList colors, int minLength) Add an ImageContour.
ContourLevelAnnotation **addContourLevelAnnotation** (ContourLevel level, Wcs wcs, int minLength) Adds a ContourLevel.
ContourLevelAnnotation **addContourLevelAnnotation** (ContourLevel level, Wcs wcs, Color color, int minLength) Adds a ContourLevel.
ContourAnnotation **addContourAnnotation** (Contour contour, int minLength, Wcs wcs, float width, Color color) Adds a Contour.
ArrayList **addPositionList** (PositionList positionList, Color color) Overlays the given source list on this Display, in the given color.
ArrayList **addPositionList** (PositionList positionList) Overlays the given source list on this Display, in the given color.
ArrayList **addPositionList** (PositionList positionList, Float1d sizes) Overlays the given source list on this Display, with the given sizes.
ArrayList **addPositionList** (PositionList positionList, Color color, Float1d sizes) Overlays the given source list.
ArrayList **addPositionListWcs** (PositionList positionList, Color color, Float1d sizes) Overlays the given source list on this Display.
ArrayList **addSourceFit** (SourceFittingProduct source, Color color) Visualises the given SourceFittingProduct on this Display.
ArrayList **addSourceFit** (SourceFittingProduct source) Visualises the given SourceFittingProduct on this Display.
ArrayList **showAxes** (boolean showAxis) Enables or disables the axes for the displayed image.

Methods
ImageAxis **getLeftaxis** Returns the left axis.
ImageAxis **getRightaxis** Returns the right axis.
ImageAxis **getBottomaxis** Returns the bottom axis.
ImageAxis **getTopaxis** Returns the top axis.
ArrayList **getAxes** Returns an array with the axes.
addAxis (String label, AxisConstants.Position orientation) Adds new axis.
addAxis (ImageAxis axis) Adds new axis.
deleteAxis (ImageAxis axis) Deletes the given axis.
setCenter (int row, int column) Sets the center of the image.
setCenter (double row, double column) Sets the center of the image.
setCenter (double row, double column, boolean update) Sets the center of the image.
getCenter Returns the center of the drawn image.
close Closes the display and frees the memory.
editColors Edit the colors using a popup.
editCutLevels Edit the cut levels using a popup.
String **getColortable** Returns the name of the color table.
double[] **getCutLevels** Returns the cut levels.
Image **getImage** Returns the displayed image.
Image **getOriginalImage** Returns the displayed image.
RgbImage **getRgbImage** Returns the displayed image.
double[] **getPixelCoordinates** (double c1, double c2)

Methods
Returns the image coordinates.
Number **getIntensity** (int row, int column) Returns the intensity of the pixel.
Number **getIntensityFromWorldCoordinates** (double c1, double c2) Returns the intensity of the pixel.
Unit **getUnit** Returns the unit of the image
float **getZoomFactor** Returns the used zoom factor.
float **getZoomFactorX** Returns the used X zoom factor.
float **getZoomFactorY** Returns the used Y zoom factor.
printDialog Opens a printer dialog.
getPrintControl Returns the printControl.
createPrintJob Creates a print job.
PrinterJob **getPrintJob** Returns the printer job.
setPrintJob (PrinterJob job) Set a given printer job.
print (HashPrintRequestAttributeSet attributes) Print the displayed image to a ps file.
save Pops up a dialog to save your image.
save (String filename) Saves the whole image, including annotations.
saveAsJPG (String filename) Saves the display as a jpg file.
saveAsPNG (String filename) Saves the display as a png file.
saveAsEPS (String filename) Saves the display as a eps file.
saveAsPDF (String filename) Saves the display as a pdf file.
saveCurrentViewAsJPG (String filename) Saves the display as a jpg file.
saveCurrentViewAsPNG (String filename) Saves the display as a png file.

Methods
saveCurrentViewAsEPS (String filename) Saves the display as a eps file.
saveCurrentViewAsPDF (String filename) Saves the display as a pdf file.
setTitle (String text) Sets Sets the title of the image.
saveCurrentView (String filename) Saves the current view of the image.
setColortable (String colortableName) Sets the color table of the displayed image.
setColortable (String colortableName, String intensityName) Sets the colortable of the displayed image.
setColortable (String colortableName, String intensityName, String scaleName) Sets the colortable of the displayed image
setCutLevelsPercentage (double percent) Sets the cut level of the displayed image.
setCutLevelsMin (double min) Sets the minimum cut levels.
setCutLevelsMax (double max) Sets the maximum cut levels.
setCutLevelsMinMax (double min, double max) Sets the cut level of the displayed image.
calcCutLevels Sets the cut level of the displayed image.
setCutLevels (double[] minmax) Sets the cut level of the displayed image.
setUnit (Unit<?> u) Sets the unit of the displayed image.
setZoomFactor (float zoomFactor) Sets the zoomfactor of the displayed image.
setZoomFactorX (float zoomFactorX) Sets the X zoomfactor of the displayed image.
setZoomFactorY (float zoomFactorY) Sets the Y zoomfactor of the displayed image.
zoom (double row, double column, float zoomFactor) Zooms the image.
zoomWorldCoordinates (double ra, double decl, float zoomFactor) Zooms the image.
zoomIn Zooms the image in.
zoomOut

Methods
Zoom the image out.
zoomFit Zoom the image.
flipYAxis Flips the y axis of the displayed image.
flipNoAxis Do not flip the x or y axis of the displayed image.
flipXAxis Flips the x axis of the displayed image.
flipXYAxis Flips the xy axis of the displayed image.
flipYXAxis Flips the yx axis of the displayed image.
setFlipYAxis `(boolean flipAxis)` Sets whether the current image should be flipped.
setFlipXAxis `(boolean flipXAxis)` Sets whether the current image should be flipped.
setFlipXYAxis `(boolean flipXYAxis)` Sets whether the current image should be flipped.
setFlipYXAxis `(boolean flipYXAxis)` Sets whether the current image should be flipped.
getFlipYAxis Returns whether the Y axis is flipped.
getFlipXAxis Returns whether the X axis is flipped.
getFlipXYAxis Returns whether the image is flipped.
getFlipYXAxis Returns whether the image is flipped.
isFlipped Returns whether the current layer is flipped.
isFlippedX Returns whether the current layer is flipped.
isFlippedXY Returns whether the current layer is xy-flipped.
isFlippedYX Returns whether the current layer is yx-flipped.
setDepthAxis `(int depthAxis)` Sets the depth axis for cubes.
getDepthAxis Returns the depth axis for cubes.

API Details

Constructors

Display

The standard constructor for Display.

This constructor shows an empty display window.

Display (boolean useAsComponent, boolean imageVisible)

A constructor to display the image in the background.

A constructor where you have the possibility to display the image in the background (which means the image will not be shown) or to use the display as a component to include in a gui in HIPE.

Arguments

`boolean` **`useAsComponent`** `[INPUT, MANDATORY, default=no default value]`

True if the Display should be component based, false otherwise.

`boolean` **`imageVisible`** `[INPUT, MANDATORY, default=no default value]`

True if the image should be visible, false otherwise.

Display (boolean useAsComponent)

A constructor to use Display as a component.

A constructor where you have the possibility to use the display as a component to include in a gui in HIPE.

Argument

`boolean` **`useAsComponent`** `[INPUT, MANDATORY, default=no default value]`

True if the Display should be component based, false otherwise.

Example

Example how to use Display as a component

```
# Using swing components, should be run in EDT (swing) thread!
from javax.swing import JFrame
from java.awt import BorderLayout
dc = Display(True)
frame = JFrame()
frame.setTitle("Image display using Components")
frame.setSize(800, 600)
# The main component, with the image
component = dc.getComponent()
# The zoom, panner, status and colorbar components
zoomComponent = dc.getZoomComponent()
pannerComponent = dc.getPannerComponent()
statusComponent = dc.getStatusPanelComponent()
colorbarComponent = dc.getColorBarComponent()
# Adding the components to the frame
frame.getContentPane().add(component, BorderLayout.CENTER)
frame.getContentPane().add(statusComponent, BorderLayout.NORTH)
frame.getContentPane().add(colorbarComponent, BorderLayout.SOUTH)
frame.show()
dc.setImage(image)
```

Display (Image imds)

A constructor which displays an Image.

Display (Image imds)

A constructor which immediately displays the given Image.

Argument

`Image` **`imds`** `[INPUT, MANDATORY, default=no default value]`

The Image that should be shown.

Display (Cube cube)

A constructor which displays a Cube.

A constructor which immediately displays the given Cube.

Argument

`Cube` **`cube`** `[INPUT, MANDATORY, default=no default value]`

The Cube that should be shown.

Display (RgbImage imds)

A constructor which displays an RgbImage.

A constructor which immediately displays the given RgbImage.

Argument

`RgbImage` **`imds`** `[INPUT, MANDATORY, default=no default value]`

The RgbImage that should be shown.

Display (Image imds, pabooleanramType imageVisible)

A Display constructor for an Image.

A constructor which immediately displays the given Image. It is possible to still hide the Image.

Arguments

`Image` **`imds`** `[INPUT, MANDATORY, default=no default value]`

The Image that should be shown.

`pabooleanramType` **`imageVisible`** `[INPUT, MANDATORY, default=no default value]`

True if the image should be visible, false otherwise.

Display (Cube cube, boolean imageVisible)

A Display constructor for a Cube.

A constructor which immediately displays the given Cube. It is possible to still hide the Cube.

Arguments

`Cube` **`cube`** `[INPUT, MANDATORY, default=no default value]`

The Cube that should be shown.

`boolean` **`imageVisible`** `[INPUT, MANDATORY, default=no default value]`

True if the image should be visible, false otherwise.

Display (Array2dData data)

A constructor to display a Numeric2d.

A constructor which immediately displays the given numeric2d

Display (Array2dData data)

Argument

`Array2dData` **`data`** `[INPUT, MANDATORY, default=no default value]`

An implementation of the herschel.ia.numeric.Array2dData interface. This should be a Bool2d, Float2d, Int2d, Long2d or a Double2d.

Display (Array2dData data, boolean imageVisible)

A Display constructor for Array2dData.

A constructor which displays the given numeric2d. It is possible to still hide the image.

Arguments

`Array2dData` **`data`** `[INPUT, MANDATORY, default=no default value]`

An implementation of the herschel.ia.numeric.Array2dData interface. This should be a Bool2d, Float2d, Int2d, Long2d or a Double2d.

`boolean` **`imageVisible`** `[INPUT, MANDATORY, default=no default value]`

True if the image should be visible, false otherwise.

Display (Array3dData data)

A constructor to display a numeric3d.

A constructor which immediately displays the given numeric3d.

Argument

`Array3dData` **`data`** `[INPUT, MANDATORY, default=no default value]`

An implementation of the herschel.ia.numeric.Array3dData interface. This should be a Bool3d, Float3d, Int3d, Long3d or a Double3d.

Display (Array3dData data, boolean imageVisible)

A display constructor for 3d data.

A constructor which immediately displays the given numeric3d.

Arguments

`Array3dData` **`data`** `[INPUT, MANDATORY, default=no default value]`

An implementation of the herschel.ia.numeric.Array3dData interface. This should be a Bool3d, Float3d, Int3d, Long3d or a Double3d.

`boolean` **`imageVisible`** `[INPUT, MANDATORY, default=no default value]`

True if the image should be visible, false otherwise.

Methods

setSize (int x, int y)

Adapt the dimensions of the frame.

Arguments

`int` **`x`** `[INPUT, MANDATORY, default=no default value]`

The x size of the frame.

`int` **`y`** `[INPUT, MANDATORY, default=no default value]`

The y size of the frame.

getSize

Returns the dimensions of the frame.

***int* getLayerShown**

Returns the number of the layer which is shown.

Return

int

The number of the layer which is shown.

setVisible (boolean imageVisible)

Toggles the display visible or invisible.

True to make the image visible, False to hide the image.

Argument

`boolean` **`imageVisible`** `[INPUT, MANDATORY, default=no default value]`

True if the image should be visible, false otherwise.

***JPanel* getComposedComponent**

Returns a panel with all components of this Display on it.

Return

JPanel

Returns a panel with all component of this Display on it.

***ImageColorbar* getColorBarComponent**

Returns the color bar component.

Returns a JComponent that contains the color bar. The color bar can be used to create your own component based Display.

Return

ImageColorbar

The ImageColorbar to use in you own components.

***DivaMainImageDisplay* getComponent**

Returns the component where the image is displayed.

This can be used to make your own GUI's with an image.

Return

DivaMainImageDisplay

The component to use in your own GUI's

***ImagePanner* getPannerComponent**

Returns the panner component with the overview of the image.

Returns the component which contains the overview of the image (100x100). This can be used to make your own GUI's with an image.

Return

ImagePanner

***ImagePanner* getPannerComponent**

The component with the overview to use in your own GUI's

***StatusPanel* getStatusPanelComponent**

Returns the component which contains the status bar of the image.

The status bar exists of zoom buttons, magnification, pixel coordinates, pixel intensity and sky coordinates. This can be used to make your own GUI's with an image.

Return

StatusPanel

The component with the status panel to use in your own GUI's

setStatusPanelComponent (StatusPanel imageDisplayStatusPanel)

Set a new status panel for the display.

Attaches a new status panel to the display.

Argument

`StatusPanel` **`imageDisplayStatusPanel`** `[INPUT, MANDATORY, default=no default value]`

The Jcomponent including the status bar (with Zoom buttons, magnification, Pixel coordinates, intensity and Sky coordinates), that the display will use.

***ImageZoom* getZoomComponent**

Returns the component with the zoomed view.

The zoomed view has a zoom factor of 4. This component can be used to make your own GUI's with an image.

Return

ImageZoom

The component with the zoomed view to use in your own GUI's

setImage (Array2dData imageData)

Set a new numeric2d image.

Sets a new image on the display. A Bool2d, Int2d, Double2d, ... can be used.

Argument

`Array2dData` **`imageData`** `[INPUT, MANDATORY, default=no default value]`

The numeric2D datatype to show.

setImage (Array3dData imageData)

Sets a new numeric3d image.

Shows a new image on the display. A Bool3d, Int3d, Double3d, ... can be used. The first image is shown, the other images are used as extra layers.

Argument

`Array3dData` **`imageData`** `[INPUT, MANDATORY, default=no default value]`

The numeric3D datatype to show.

setImage (Image imds)

Sets a new Image to Display.

Shows a new image on the display. The world-coordinate system of the Image is used.

Argument

`Image` **`imds`** `[INPUT, MANDATORY, default=no default value]`

The Image to show.

setImage (Cube cube)

Sets a new cube to display.

A Cube is shown on the display. The world-coordinate system of the Cube is used.

Argument

`Cube` **`cube`** `[INPUT, MANDATORY, default=no default value]`

The Cube to show.

setImage (RgbImage imds)

Sets a new RgbImage to Display.

Shows a new RgbImage on the display.

Argument

`RgbImage` **`imds`** `[INPUT, MANDATORY, default=no default value]`

The RgbImage to show.

addLayer (Layer layer)

Adds a new layer to the display.

Adds a new layer to the display. A Layer must be constructed before.

Argument

`Layer` **`layer`** `[INPUT, MANDATORY, default=no default value]`

The layer to add.

addLayer (Array2dData imageData)

Adds a new layer with numeric2d data.

Adds a new layer to the display. A Bool2d, Int2d, Double2d, ... can be used.

Argument

`Array2dData` **`imageData`** `[INPUT, MANDATORY, default=no default value]`

The numeric2d layer to add.

addLayer (Array3dData imageData)

Adds a new layer with numeric3d data.

Adds new layers to the display. A Bool3d, Int3d, Double3d, ... can be used.

Argument

`Array3dData` **`imageData`** `[INPUT, MANDATORY, default=no default value]`

The numeric3d describing the layers to add.

addLayer (Image imds)

Adds a new layer with an Image to the display.

Adds a new layer to the display. An Image is added as extra layer of the display.

Argument

`Image` **`imds`** `[INPUT, MANDATORY, default=no default value]`

The image to add to the layer.

addLayer (RgbImage imds)

Adds a new layer with an RgbImage to the display.

Adds a new layer to the display. An RgbImage is added as extra layer of the display.

Argument

`RgbImage` **`imds`** `[INPUT, MANDATORY, default=no default value]`

The rgb image to add to the layer.

addLayer (Cube cube)

Adds a new layer with a Cube to the display.

Adds a new layer to the display. A Cube is added as extra layer of the display.

Argument

`Cube` **`cube`** `[INPUT, MANDATORY, default=no default value]`

The Cube to add to the displayed image.

addLayer (SkyMask skyMask)

Adds a new layer with a SkyMask to the display.

A SkyMask is added as extra layer of the display.

Argument

`SkyMask` **`skyMask`** `[INPUT, MANDATORY, default=no default value]`

The SkyMask to add to the displayed image.

Layer **getLayer**

Returns the shown layer.

Returns the Layer that is shown.

Return

Layer

The layer that is shown at the moment.

Layer **getLayer (int i)**

Returns the Layer object.

The Layer with the given index is returned.

Argument

`int` **`i`** `[INPUT, MANDATORY, default=no default value]`

The Layer to return.

Return

Layer

Layer getLayer (int i)

The layer with the given index.

showLayer (int i)

Shows a chosen layer.

Shows the Layer with the given index. The Layer with the given index is shown.

Argument

`int` **`i`** `[INPUT, MANDATORY, default=no default value]`

The layer to show.

removeLayer

Removes the current layer.

Removes the Layer that is shown.

removeLayer (int i)

Removes the given Layer.

Argument

`int` **`i`** `[INPUT, MANDATORY, default=no default value]`

The layer to remove.

updateImage (Array2dData imageData)

Updates the layer with numeric2d data.

Updates the layer of the image that is displayed. A numeric2d will be used as input for the new layer. The annotations, cut levels and zoom factor are kept.

Argument

`Array2dData` **`imageData`** `[INPUT, MANDATORY, default=no default value]`

The numeric2D data type to show.

updateImage (Image imds)

Updates the layer with an Image.

Updates the layer of the image that is displayed. An Image will be used as input for the new layer. The annotations, cut levels and zoom factor are kept.

Argument

`Image` **`imds`** `[INPUT, MANDATORY, default=no default value]`

The Image to show.

updateImage (RgbImage imds)

Updates the layer with an Image.

Updates the layer of the image that is displayed. An RgbImage will be used as input for the new layer. The annotations, cut levels and zoom factor are kept.

Argument

`RgbImage` **`imds`** `[INPUT, MANDATORY, default=no default value]`

The RgbImage to show.

updateImage (Array3dData imageData)

Updates the layer with numeric3d data.

Updates the layer of the image that is displayed. A numeric3d will be used as input for the new layer. The annotations, cut levels and zoom factor are kept.

Argument

`Array3dData` **`imageData`** `[INPUT, MANDATORY, default=no default value]`

The numeric3D data type to show.

updateImage (Cube imds)

Updates the layer with a cube.

Updates the layer of the image that is displayed. A cube will be used as input for the new layer. The annotations, cut levels and zoom factor are kept.

Argument

`Cube` **`imds`** `[INPUT, MANDATORY, default=no default value]`

The Cube to show.

setBackground (Color bgColor)

Sets the background color.

Changes the background of the image. The background can be black or white.

Argument

`Color` **`bgColor`** `[INPUT, MANDATORY, default=no default value]`

The new background Color (Black or White).

getBackground

Returns the background color.

Returns the background of the image. The background can be black or white.

annotationToolbox

Fires up the annotation toolbox.

Returns the AnnotationToolbox for the display. The AnnotationToolbox lets the user put annotation on the image, like ellipses, rectangles, text, ...

skyMaskToolbox

Fires up the SkyMask toolbox.

Returns the SkyMaskToolbox for the display. The SkyMaskToolbox lets the user create a SkyMask by adding annotations to the image, like ellipses, rectangles, ...

CanvasFigure **addAnnotation (String text, double row, double column)**

Adds a text annotation.

A text annotation will be placed, starting at the given pixel coordinates. To change the color of the annotation, you can use the method setFillPaint(java.awt.Color) on the returned annotation.

Arguments

`String` **`text`** `[INPUT, MANDATORY, default=no default value]`

CanvasFigure **addAnnotation (String text, double row, double column)**

The text to add to the image.

`double` **`row`** `[INPUT, MANDATORY, default=no default value]`

The row where the annotation should be placed.

`double` **`column`** `[INPUT, MANDATORY, default=no default value]`

The column where the annotation should be placed.

Return

CanvasFigure

The annotation as a CanvasFigure.

TextAnnotation **addTextAnnotation (String text, double row, double column, Font font, Color fontColor)**

Adds a text annotation at given pixel coordinates.

A text annotation will be placed, starting at the given pixel coordinates.

Arguments

`String` **`text`** `[INPUT, MANDATORY, default=no default value]`

The text to add to the image.

`double` **`row`** `[INPUT, MANDATORY, default=no default value]`

The row where the annotation should be placed.

`double` **`column`** `[INPUT, MANDATORY, default=no default value]`

The column where the annotation should be placed.

`Font` **`font`** `[INPUT, MANDATORY, default=no default value]`

The font to use.

`Color` **`fontColor`** `[INPUT, MANDATORY, default=no default value]`

The color for the font.

Return

TextAnnotation

The annotation as a TextAnnotation

CanvasFigure **addAnnotationWorldCoordinates (String text, double ra, double decl)**

Adds a text annotation at given sky coordinates.

A text annotation will be placed, starting at the given sky coordinates. To change the color of the annotation, you can use the method setFillPaint(java.awt.Color) on the returned annotation.

Arguments

`String` **`text`** `[INPUT, MANDATORY, default=no default value]`

The text to add to the image.

`double` **`ra`** `[INPUT, MANDATORY, default=no default value]`

The right ascension where the annotation should be placed.

`double` **`decl`** `[INPUT, MANDATORY, default=no default value]`

The declination where the annotation should be placed.

Return

CanvasFigure

CanvasFigure **addAnnotationWorldCoordinates (String text, double ra, double decl)**

The annotation as a CanvasFigure.

TextAnnotation **addTextAnnotationWorldCoordinates (String text, double ra, double decl, Font font, Color fontColor)**

Adds a text annotation at given sky coordinates.

A text annotation will be placed, starting at the given sky coordinates. To change the color of the annotation, you can use the method setFillPaint(java.awt.Color) on the returned annotation.

Arguments

`String` **text** `[INPUT, MANDATORY, default=no default value]`

The text to add to the image.

`double` **ra** `[INPUT, MANDATORY, default=no default value]`

The right ascension where the annotation should be placed.

`double` **decl** `[INPUT, MANDATORY, default=no default value]`

The declination where the annotation should be placed.

`Font` **font** `[INPUT, MANDATORY, default=no default value]`

The font to use.

`Color` **fontColor** `[INPUT, MANDATORY, default=no default value]`

The color for the font.

Return

TextAnnotation

The annotation as a TextAnnotation.

CanvasFigure **addGreekAnnotation (String text, double row, double column)**

Adds a Greek annotation at the given pixel coordinates.

A greek text annotation will be placed, starting at the given pixel coordinates. All ascii text will be converted to greek characters : a becomes alpha, b becomes beta, ... To change the color of the annotation, you can use the method setFillPaint(java.awt.Color) on the returned annotation.

Arguments

`String` **text** `[INPUT, MANDATORY, default=no default value]`

The text to add to the image (a becomes alpha, b beta, ...).

`double` **row** `[INPUT, MANDATORY, default=no default value]`

The row where the annotation should be placed.

`double` **column** `[INPUT, MANDATORY, default=no default value]`

The column where the annotation should be placed.

Return

CanvasFigure

The annotation as a CanvasFigure.

TextAnnotation **addGreekTextAnnotation (String text, double row, double column)**

Adds a Greek annotation at the given pixel coordinates.

***TextAnnotation* addGreekTextAnnotation (String text, double row, double column)**

A greek text annotation will be placed, starting at the given pixel coordinates. All ascii text will be converted to greek characters : a becomes alpha, b becomes beta, ... To change the color of the annotation, you can use the method setFillPaint(java.awt.Color) on the returned annotation.

Arguments

`String` **text** [INPUT, MANDATORY, default=no default value]

The text to add to the image (a becomes alpha, b beta, ...).

`double` **row** [INPUT, MANDATORY, default=no default value]

The row where the annotation should be placed.

`double` **column** [INPUT, MANDATORY, default=no default value]

The column where the annotation should be placed.

Return

TextAnnotation

The annotation as a TextAnnotation.

***CanvasFigure* addGreekAnnotationWorldCoordinates (String text, double ra, double decl)**

Adds a Greek annotation at the given world coordinates.

A greek text annotation will be placed, starting at the given world coordinates. All ascii text will be converted to greek characters : a becomes alpha, b becomes beta, ... To change the color of the annotation, you can use the method setFillPaint(java.awt.Color) on the returned annotation.

Arguments

`String` **text** [INPUT, MANDATORY, default=no default value]

The text to add to the image.

`double` **ra** [INPUT, MANDATORY, default=no default value]

The right ascension where the annotation should be placed.

`double` **decl** [INPUT, MANDATORY, default=no default value]

The declination where the annotation should be placed.

Return

CanvasFigure

The annotation as a CanvasFigure.

***TextAnnotation* addGreekTextAnnotationWorldCoordinates (String text, double ra, double decl)**

Adds a Greek annotation at the given world coordinates.

A greek text annotation will be placed, starting at the given world coordinates. All ascii text will be converted to greek characters : a becomes alpha, b becomes beta, ... To change the color of the annotation, you can use the method setFillPaint(java.awt.Color) on the returned annotation.

Arguments

`String` **text** [INPUT, MANDATORY, default=no default value]

The text to add to the image.

`double` **ra** [INPUT, MANDATORY, default=no default value]

The right ascension where the annotation should be placed.

`double` **decl** [INPUT, MANDATORY, default=no default value]

TextAnnotation **addGreekTextAnnotationWorldCoordinates (String text, double ra, double decl)**

The declination where the annotation should be placed.

Return

TextAnnotation

The annotation as a TextAnnotation.

Font **getAnnotationFont**

Returns the default font for annotations.

Return

Font

The font used for the annotations.

Font **getAnnotationFont (double row, double column)**

Returns the font for the specified annotation.

Returns the font used for the annotation that begins at the given pixel coordinates.

Arguments

`double` **`row`** `[INPUT, MANDATORY, default=no default value]`

The row.

`double` **`column`** `[INPUT, MANDATORY, default=no default value]`

The column.

Return

Font

The font used for the specified annotation.

Font **getAnnotationFontWorldCoordinates (double ra, double dec)**

Returns the font for the specified annotation.

Returns the font used for the annotation that begins at the given world coordinates.

Arguments

`double` **`ra`** `[INPUT, MANDATORY, default=no default value]`

The right ascension where the annotation should be placed.

`double` **`dec`** `[INPUT, MANDATORY, default=no default value]`

The declination where the annotation should be placed.

Return

Font

The font used for the specified annotation.

Color **getAnnotationFontColor**

Returns the default color for annotations.

Return

Color

The default color used for the text annotations.

Color getAnnotationFontColor (double row, double column)

Returns the color of the specified annotation.

Returns the font color used for the annotation that begins at the given pixel coordinates.

Arguments

`double` **row** `[INPUT, MANDATORY, default=no default value]`

The row.

`double` **column** `[INPUT, MANDATORY, default=no default value]`

The column.

Return

Color

The color used for the specified text annotation.

Color getAnnotationFontColorWorldCoordinates (double ra, double dec)

Returns the color of the specified annotation.

Returns the color used for the annotation that begins at the given world coordinates.

Arguments

`double` **ra** `[INPUT, MANDATORY, default=no default value]`

The right ascension.

`double` **dec** `[INPUT, MANDATORY, default=no default value]`

The declination.

Return

Color

The color used for the specified text annotation.

removeAnnotation (double row, double column)

Removes the specified annotation.

Removes the annotation at the given pixel coordinates.

Arguments

`double` **row** `[INPUT, MANDATORY, default=no default value]`

The row of the annotation object to remove.

`double` **column** `[INPUT, MANDATORY, default=no default value]`

The column of the annotation object to remove.

removeAnnotationWorldCoordinates (double ra, double dec)

Remove the specified annotation.

Removes the annotation at the given world coordinates.

Arguments

`double` **ra** `[INPUT, MANDATORY, default=no default value]`

The right ascension of the annotation object to remove.

`double` **dec** `[INPUT, MANDATORY, default=no default value]`

The declination of the annotation object to remove.

removeAnnotation (CanvasFigure fig)

Removes the specified annotation.

Argument

CanvasFigure **fig** [INPUT, MANDATORY, default=no default value]

The annotation to remove.

removeAnnotations

Removes all annotations.

Removes all the annotations from the display.

setAnnotationFont (Font f)

Changes the font of all Annotations.

Changes the font of all annotations that are visible on the Display.

Argument

Font **f** [INPUT, MANDATORY, default=no default value]

The new font for all the annotations.

setAnnotationFont (double row, double column, Font f)

Changes the font of the specified annotations.

Changes the font of the annotations which are located at the given pixel coordinates.

Arguments

double **row** [INPUT, MANDATORY, default=no default value]

The row.

double **column** [INPUT, MANDATORY, default=no default value]

The column.

Font **f** [INPUT, MANDATORY, default=no default value]

The new font for the annotations.

setAnnotationFontWorldCoordinates (double ra, double dec, Font f)

Changes the font of the annotations at given sky coordinates.

Changes the font of the annotations which are located at the given world coordinates.

Arguments

double **ra** [INPUT, MANDATORY, default=no default value]

The right ascension.

double **dec** [INPUT, MANDATORY, default=no default value]

The declination.

Font **f** [INPUT, MANDATORY, default=no default value]

The new font for the annotations.

setAnnotationFont (int size)

Changes the font size of all annotations.

Argument

int **size** [INPUT, MANDATORY, default=no default value]

The new font size for the annotations.

setAnnotationFont (double row, double column, int size)

Changes the font size of the specified annotations.

Changes the font size of the annotations which are located at the given pixel coordinates.

Arguments

`double` **row** `[INPUT, MANDATORY, default=no default value]`

The row.

`double` **column** `[INPUT, MANDATORY, default=no default value]`

The column.

`int` **size** `[INPUT, MANDATORY, default=no default value]`

The new size of the font.

setAnnotationFontWorldCoordinates (double ra, double dec, int size)

Changes the font size of the specified annotations.

Changes the font size of the annotations which are located at the given world coordinates.

Arguments

`double` **ra** `[INPUT, MANDATORY, default=no default value]`

The right ascension.

`double` **dec** `[INPUT, MANDATORY, default=no default value]`

The declination.

`int` **size** `[INPUT, MANDATORY, default=no default value]`

The new fontsize for the annotations.

setAnnotationFontColor (Color color)

Sets the default color for annotations.

Argument

`Color` **color** `[INPUT, MANDATORY, default=no default value]`

The default color for annotations (java.awt.Color).

setAnnotationFontColor (double row, double column, Color color)

Sets the font color for the specified annotation.

Changes the font color of the annotations which are located at the given pixel coordinates.

Arguments

`double` **row** `[INPUT, MANDATORY, default=no default value]`

The row.

`double` **column** `[INPUT, MANDATORY, default=no default value]`

The column.

`Color` **color** `[INPUT, MANDATORY, default=no default value]`

The default color for annotations (java.awt.Color).

setAnnotationFontColorWorldCoordinates (double ra, double dec, Color color)

Sets the font color for the specified annotation.

Changes the font color of the annotations which are located at the given world coordinates.

setAnnotationFontColorWorldCoordinates (double ra, double dec, Color color)

Arguments

`double` **`ra`** `[INPUT, MANDATORY, default=no default value]`

The right ascension.

`double` **`dec`** `[INPUT, MANDATORY, default=no default value]`

The declination.

`Color` **`color`** `[INPUT, MANDATORY, default=no default value]`

The default color for annotations (java.awt.Color).

EllipseAnnotation **addEllipseAnnotation (double row, double column, double w, double h, float lineWidth, Color color, double positionAngle, boolean fixed)**

Adds an ellipse.

Adds an ellipse to the image at a given place, with a given dimension, line width, color and position angle.

Arguments

`double` **`row`** `[INPUT, MANDATORY, default=no default value ]`

The row of the center of the ellipse.

`double` **`column`** `[INPUT, MANDATORY, default=no default value]`

The column of the center of the ellipse.

`double` **`w`** `[INPUT, MANDATORY, default=no default value]`

The width of the ellipse in pixels.

`double` **`h`** `[INPUT, MANDATORY, default=no default value]`

The heigth of the ellipse in pixels.

`float` **`lineWidth`** `[INPUT, MANDATORY, default=no default value]`

The line width of the ellipse.

`Color` **`color`** `[INPUT, MANDATORY, default=no default value]`

The color of the ellipse.

`double` **`positionAngle`** `[INPUT, MANDATORY, default=no default value]`

The position Angle in radians (positive is clockwise). The angle between the major axis and the horizontal.

`boolean` **`fixed`** `[INPUT, MANDATORY, default=no default value]`

Annotation is not moveable when this parameter is true.

Return

EllipseAnnotation

The annotation as EllipseAnnotation.

EllipseAnnotation **addCircleAnnotation (double row, double column, double radius, float lineWidth, Color color)**

Adds a circle.

Adds a circle to the image at a given place, with a given dimension, line width and color.

Arguments

`double` **`row`** `[INPUT, MANDATORY, default=no default value ]`

***EllipseAnnotation* addCircleAnnotation (double row, double column, double radius, float lineWidth, Color color)**

The row of the center of the ellipse.

`double` **`column`** `[INPUT, MANDATORY, default=no default value]`

The column of the center of the ellipse.

`double` **`radius`** `[INPUT, MANDATORY, default=no default value]`

The radius of the circle in rows.

`float` **`lineWidth`** `[INPUT, MANDATORY, default=no default value]`

The line width of the ellipse.

`Color` **`color`** `[INPUT, MANDATORY, default=no default value]`

The color of the ellipse.

Return

EllipseAnnotation

The annotation as EllipseAnnotation

***LineAnnotation* addLineAnnotation (double r1, double c1, double r2, double c2, float lineWidth, Color color, boolean fixed)**

Adds a line.

Adds an line to the image at a given place, with a given line width and color.

Arguments

`double` **`r1`** `[INPUT, MANDATORY, default=no default value ]`

The row of the point where the line should start.

`double` **`c1`** `[INPUT, MANDATORY, default=no default value]`

The column of the point where the line should start.

`double` **`r2`** `[INPUT, MANDATORY, default=no default value ]`

The row of the point where the line should stop.

`double` **`c2`** `[INPUT, MANDATORY, default=no default value]`

The column of the point where the line should stop.

`float` **`lineWidth`** `[INPUT, MANDATORY, default=no default value]`

The line width of the ellipse.

`Color` **`color`** `[INPUT, MANDATORY, default=no default value]`

The color of the ellipse.

`boolean` **`fixed`** `[INPUT, MANDATORY, default=no default value]`

True to make the annotation fixed.

Return

LineAnnotation

The annotation as LineAnnotation.

***CrossAnnotation* addCrossAnnotation (double row, double column, double size, float width, Color color)**

Adds a cross.

Adds a cross to the image at a given place, with a given size, line width and color.

Arguments

***CrossAnnotation* addCrossAnnotation (double row, double column, double size, float width, Color color)**

`double` **`row`** `[INPUT, MANDATORY, default=no default value ]`

The row of the center of the cross.

`double` **`column`** `[INPUT, MANDATORY, default=no default value ]`

The column of the center of the cross.

`double` **`size`** `[INPUT, MANDATORY, default=no default value ]`

The size (width and height) of the cross in pixels.

`float` **`width`** `[INPUT, MANDATORY, default=no default value ]`

The line width of the cross.

`Color` **`color`** `[INPUT, MANDATORY, default=no default value ]`

The color of the cross.

Return

CrossAnnotation

The cross as a CrossAnnotation.

***PolygonAnnotation* addPolygonAnnotation (double[] coords, float lineWidth, Color color, boolean fixed)**

Adds a polygon.

Adds a polygon to the image at a given place, with a given line width and color. The coordinates should be given as [row1, column1, row2, column2, ...].

Arguments

`double[]` **`coords`** `[INPUT, MANDATORY, default=no default value ]`

The coordinates of the points which connect to a polygon. First the row and column of the first point, then the second,... Example : [row1, column1, row2, column2]

`float` **`lineWidth`** `[INPUT, MANDATORY, default=no default value ]`

The line width of the polygon.

`Color` **`color`** `[INPUT, MANDATORY, default=no default value ]`

The color of the polygon.

`boolean` **`fixed`** `[INPUT, MANDATORY, default=no default value ]`

If true, the polygon can not be moved on the image.

Return

PolygonAnnotation

The annotation as CanvasFigure

***PolylineAnnotation* addPolylineAnnotation (double[] coords, float lineWidth, Color color, boolean fixed)**

Adds a polyline.

Adds a polyline to the image at a given place, with a given line width and color. The coordinates should be given as [row1, column1, row2, column2, ...].

Arguments

`double[]` **`coords`** `[INPUT, MANDATORY, default=no default value ]`

The coordinates (in pixels) of the points which connect to a polyline. First the row and column of the first point, then the second, ... Example : [row1, column1, row2, column2]

***PolylineAnnotation* addPolylineAnnotation (double[] coords, float lineWidth, Color color, boolean fixed)**

`float` **`lineWidth`** `[INPUT, MANDATORY, default=no default value ]`

The line width of the polyline.

`Color` **`color`** `[INPUT, MANDATORY, default=no default value ]`

The color of the polyline.

`boolean` **`fixed`** `[INPUT, MANDATORY, default=no default value ]`

If true, the polyline can not be moved on the image.

Return

PolylineAnnotation

The annotation as CanvasFigure

***RectangleAnnotation* addRectangleAnnotation (double row, double column, double w, double h, float lineWidth, Color color, boolean fixed)**

Adds a rectangle.

Adds an rectangle to the image at a given place, with a given dimension, line width and color.

Arguments

`double` **`row`** `[INPUT, MANDATORY, default=no default value ]`

The minimum row of the rectangle.

`double` **`column`** `[INPUT, MANDATORY, default=no default value ]`

The minimum column of the rectangle.

`double` **`w`** `[INPUT, MANDATORY, default=no default value ]`

The width of the rectangle in columns.

`double` **`h`** `[INPUT, MANDATORY, default=no default value ]`

The heigth of the rectangle in rows.

`float` **`lineWidth`** `[INPUT, MANDATORY, default=no default value ]`

The line width of the rectangle.

`Color` **`color`** `[INPUT, MANDATORY, default=no default value ]`

The color of the rectangle.

`boolean` **`fixed`** `[INPUT, MANDATORY, default=no default value ]`

True to make the annotation fixed

Return

RectangleAnnotation

The annotation as CanvasFigure

addCompassAnnotation (double row, double column, double size, float width, Color color)

Adds a compass with the north and east directions.

Arguments

`double` **`row`** `[INPUT, MANDATORY, default=no default value ]`

The row where to paint the compass.

`double` **`column`** `[INPUT, MANDATORY, default=no default value ]`

The column where to paint the compass.

addCompassAnnotation (double row, double column, double size, float width, Color color)

`double` **size** `[INPUT, MANDATORY, default=no default value ]`

The size of the compass.

`float` **width** `[INPUT, MANDATORY, default=no default value ]`

The line width of the compass.

`Color` **color** `[INPUT, MANDATORY, default=no default value ]`

The color of the compass.

addArcSecsAnnotation (double arcsecs, double row, double column, float lineWidth, Color color)

Adds a line which of a requested number of arcseconds to the image.

Arguments

`double` **arcsecs** `[INPUT, MANDATORY, default=no default value ]`

The number of arcseconds

`double` **row** `[INPUT, MANDATORY, default=no default value ]`

The row where to paint the line.

`double` **column** `[INPUT, MANDATORY, default=no default value ]`

The column where to paint the line.

`float` **lineWidth** `[INPUT, MANDATORY, default=no default value ]`

The line width of the line.

`Color` **color** `[INPUT, MANDATORY, default=no default value ]`

The color of the line.

ImageContourAnnotation **addImageContourAnnotation (Color imageContour, Color colors, Color minLength)**

Adds an ImageContour.

Draws the Contours of the given ImageContour in the given Color on the image (if their length is at least the given minimum length) and returns the drawn Contours as an ArrayList of ImageFigures.

Arguments

`Color` **imageContour** `[INPUT, MANDATORY, default=no default value ]`

The ImageFigure for which the Contours must be drawn on the image.

`Color` **colors** `[INPUT, MANDATORY, default=no default value ]`

The Colors in which the Contours of the different ContourLevels of the given ImageContour must be drawn.

`Color` **minLength** `[INPUT, MANDATORY, default=no default value ]`

The minimum length for Contours to be drawn on the image.

Return

ImageContourAnnotation

The drawn Contours as an ImageContourAnnotation.

ImageContourAnnotation **addImageContourAnnotation (ImageContour imageContour)**

Adds an ImageContour.

***ImageContourAnnotation* addImageContourAnnotation (ImageContour imageContour)**

Draws the Contours of the given ImageContour on the image and returns the drawn Contours as an ImageContourAnnotation.

Argument

`ImageContour` **`imageContour`** `[INPUT, MANDATORY, default=no default value]`

The ImageContour for which the Contours must be drawn on the image.

Return

ImageContourAnnotation

Returns the drawn Contours as an ImageContourAnnotation.

***ImageContourAnnotation* addWcsImageContourAnnotation (ImageContour imageContour, ArrayList colors, int minLength)**

Add an ImageContour.

Draws the Contours of the given ImageContour in the given Color on the image based on the Wcs (if their length is at least the given minimum length) and returns the drawn Contours as an ArrayList of ImageFigures.

Arguments

`ImageContour` **`imageContour`** `[INPUT, MANDATORY, default=no default value ]`

The ImageFigure for which the Contours must be drawn on the image.

`ArrayList` **`colors`** `[INPUT, MANDATORY, default=no default value ]`

The Colors in which the Contours of the different ContourLevels of the given ImageContour must be drawn.

`int` **`minLength`** `[INPUT, MANDATORY, default=no default value ]`

The minimum length for Contours to be drawn on the image.

Return

ImageContourAnnotation

The drawn Contours as an ImageContourAnnotation.

***ContourLevelAnnotation* addContourLevelAnnotation (ContourLevel level, Wcs wcs, int minLength)**

Adds a ContourLevel.

Draws the Contours of the given ContourLevel on the image (if their length is at least the given minimum length) and returns the drawn Contours as aContourLevelAnnotation.

Arguments

`ContourLevel` **`level`** `[INPUT, MANDATORY, default=no default value ]`

The ContourLevel for which the Contours must be drawn.

`Wcs` **`wcs`** `[INPUT, MANDATORY, default=no default value ]`

The Wcs attached to the given ContourLevel.

`int` **`minLength`** `[INPUT, MANDATORY, default=no default value ]`

The minimum length for Contours to be drawn on the image.

Return

ContourLevelAnnotation

ContourLevelAnnotation **addContourLevelAnnotation (ContourLevel level, Wcs wcs, int minLength)**

Returns the drawn ContourLevel as a ContourLevelAnnotation.

ContourLevelAnnotation **addContourLevelAnnotation (ContourLevel level, Wcs wcs, Color color, int minLength)**

Adds a ContourLevel.

Draws the Contours of the given ContourLevel on the image (if their length is at least the given minimum length) and returns the drawn Contours as aContourLevelAnnotation.

Arguments

`ContourLevel` **`level`** `[INPUT, MANDATORY, default=no default value ]`

The ContourLevel for which the Contours must be drawn.

`Wcs` **`wcs`** `[INPUT, MANDATORY, default=no default value ]`

The Wcs attached to the given ContourLevel.

`Color` **`color`** `[INPUT, MANDATORY, default=no default value ]`

The color for the contours of this contourLevel.

`int` **`minLength`** `[INPUT, MANDATORY, default=no default value ]`

The minimum length for Contours to be drawn on the image.

Return

ContourLevelAnnotation

Returns the drawn ContourLevel as a ContourLevelAnnotation.

ContourAnnotation **addContourAnnotation (Contour contour, int minLength, Wcs wcs, float width, Color color)**

Adds a Contour.

Draws the given Contour on the image (if its length is at least the given minimum length) and returns the drawn Contour as an ImageFigure.

Arguments

`Contour` **`contour`** `[INPUT, MANDATORY, default=no default value ]`

The Contour to be drawn on the image.

`int` **`minLength`** `[INPUT, MANDATORY, default=no default value ]`

The minimum length for the given Contour to be drawn on the image.

`Wcs` **`wcs`** `[INPUT, MANDATORY, default=no default value ]`

The Wcs attached to the given Contour.

`float` **`width`** `[INPUT, MANDATORY, default=no default value ]`

The width of the contour in pixels.

`Color` **`color`** `[INPUT, MANDATORY, default=no default value ]`

The Color in which the given Contour must be drawn on the image.

Return

ContourAnnotation

Returns the drawn Contour as an ImageFigure.

ArrayList **addPositionList (PositionList positionList, Color color)**

Overlays the given source list on this Display, in the given color.

ArrayList addPositionList (PositionList positionList, Color color)

Arguments

`PositionList` **`positionList`** `[INPUT, MANDATORY, default=no default value ]`

The pisition list to overlay.

`Color` **`color`** `[INPUT, MANDATORY, default=no default value ]`

The color in which to overlay the source list.

Return

ArrayList

Returns a list of sources.

ArrayList addPositionList (PositionList positionList)

Overlays the given source list on this Display, in the given color.

Argument

`PositionList` **`positionList`** `[INPUT, MANDATORY, default=no default value]`

The position list to overlay.

Return

ArrayList

Returns a list of sources.

ArrayList addPositionList (PositionList positionList, Float1d sizes)

Overlays the given source list on this Display, with the given sizes.

Arguments

`PositionList` **`positionList`** `[INPUT, MANDATORY, default=no default value ]`

The position list to overlay.

`Float1d` **`sizes`** `[INPUT, MANDATORY, default=no default value ]`

The sizes to use for the positionList

Return

ArrayList

Returns a list of sources.

ArrayList addPositionList (PositionList positionList, Color color, Float1d sizes)

Overlays the given source list.

Overlays the given source list on this Display, in the given color, with the sizes defined in the sizes Float1d

Arguments

`PositionList` **`positionList`** `[INPUT, MANDATORY, default=no default value ]`

The position list to overlay.

`Color` **`color`** `[INPUT, MANDATORY, default=no default value ]`

ArrayList addPositionList (PositionList positionList, Color color, Float1d sizes)

The color in which to overlay the source list.

`Float1d` **`sizes`** `[INPUT, MANDATORY, default=no default value ]`

The sizes for each source (in pixel size).

Return

ArrayList

Returns a list of sources.

ArrayList addPositionListWcs (PositionList positionList, Color color, Float1d sizes)

Overlays the given source list on this Display.

Overlays the given source list on this Display, in the given color, with the sizes defined in the sizes Float1d (in arcseconds)

Arguments

`PositionList` **`positionList`** `[INPUT, MANDATORY, default=no default value ]`

The position list to overlay.

`Color` **`color`** `[INPUT, MANDATORY, default=no default value ]`

The color in which to overlay the source list.

`Float1d` **`sizes`** `[INPUT, MANDATORY, default=no default value ]`

The sizes for each source (in arc seconds).

Return

ArrayList

Returns a list of sources.

ArrayList addSourceFit (SourceFittingProduct source, Color color)

Visualises the given SourceFittingProduct on this Display.

Visualises the given SourceFittingProduct on this Display in the given color. Returns a list of ellipses/circles.

Arguments

`SourceFittingProduct` **`source`** `[INPUT, MANDATORY, default=no default value ]`

The given source fit.

`Color` **`color`** `[INPUT, MANDATORY, default=no default value ]`

The color.

Return

ArrayList

Returns a list of ellipses/circles.

ArrayList addSourceFit (SourceFittingProduct source)

Visualises the given SourceFittingProduct on this Display.

Returns a list of ellipses/circles.

Argument

ArrayList addSourceFit (SourceFittingProduct source)

SourceFittingProduct **source** [INPUT, MANDATORY, default=no default value]

The given source fit.

Return

ArrayList

Returns a list of ellipses/circles.

ArrayList showAxes (boolean showAxis)

Enables or disables the axes for the displayed image.

Enables or disables the axes for the displayed image. If the parameter is true, two axes are drawn, one on the left side and one at the bottom. An array with the axes is returned.

Argument

boolean **showAxis** [INPUT, MANDATORY, default=no default value]

True if the Axes should be shown.

Return

ArrayList

An ArrayList with the axes.

ImageAxis getLeftaxis

Returns the left axis.

Returns the left axis of the Display. If there is no left axis, the standard left axis is made, shown and returned.

Return

ImageAxis

The left axis

ImageAxis getRightaxis

Returns the right axis.

Returns the right axis of the Display. If there is no right axis, the standard right axis is made, shown and returned.

Return

ImageAxis

The right axis

ImageAxis getBottomaxis

Returns the bottom axis.

Returns the bottom axis of the Display. If there is no bottom axis, the standard bottom axis is made, shown and returned.

Return

ImageAxis

The bottom axis

ImageAxis getTopaxis

Returns the top axis.

Returns the top axis of the Display. If there is no top axis, the standard top axis is made, shown and returned.

Return

ImageAxis

The top axis

ArrayList getAxes

Returns an array with the axes.

Returns an array with the axes. Using methods on the Axes, the appearance can be changed.

Return

ArrayList

An ArrayList with the Axes.

addAxis (String label, AxisConstants.Position orientation)

Adds new axis.

Arguments

`String` **label** `[INPUT, MANDATORY, default=no default value ]`

The label of the axis

`AxisConstants.Position` **orientation** `[INPUT, MANDATORY, default=no default value ]`

The orientation for the axis

addAxis (ImageAxis axis)

Adds new axis.

Argument

`ImageAxis` **axis** `[INPUT, MANDATORY, default=no default value ]`

The axis to add

deleteAxis (ImageAxis axis)

Deletes the given axis.

Argument

`ImageAxis` **axis** `[INPUT, MANDATORY, default=no default value ]`

The axis to delete

setCenter (int row, int column)

Sets the center of the image.

Sets the (geometrical, not as defined in the Wcs) center of the image to a certain pixel on the display

Arguments

`int` **row** `[INPUT, MANDATORY, default=no default value ]`

The row position

`int` **column** `[INPUT, MANDATORY, default=no default value ]`

setCenter (int row, int column)

The column position

setCenter (double row, double column)

Sets the center of the image.

Sets the (geometrical, not as defined in the Wcs) center of the image to a certain pixel on the display

Arguments

`double` **`row`** `[INPUT, MANDATORY, default=no default value ]`

The row position

`double` **`column`** `[INPUT, MANDATORY, default=no default value ]`

The column position

setCenter (double row, double column, boolean update)

Sets the center of the image.

Sets the (geometrical, not as defined in the Wcs) center of the image to a certain pixel on the display

Arguments

`double` **`row`** `[INPUT, MANDATORY, default=no default value ]`

The row position

`double` **`column`** `[INPUT, MANDATORY, default=no default value ]`

The column position

`boolean` **`update`** `[INPUT, MANDATORY, default=no default value ]`

True if the image should be changed to the new center coordinates.

getCenter

Returns the center of the drawn image.

close

Closes the display and frees the memory.

editColors

Edit the colors using a popup.

A windows opens where you can change the color table, intensity and scale.

editCutLevels

Edit the cut levels using a popup.

A windows opens where you can set the cut levels of the image.

String **getColortable**

Returns the name of the color table.

Return

String

The name of the color table

double[] getCutLevels

Returns the cut levels.

Returns an array with the cut levels of the displayed image.

Return

double[]

The cut levels of the displayed image.

Image getImage

Returns the displayed image.

Return

Image

The displayed Image

Image getOriginalImage

Returns the displayed image.

Return

Image

The displayed Image

RgbImage getRgbImage

Returns the displayed image.

Return

RgbImage

The displayed Image

double[] getPixelCoordinates (double c1, double c2)

Returns the image coordinates.

Returns the image coordinates of the given world coordinates

Arguments

`double` **`c1`** `[INPUT, MANDATORY, default=no default value]`

The first world coordinate (right ascension in case of equatorial coordinates) of the image.

`double` **`c2`** `[INPUT, MANDATORY, default=no default value ]`

The second world coordinate (declination in case of equatorial coordinates) of the image.

Return

double[]

The corresponding image coordinates as a 2 dimensional array of doubles, the first one describing the row and the second the column.

Number getIntensity (int row, int column)

Returns the intensity of the pixel.

Returns the intensity of the pixel with given pixel coordinates.

Arguments

Number **getIntensity (int row, int column)**

`int` **`row`** `[INPUT, MANDATORY, default=no default value]`

The row of the image.

`int` **`column`** `[INPUT, MANDATORY, default=no default value]`

The column of the image.

Return

Number

The intensity of the given pixel

Number **getIntensityFromWorldCoordinates (double c1, double c2)**

Returns the intensity of the pixel.

Returns the intensity of the pixel with given world coordinates.

Arguments

`double` **`c1`** `[INPUT, MANDATORY, default=no default value]`

The first world coordinate of the image.

`double` **`c2`** `[INPUT, MANDATORY, default=no default value]`

The second world coordinate of the image.

Return

Number

double The intensity of the given pixel.

Unit **getUnit**

Returns the unit of the image

Return

Unit

The unit of the image.

float **getZoomFactor**

Returns the used zoom factor.

Returns the used zoom factor. The returned zoomfactor is bigger than 1 if the image is zoomed in, otherwise, the zoomfactor is between 0 and 1

Return

float

The zoomfactor of the displayed image.

float **getZoomFactorX**

Returns the used X zoom factor.

Returns the used X zoom factor. The returned zoomfactor is bigger than 1 if the image is zoomed in, otherwise, the zoomfactor is between 0 and 1

Return

float

The X zoomfactor of the displayed image.

float getZoomFactorY

Returns the used Y zoom factor.

Returns the used Y zoom factor. The returned Y zoomfactor is bigger than 1 if the image is zoomed in, otherwise, the zoomfactor is between 0 and 1

Return

float

The Y zoomfactor of the displayed image.

printDialog

Opens a printer dialog.

A dialog is opened where you can choice the printer, number of copies, page setup and appearance.

getPrintControl

Returns the printControl.

Returns the printControl to make it possible to print from the command line.

Example

How to print from the command line

```
job = d.getPrintJob()
job.setCopies(2) # Set the number of copies
pservices = job.lookupPrintServices()
job.setPrintService(pservices[3])
#Use the printer of choice
d.setPrintJob(job)
job.print()
```

createPrintJob

Creates a print job.

PrinterJob getPrintJob

Returns the printer job.

Return

PrinterJob

The printerJob.

setPrintJob (PrinterJob job)

Set a given printer job.

Argument

PrinterJob **job** [INPUT, MANDATORY, default=no default value]

The printer job

print (HashPrintRequestAttributeSet attributes)

Print the displayed image to a ps file.

Argument

HashPrintRequestAttributeSet **attributes** [INPUT, MANDATORY, default=no default value]

print (HashPrintRequestAttributeSet attributes)

The attributes to print.

save

Pops up a dialog to save your image.

Pops up a file chooser and saves the image as fits, jpeg, tiff, png or bmp file.

save (String filename)

Saves the whole image, including annotations.

Argument

`String` **filename** `[INPUT, MANDATORY, default=no default value]`

Saves the whole image, including annotations.

saveAsJPG (String filename)

Saves the display as a jpg file.

Argument

`String` **filename** `[INPUT, MANDATORY, default=no default value]`

The filename with its path

saveAsPNG (String filename)

Saves the display as a png file.

Argument

`String` **filename** `[INPUT, MANDATORY, default=no default value]`

The filename with its path

saveAsEPS (String filename)

Saves the display as a eps file.

Argument

`String` **filename** `[INPUT, MANDATORY, default=no default value]`

The filename with its path

saveAsPDF (String filename)

Saves the display as a pdf file.

Argument

`String` **filename** `[INPUT, MANDATORY, default=no default value]`

The filename with its path

saveCurrentViewAsJPG (String filename)

Saves the display as a jpg file.

Argument

`String` **filename** `[INPUT, MANDATORY, default=no default value]`

The filename with its path

saveCurrentViewAsPNG (String filename)

Saves the display as a png file.

Argument

saveCurrentViewAsPNG (String filename)

String **filename** [INPUT, MANDATORY, default=no default value]

The filename with its path

saveCurrentViewAsEPS (String filename)

Saves the display as a eps file.

Argument

String **filename** [INPUT, MANDATORY, default=no default value]

The filename with its path

saveCurrentViewAsPDF (String filename)

Saves the display as a pdf file.

Argument

String **filename** [INPUT, MANDATORY, default=no default value]

The filename with its path

setTitle (String text)

Sets Sets the title of the image.

Argument

String **text** [INPUT, MANDATORY, default=no default value]

The title to set

saveCurrentView (String filename)

Saves the current view of the image.

Saves the view as screenshot. The annotations are also saved!

Argument

String **filename** [INPUT, MANDATORY, default=no default value]

The filename with its path

setColortable (String colortableName)

Sets the color table of the displayed image.

Argument

String **colortableName** [INPUT, MANDATORY, default=no default value]

The name of the color table as a String.

setColortable (String colortableName, String intensityName)

Sets the colortable of the displayed image.

Arguments

String **colortableName** [INPUT, MANDATORY, default=no default value]

The name of the colortable as a String.

String **intensityName** [INPUT, MANDATORY, default=no default value]

The name of an intensity lookup table that can be added to the image processing chain to rearrange the order of the colors in the colormap.

setColortable (String colortableName, String intensityName, String scaleName)

Sets the colortable of the displayed image

Arguments

String **colortableName** [INPUT, MANDATORY, default=no default value]

The name of the colortable as a String.

String **intensityName** [INPUT, MANDATORY, default=no default value]

The name of an intensity lookup table that can be added to the image processing chain to rearrange the order of the colors in the colormap.

String **scaleName** [INPUT, MANDATORY, default=no default value]

The algorithm to use to scale the image to byte range.

setCutLevelsPercentage (double percent)

Sets the cut level of the displayed image.

Argument

double **percent** [INPUT, MANDATORY, default=no default value]

The percentage of pixels that should be taken into account.

setCutLevelsMin (double min)

Sets the minimum cut levels.

Sets the minimum cut level of the displayed image.

Argument

double **min** [INPUT, MANDATORY, default=no default value]

The minimum for the cutlevels, the maximum is not adapted.

setCutLevelsMax (double max)

Sets the maximum cut levels.

Sets the maximum cut level of the displayed image.

Argument

double **max** [INPUT, MANDATORY, default=no default value]

The maximum for the cutlevels, the minimum is not adapted.

setCutLevelsMinMax (double min, double max)

Sets the cut level of the displayed image.

Arguments

double **min** [INPUT, MANDATORY, default=no default value]

The low cut value.

double **max** [INPUT, MANDATORY, default=no default value]

The high cut value.

calcCutLevels

Sets the cut level of the displayed image.

Sets the cut levels of the displayed image to the minimum and maximum value of the image.

setCutLevels (double[] minmax)

Sets the cut level of the displayed image.

Argument

`double[] minmax [INPUT, MANDATORY, default=no default value]`

An array of doubles with the low cut value and the high cut value.

setUnit (Unit<?> u)

Sets the unit of the displayed image.

Argument

`Unit&lt;?&gt; u [INPUT, MANDATORY, default=no default value]`

The unit to set

setZoomFactor (float zoomFactor)

Sets the zoomfactor of the displayed image.

Argument

`float zoomFactor [INPUT, MANDATORY, default=no default value]`

The new zoom factor of the image.

setZoomFactorX (float zoomFactorX)

Sets the X zoomfactor of the displayed image.

Argument

`float zoomFactorX [INPUT, MANDATORY, default=no default value]`

The new zoom factor of the image, only in X-direction.

setZoomFactorY (float zoomFactorY)

Sets the Y zoomfactor of the displayed image.

Argument

`float zoomFactorY [INPUT, MANDATORY, default=no default value]`

The new zoom factor of the image, only in Y-direction.

zoom (double row, double column, float zoomFactor)

Zooms the image.

Sets the zoomfactor of the displayed image and the pixel coordinates which should appear in the center of the image.

Arguments

`double row [INPUT, MANDATORY, default=no default value]`

The row where the zoom should be centered on.

`double column [INPUT, MANDATORY, default=no default value]`

The column where the zoom should be centered on.

`float zoomFactor [INPUT, MANDATORY, default=no default value]`

The needed zoomFactor.

zoomWorldCoordinates (double ra, double decl, float zoomFactor)

Zooms the image.

zoomWorldCoordinates (double ra, double decl, float zoomFactor)

Sets the zoomfactor of the displayed image and the world coordinates which should appear in the center of the image.

Arguments

`double` **`ra`** `[INPUT, MANDATORY, default=no default value]`

The right ascension of the image where the zoom should be centered on.

`double` **`decl`** `[INPUT, MANDATORY, default=no default value]`

The declination of the image where the zoom should be centered on.

`float` **`zoomFactor`** `[INPUT, MANDATORY, default=no default value]`

The needed zoomFactor.

zoomIn

Zooms the image in.

Zooms the displayed image. The zoom factor changes the following way : ...1/8, 1/4, 1/2, 1, 2, 4, 8, ...

zoomOut

Zoom the image out.

Zooms the displayed image out. The zoom factor changes the following way : ..., 8, 4, 2, 1, 1/2, 1/4, 1/8, ...

zoomFit

Zoom the image.

Zooms the image to a zoomfactor so that the image is shown in total on the screen. Axes are taken into account

flipYAxis

Flips the y axis of the displayed image.

If this method is called, the current y-axis is flipped. If the displayed image has the WCS keyword flipy, this method will have no effect.

flipNoAxis

Do not flip the x or y axis of the displayed image.

If this method is called, the current axes are not flipped. If the displayed image has the WCS keyword flipy, this method will have no effect.

flipXAxis

Flips the x axis of the displayed image.

If this method is called, the current x-axis is flipped. If the displayed image has the WCS keyword flipx, this method will have no effect.

flipXYAxis

Flips the xy axis of the displayed image.

If this method is called, the axes are flipped over a diagonal connection (0,0)-(1,1). If the displayed image has the WCS keyword flipxy, this method will have no effect.

flipYXAxis

Flips the yx axis of the displayed image.

If this method is called, the axes are flipped over a diagonal connection (1,0)-(0,1). If the displayed image has the WCS keyword flipyx, this method will have no effect.

setFlipYAxis (boolean flipAxis)

Sets whether the current image should be flipped.

The current image will be flipped over the Y-axis.

Argument

`boolean` **`flipAxis`** `[INPUT, MANDATORY, default=no default value]`

True to flip the y axis of the current image.

setFlipXAxis (boolean flipXAxis)

Sets whether the current image should be flipped.

The current image will be flipped over the x-axis.

Argument

`boolean` **`flipXAxis`** `[INPUT, MANDATORY, default=no default value]`

True to flip the x axis of the current image.

setFlipXYAxis (boolean flipXYAxis)

Sets whether the current image should be flipped.

The current image will be flipped over the diagonal connecting (0,0)-(1,1).

Argument

`boolean` **`flipXYAxis`** `[INPUT, MANDATORY, default=no default value]`

True to flip the xy axis of the current image.

setFlipYXAxis (boolean flipYXAxis)

Sets whether the current image should be flipped.

The current image will be flipped over the diagonal connecting (1,0)-(0,1)

Argument

`boolean` **`flipYXAxis`** `[INPUT, MANDATORY, default=no default value]`

True to flip the yx axis of the current image.

getFlipYAxis

Returns whether the Y axis is flipped.

Returns true if the current image is flipped over the y-axis.

getFlipXAxis

Returns whether the X axis is flipped.

Returns true if the current image is flipped over the x-axis.

getFlipXYAxis

Returns whether the image is flipped.

getFlipYXAxis

Returns whether the image is flipped.

isFlipped

Returns whether the current layer is flipped.

Returns true if the current layer is flipped over the y-axis.

isFlippedX

Returns whether the current layer is flipped.

Returns true if the current layer is flipped over the x-axis.

isFlippedXY

Returns whether the current layer is xy-flipped.

Returns true if the current layer is xy-flipped.

isFlippedYX

Returns whether the current layer is yx-flipped.

Returns true if the current layer is yx-flipped.

setDepthAxis (int depthAxis)

Sets the depth axis for cubes.

Set the depth axis for the current and newly added cubes. The first axis (0) is the standard depth axis.

Argument

`int` **`depthAxis`** `[INPUT, MANDATORY, default=no default value]`

The depth axis for newly added cubes.

getDepthAxis

Returns the depth axis for cubes.

Returns which axis is the depth axis.

See also

- SimpleImage
- SimpleCube
- Developers Manual: herschel.ia.gui.image.Display

1.110. DistortionMaps

Full Name:	herschel.ia.toolbox.pointing.DistortionMaps
Type:	Jython Task -
Import:	from herschel.ia.toolbox.pointing import DistortionMaps
Category	Toolboxes/Pointing

Description

Class providing access to the Herschel Distortion Maps via HSA or local file.

Configure local distortion maps using properties: hcss.pointing.distortion.map.dy.v2 hcss.pointing.distortion.map.dz.v2 hcss.pointing.distortion.map.dy.v3 hcss.pointing.distortion.map.dz.v3

API Summary

Jython Syntax

```
dm = DistortionMaps(<odNumber>)
# subpixel distortion maps
fovimagey = dm.fovimagey
fovimagez = dm.fovimagez
```

Property

```
integer odNumber [INPUT, MANDATORY, default=NO default value ]
```

API details

Property

integer odNumber [INPUT, MANDATORY, default=NO default value]

OD number to retrieve the Distortion Maps for

History

- 2015-04-07 - JPC: HCSS-17969 Pointing toolbox installs 400-800MB worth of data in user installation

1.111. divide

Full Name:	herschel.ia.toolbox.spectrum.DivideSpectrumTask
Alias:	divide
Type:	Java Task -
Import:	from herschel.ia.toolbox.spectrum import DivideSpectrumTask
Category	Spectra/Analysis

Description

Task for dividing the flux data of spectra by a scalar or for pairwise dividing spectra.

For the scalar mode, 'ds' and 'param' need to be specified - the input spectra and the scalar to divide by; for the pair-wise mode, 'ds1' and 'ds2' need to be set - two spectrum containers. In this pair-wise mode, the first spectrum in the first container is divided by the first spectrum in the second container, and so on. In case the size of the two containers is different, the result will contain a number of point spectra equal to the minimum size. Hence, the remaining spectra in the larger container are ignored. The behavior is different in case one of the data sets only contains a single spectrum: Here, the task always refers to single spectrum while iterating over all the spectra in the other container.

In the pairwise mode, the individual spectra are processed on a per frequency or wavelength bin (or whatever the wavescale unit is) basis. Hence, there is no check of whether the frequencies (or wavelengths) assigned to these bins are well aligned across the two spectra. If this is not the case, the spectra should first be re-sampled to a common wave scale grid.

The input data with the spectra to be processed needs to be an object that implements `SpectrumContainer` (e.g. `Spectrum1d, Spectrum2d, SpectralSimpleCube`). `SpectrumContainer`'s provides a uniform way to access spectrum data organized in quite different structures.

For a successful processing in the pair-wise mode, the data specified with `ds1` and `ds2` should be consistent, i.e. they should have the same wave scale range and spectral sampling. Furthermore, the segments found in all the spectra in the (two) containers should be consistent (same number of segments, same lengths, same segment indices - e.g. check with `ds.getSegmentIndices()`). Otherwise an exception is thrown.

Different selection schemes are available for selecting the point spectra or the segments to be processed. The most simple scheme is to specify lists of point spectrum indices (`selection=[1,3,4,2]`). In this case, the operation (scalar- or pair-operation) is just taken over the point spectra with corresponding indices. Instead of specifying a list of indices (which requires the knowledge of these) other recipes to specify a selection are available. Typical recipes test attribute values to match certain values (e.g. `selection={"bbtype":[6031, 6032]}`) or to be included in a given interval (e.g. `selection={"LoFrequency":(999.98,1000.02)}`). These recipes we refer to as selection models. The different options to specify selections can be combined. Here, an AND logic is adopted. See the parameter descriptions and the examples below for further detail. Alternatively, look in the `SelectSpectrum`-task.

Similarly, you can specify what segments to consider. In the most simple case, you can just specify the indices of segments to be considered (for the scalar operation `segments=[1,3,4]` and for the pair operation `segments1=[1,3,4], segments2=[3,6,5]`). In more advanced situations you can use a `SegmentSelection` object. Note that the pairs of segments to be processed need to be consistent - otherwise the processing will fail.

You can specify whether the orginal datasets should be overwritten with the 'overwrite' flag. By default no data is overwritten. In the pair-wise mode, it is not always possible to overwrite the data - in particular if non-trivial segment selections are specified.

Examples

Example 1: for scalar divide:

```
global spectra # defined elsewhere
divide = DivideSpectrumTask()
spectraOut = divide(ds=spectra, param=2.1)
# restrict the divide to a suitable selection of spectra and return the
 processed
# just select by index:
spectraOut = divide(ds=spectra, selection=[0,1,2,3], param=2.1)
# select by looking up suitable attributes attached to the spectra to be
 selected (here: "bbtype" matched):
spectraOut = divide(ds=spectra, selection={"bbtype":[6031,6613]}, param=2.1)
# select by looking up suitable attributes attached to the spectra to be
 selected (here: "Chopper" in given ranges):
spectraOut = divide(ds=spectra, selection={"Chopper":(4.,7.), "bbtype":[6613]},
 param=2.1)
# select by looking up suitable attributes attached to the spectra to be
 selected
# (here: "LoFrequency" in given interval):
spectraOut = divide(ds=spectra, selection={"LoFrequency":(550.0,570.0)},
 param=2.1)
# the same as above - this time by by modifying the input spectra:
spectra = divide(ds=spectra, selection={"bbtype":[6031,6613]}, param=2.1,
 overwrite=True)
spectra = divide(ds=spectra, selection=[0,1,2,3], param=2.1, overwrite=True)
spectra = divide(ds=spectra, selection={"Chopper":(4.,7.), "bbtype":[6613]},
 param=2.1, overwrite=True)
spectra = divide(ds=spectra, selection={"LoFrequency":(550.0,570.0)},
 param=2.1, overwrite=True)
```

Example 2: for pairwise divide:

```
global spectra1, spectra2 # defined elsewhere
divide = DivideSpectrumTask()
spectraOut = divide(ds1=spectra1, ds2=spectra2)
# restrict the divide to a suitable selection of spectra and return the
 processed
# just select by index:
spectraOut = divide(ds1=spectra1, ds2=spectra2, selection=[0,1,2,3])
# select by looking up suitable attributes attached to the spectra to be
 selected (here: "bbtype" matched):
spectraOut = divide(ds1=spectra1, ds2=spectra2, selection={"bbtype":
[6031,6613]})
# select by looking up suitable attributes attached to the spectra to be
 selected (here: "Chopper" in given ranges):
spectraOut = divide(ds1=spectra1, ds2=spectra2, selection={"Chopper":(4.,7.),
 "bbtype":[6613]})
# select by looking up suitable attributes attached to the spectra to be
 selected
# (here: "LoFrequency" in given interval):
spectraOut = divide(ds1=spectra1, ds2=spectra2, selection={"LoFrequency":
(550.0,570.0)})
```

API Summary

Properties
SpectrumContainer **ds** [INPUT, OPTIONAL, default=no default value.]
Double **param** [INPUT, OPTIONAL, default=no default value.]
Object **selection** [INPUT, OPTIONAL, default=None.]
PyDictionary|Map<String,Set<Object>> **lookup selection** [INPUT, OPTIONAL, default=no default value.]

Properties
PyList **index selection** [INPUT, OPTIONAL, default=No default value.]
Boolean **overwrite** [INPUT, OPTIONAL, default=False.]
SpectrumContainer **ds1** [INPUT, OPTIONAL, default=no default value.]
SpectrumContainer **ds2** [INPUT, OPTIONAL, default=no default value.]
Object **segments** [INPUT, OPTIONAL, default=no default value.]
Object **segments1** [INPUT, OPTIONAL, default=no default value.]
Object **segments2** [INPUT, OPTIONAL, default=no default value.]
SpectrumContainer **result** [OUTPUT, OPTIONAL, default=no default value]

API details

Properties

SpectrumContainer ds [INPUT, OPTIONAL, default=no default value.]

Input container to be processed by the task in case the task is used as a scalar operation. Examples of `SpectrumContainer` are `Spectrum1d, Spectrum2d, SpectralSimpleCube`.

Double param [INPUT, OPTIONAL, default=no default value.]

The scalar parameter to be considered when the task is used as scalar operation.

Object selection [INPUT, OPTIONAL, default=None.]

Specify what point spectra the operation should be applied to. Different ways to specify these selections are possible:

- Specify a list of indices (in jython) of the point spectra for which the divide should be applied.
- Use a dictionary (in jython) to specify a selection model (lookup for attributes, filter attributes to be included in ranges or intervals).
- Pass a string representation of a jython list or a jython dictionary. The string will be parsed and the processing delegated to the two cases above.
- Pass any java instance that implements the SelectionModel interface (for the advanced user).

See the examples below or the SelectSpectrumTask for how to specify selections.

PyDictionary|Map<String,Set<Object>> lookup_selection [INPUT, OPTIONAL, default=no default value.]

Specify a PyDictionary with keys given by the attribute name to look up and a list of admissible values. Behind the scenes, a DiscreteValueSelectionModel is instantiated. This parameter is actually obsolete but is kept for historical reasons (use `selection`).

PyList index_selection [INPUT, OPTIONAL, default=No default value.]

Specify a PyList with the indices of the point spectra to be considered. This parameter is actually obsolete but is kept for historical reasons (use `selection`).

Boolean overwrite [INPUT, OPTIONAL, default=False.]

Specify whether the input data container can be reused - the values found therein is overwritten.

SpectrumContainer ds1 [INPUT, OPTIONAL, default=no default value.]

First input container for pair-wise operations.

SpectrumContainer ds2 [INPUT, OPTIONAL, default=no default value.]

Second input container for pair-wise operations.

Object segments [INPUT, OPTIONAL, default=no default value.]

Specify what segments the operation should be applied to. There are two options available:

- Either pass an instance of a SegmentSelection which gives the information on what segments for each point spectrum included in the container.
- Specify a PyList of segment indices.

Object segments1 [INPUT, OPTIONAL, default=no default value.]

Specify the segment selection to be associated with 'ds1'.

Object segments2 [INPUT, OPTIONAL, default=no default value.]

Specify the segment selection to be associated with 'ds2'.

SpectrumContainer result [OUTPUT, OPTIONAL, default=no default value]

Result object containing the results of the operation applied.

See also

- SpectrumContainer
- select
- Developers Manual: herschel.ia.toolbox.spectrum.DivideSpectrumTask

History

- `2011-08-08 - melchior`: renamed from DivideSpectrum

1.112. Double1d

Full Name:	herschel.ia.numeric.Double1d
Type:	Java Class -
Import:	from herschel.ia.numeric import Double1d
Category	Arrays and datasets

Description

A rectangular numeric double array of rank 1.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Double1d

1.113. Double2d

Full Name:	herschel.ia.numeric.Double2d
Type:	Java Class -
Import:	from herschel.ia.numeric import Double2d
Category	Arrays and datasets

Description

A rectangular numeric double array of rank 2.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Double2d

1.114. Double3d

Full Name:	herschel.ia.numeric.Double3d
Type:	Java Class -
Import:	from herschel.ia.numeric import Double3d
Category	Arrays and datasets

Description

A rectangular numeric double array of rank 3.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Double3d

1.115. Double4d

Full Name:	herschel.ia.numeric.Double4d
Type:	Java Class -
Import:	from herschel.ia.numeric import Double4d
Category	Arrays and datasets

Description

A rectangular numeric double array of rank 4.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Double4d

1.116. Double5d

Full Name:	herschel.ia.numeric.Double5d
Type:	Java Class -
Import:	from herschel.ia.numeric import Double5d
Category	Arrays and datasets

Description

A rectangular numeric double array of rank 5.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Double5d

1.117. Dwt

Full Name:	herschel.ia.numeric.toolbox.wavelet.walgo.Dwt
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.wavelet.walgo import Dwt

Description

A discrete wavelet transform algorithm

This class is not intended to be used directly. DiscreteWavelet handles this class and should be used by the users.

Example

Example 1: Jython Usage Example

```
from herschel.ia.numeric.toolbox.wavelet.walgo    import Dwt
from herschel.ia.numeric.toolbox.wavelet.wlibrary import WaveletLoader
from herschel.ia.numeric.toolbox.wavelet.wutil    import WBorder
from herschel.ia.numeric.toolbox.wavelet.wdemo    import WSigGenerator
from herschel.ia.numeric.toolbox.wavelet.wdemo    import WReadImage
#
sigGen = WSigGenerator();
sig    = sigGen.getPredefinedRealFunction3()
#
o = WaveletLoader()
o.setDwtPreference()
w = o.get("coif3")
#
dwt    = Dwt()
coefs  = dwt.decompose(sig, w,WBorder.SYMMETRIC,3)
#
res  = dwt.synthesis(coefs)
plot = PlotXY(res);
plot.setSubtitleText(w.getName()+": Synthesis signal")
#--- 2D ---
image = Double2d(129, 129)
for y in range(129):
    for x in range(129):
        image[y, x] = COS(y / 3) + COS(y / 7) + COS(x / 60) + COS(x / 97)
#
w         = o.get("db1")
dwt       = Dwt()
tree      = dwt.decompose(image,w,WBorder.SYMMETRIC,4)
d         = Display(image)
c         = Display(tree.compose())
# -- synthesis
dwt.synthesis(tree)
res       = tree.getRoot().getData();
r         = Display(res)
# -- checking accuracy
dim  = image.dimensions
dim[0] -= w.length
dim[1] -= w.length
diff = image[0:dim[0],0:dim[1]]        # clip few border pixels
diff.subtract(res[0:dim[0],0:dim[1]])
print "Original image properties:"
print "MIN    =",MIN(image)
print "MAX    =",MAX(image)
print "Synthesis image minus original image :"
print "STDDev =",STDDEV(diff)
print "MEAN   =",MEAN(diff)
print "MIN    =",MIN(diff)
print "MAX    =",MAX(diff)
```

Example 1: Jython Usage Example

```
# display remainder
Display(diff)
```

See also

- Developers Manual: herschel.ia.numeric.toolbox.wavelet.walgo.Dwt

History

- `2010-09-10 - BM`: initial version

1.118. EigenvalueDecomposition

Full Name:	herschel.ia.numeric.toolbox.matrix.EigenvalueDecomposition
Alias:	EigenvalueDecomposition
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.matrix import EigenvalueDecomposition
Category	Mathematics/Matrices

Description

EigenvalueDecomposition

Eigenvalues and eigenvectors of a real matrix. If A is symmetric, then A = V*D*V' where the eigenvalue matrix D is diagonal and the eigenvector matrix V is orthogonal. If A is not symmetric, then the eigenvalue matrix D is block diagonal with the real eigenvalues in 1-by-1 blocks and any complex eigenvalues, lambda + i*mu, in 2-by-2 blocks, [lambda, mu; -mu, lambda]. The columns of V represent the eigenvectors in the sense that A*V = V*D. The matrix V may be badly conditioned, or even singular, so the validity of the equation A = V*D*inverse(V) depends upon Vcond().

Example

Example 1: The columns of V represent the Eigenvectors: A*V = V*D

```
A=Double2d( [ [1.,1.,1.],[1.,2.,3.],[1.,3.,6.] ] )
evd=A.apply(EigenvalueDecomposition())
D = evd.d
V = evd.v
print A.apply(MatrixMultiply(V))
# [
# [-0.06907722348532569,-0.8164965809277256,1.5259625390319682],
# [0.09922885633779388,-0.4082482904638629,3.7179950198195835],
# [-0.03892559063285694,0.4082482904638627,6.769920097883521]
# ]
print V.apply(MatrixMultiply(D))
# [
# [-0.06907722348532568,-0.8164965809277251,1.525962539031968],
# [0.09922885633779421,-0.4082482904638631,3.717995019819584],
# [-0.03892559063285706,0.4082482904638628,6.769920097883519]
# ]
```

API Summary

Jython Syntax

```
A=Double2d()
evd=A.apply(EigenvalueDecomposition())
# get the eigenvalue matrix D
D = evd.d
# get the eigenvector matrix V
V = evd.v
# Get the imaginary eigenvalues
imagEigenvalues = evd.imagEigenvalues
# Get the real eigenvalues
realEigenvalues = evd.realEigenvalues
```

Property
Double2d **A** [INPUT, MANDATORY, default=no default value]

API details

Property

Double2d A [INPUT, MANDATORY, default=no default value]
Input must be a Double2d or Float2d array.

See also

- Developers Manual: herschel.ia.numeric.toolbox.matrix.EigenvalueDecomposition

1.119. ellipseHistogram

Full Name:	herschel.ia.toolbox.image.EllipseHistogramTask
Alias:	ellipseHistogram
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import EllipseHistogramTask
Category	Images/Display

Description

This is a task to make a histogram of a region of interest on image which is bounded by an ellipse.

This is a task to make a histogram of a region of interest on an image which is bounded by an ellipse. You must specify the number of bins for the histogram. By default the cut levels of the image will be used to construct the histogram, but it is also possible for you to specify the low and high cut level for the histogram. To define the position of the ellipse, you must specify the pixel or sky coordinates of the center of the ellipse and its width and height. Note that the axes of the ellipse are aligned with the x- and y-axis.

Example

Example 1: This is an example of how to use the ellipseHistogram :

```
histogram = ellipseHistogram(image = myImage, bins = 200, lowCut = 100.0,
 highCut = 150.0,
                              centerX = 100.0, centerY = 230.0, widthArcsec =
 50.0,
                              heightArcsec = 30.0)
histogram = ellipseHistogram(image = myImage, bins = 200, lowCut = 100.0,
 highCut = 150.0,
                              centerRa = "05:46:45.9", centerDec =
 "-51:07:57.0",
                              widthPixels = 50.0, heightPixels = 30.0)
```

API Summary

Properties
Image **image** [INPUT, MANDATORY, default=None]
Double **lowCut** [INPUT, OPTIONAL, default=NaN]
Double **highCut** [INPUT, OPTIONAL, default=NaN]
Integer **bins** [INPUT, OPTIONAL, default=2000]
Double **centerX** [INPUT, OPTIONAL, default=NaN]
Double **centerY** [INPUT, OPTIONAL, default=NaN]
String **centerRa** [INPUT, OPTIONAL, default="centerRa"]
String **centerDec** [INPUT, OPTIONAL, default="centerDec"]
Double **widthPixels** [INPUT, OPTIONAL, default=NaN]
Double **heightPixels** [INPUT, OPTIONAL, default=NaN]
Double **widthArcsec** [INPUT, OPTIONAL, default=NaN]
Double **heightArcsec** [INPUT, OPTIONAL, default=NaN]
String **centerRA** [INPUT, OPTIONAL, default="centerRA"]

Properties
CircleHistogramProduct **histogram** [OUTPUT, MANDATORY, default=None]

API details

Properties

Image image [INPUT, MANDATORY, default=None]

This is the input image.

Double lowCut [INPUT, OPTIONAL, default=NaN]

This is the low cut level you want to use for the histogram. If this is not specified, the image cut levels will be used.

Double highCut [INPUT, OPTIONAL, default=NaN]

This is the high cut level you want to use for the histogram. If this is not specified, the image cut levels will be used.

Integer bins [INPUT, OPTIONAL, default=2000]

This is the number of bins in the histogram.

Double centerX [INPUT, OPTIONAL, default=NaN]

The x-pixel-coordinate of the center of the ellipse.

Double centerY [INPUT, OPTIONAL, default=NaN]

The y-pixel-coordinate of the center of the ellipse.

String centerRa [INPUT, OPTIONAL, default="centerRa"]

This is the right ascension of the center of the ellipse. Using this only makes sense if the input image has a valid Wcs attached to it. The input format for the right ascension is "hh mm ss.s", "hh:mm:ss.s" or "hh.hhh".

String centerDec [INPUT, OPTIONAL, default="centerDec"]

This is the declination of the center of the ellipse. Using this only makes sense if the input image has a valid Wcs attached to it. The input format for the declination is "[+/-]dd mm ss.s", "[+/-]dd:mm:ss.s" or "[+/-]dd.ddd".

Double widthPixels [INPUT, OPTIONAL, default=NaN]

This is the width of the ellipse in pixels.

Double heightPixels [INPUT, OPTIONAL, default=NaN]

This is the height of the ellipse in pixels.

Double widthArcsec [INPUT, OPTIONAL, default=NaN]

This is the width of the ellipse in arcsec.

Double heightArcsec [INPUT, OPTIONAL, default=NaN]

This is the height of the ellipse in arcsec.

String centerRA [INPUT, OPTIONAL, default="centerRA"]
This is the right ascension of the center of the ellipse. Using this only makes sense if the input image has a valid Wcs attached to it. The input format for the right ascension is "hh mm ss.s", "hh:mm:ss.s" or "hh.hhh". Deprecated : Use CenterRa.

CircleHistogramProduct histogram [OUTPUT, MANDATORY, default=None]
This is the resulting histogram.

See also

- Developers Manual: herschel.ia.toolbox.image.EllipseHistogramTask

1.120. EllipseHistogramProduct

Full Name:	herschel.ia.dataset.image.EllipseHistogramProduct
Type:	Java Class -
Import:	from herschel.ia.dataset.image import EllipseHistogramProduct
Category	Images/Display

Description

A class to deal with the results of an ellipse histogram.

API Summary

Constructor
`EllipseHistogramProduct` The constructor of a new EllipseHistogramProduct.

Methods
`setCenter (double centerX, double centerY)` Sets the center for this EllipseHistogramProduct to the given pixel coordinates.
`setCenter (double centerX, double centerY, String centerRA, String centerDec)` Sets the center for this EllipseHistogramProduct to the given pixel and sky coordinates.
`setDimensions (double widthPixels, double heightPixels)` Sets the dimensions for this EllipseHistogramProduct to the given dimensions in pixels.
`setDimensions (double widthPixels, double heightPixels, double widthArcsec, double heightArcsec)` Sets the dimensions for this EllipseHistogramProduct to the given dimensions in pixels and arc-sec.
`Double1d` **`getCenterPixelCoordinates`** Returns the center for this EllipseHistogramProduct in pixel coordinates.
`String1d` **`getCenterSkyCoordinates`** Returns the center for this EllipseHistogramProduct in sky coordinates.
`double` **`getWidthPixels`** Returns the width for this EllipseHistogramProduct in pixels.
`double` **`getWidthArcsec`** Returns the width for this EllipseHistogramProduct in arcsec.
`double` **`getHeightPixels`** Returns the height for this EllipseHistogramProduct in pixels.
`double` **`getHeightArcsec`** Returns the height for this EllipseHistogramProduct in arcsec.

API Details

Constructor

EllipseHistogramProduct

The constructor of a new EllipseHistogramProduct.

Methods

setCenter (double centerX, double centerY)

Sets the center for this EllipseHistogramProduct to the given pixel coordinates.

Arguments

`double` **centerX** `[INPUT, MANDATORY, default=no default value]`

The x-pixel-coordinate of the center, as double

`double` **centerY** `[INPUT, MANDATORY, default=no default value]`

The y-pixel-coordinate of the center, as double

setCenter (double centerX, double centerY, String centerRA, String centerDec)

Sets the center for this EllipseHistogramProduct to the given pixel and sky coordinates.

Arguments

`double` **centerX** `[INPUT, MANDATORY, default=no default value]`

The x-pixel-coordinate of the center, as double

`double` **centerY** `[INPUT, MANDATORY, default=no default value]`

The y-pixel-coordinate of the center, as double

`String` **centerRA** `[INPUT, MANDATORY, default=no default value]`

The right ascension of the center, as String

`String` **centerDec** `[INPUT, MANDATORY, default=no default value]`

The declination of the center, as String

setDimensions (double widthPixels, double heightPixels)

Sets the dimensions for this EllipseHistogramProduct to the given dimensions in pixels.

Arguments

`double` **widthPixels** `[INPUT, MANDATORY, default=no default value]`

The width in pixels, as double

`double` **heightPixels** `[INPUT, MANDATORY, default=no default value]`

The height in pixels, as double

setDimensions (double widthPixels, double heightPixels, double widthArcsec, double heightArcsec)

Sets the dimensions for this EllipseHistogramProduct to the given dimensions in pixels and arcsec.

Arguments

`double` **widthPixels** `[INPUT, MANDATORY, default=no default value]`

The width in pixels, as double

`double` **heightPixels** `[INPUT, MANDATORY, default=no default value]`

setDimensions (double widthPixels, double heightPixels, double widthArcsec, double heightArcsec)

The height in pixels, as double

`double` **`widthArcsec`** `[INPUT, MANDATORY, default=no default value]`

The width in arcsec, as double

`double` **`heightArcsec`** `[INPUT, MANDATORY, default=no default value]`

The width in arcsec, as double

Double1d **getCenterPixelCoordinates**

Returns the center for this EllipseHistogramProduct in pixel coordinates.

Return

Double1d

Returns the center for this EllipseHistogramProduct in pixel coordinates.

String1d **getCenterSkyCoordinates**

Returns the center for this EllipseHistogramProduct in sky coordinates.

Return

String1d

Returns the center for this EllipseHistogramProduct in sky coordinates.

double **getWidthPixels**

Returns the width for this EllipseHistogramProduct in pixels.

Return

double

Returns the width for this EllipseHistogramProduct in pixels.

double **getWidthArcsec**

Returns the width for this EllipseHistogramProduct in arcsec.

Return

double

Returns the width for this EllipseHistogramProduct in arcsec.

double **getHeightPixels**

Returns the height for this EllipseHistogramProduct in pixels.

Return

double

Returns the height for this EllipseHistogramProduct in pixels.

double **getHeightArcsec**

Returns the height for this EllipseHistogramProduct in arcsec.

Return

double

Returns the height for this EllipseHistogramProduct in arcsec.

See also

- Developers Manual: herschel.ia.dataset.image.EllipseHistogramProduct

1.121. EngineList

Full Name:	herschel.ia.numeric.toolbox.fit.sample.EngineList
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit.sample import EngineList
Category	Mathematics/Fitting

Description

EngineList is a ArrayList of Engines generated by a Enginer.

EngineList is the main result of the NestedSampler. It contains all information to calculate averages, medians, modi or maximum likihood solutions of the parameters, or of any function of the parameters; in particular of the AbstractModel.

To make averages one has to take into account the weights. Each Engine has a weight and all weights sum to 1.0. So the average of any function f of the parameters p is

$E(f(p)) = \sum w_k f(p_k)$

where the sum is over all samples k.

A large set of utility functions is provided to extract the information from the EngineList.

See example for an script, where the EngineList is well used.

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.sample.EngineList

1.122. ERFC

Full Name:	herschel.ia.numeric.toolbox.basic.Erfc
Alias:	ERFC
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Erfc
Category	Mathematics/Statistics

Description

Returns the complementary error function of a number or array.

For more information on the complementary error function, see for instance the MathWorld website.

Example

Example 1: Apply ERFC to a Double1d

```
x = Double1d([-5.0,-1.0,-0.5,0.0,0.5,1.0,5.0])
erc = ERFC(x)
print erc
# [1.9999999999984626,1.8427007900291827,1.520499876068384,
#  1.0,0.4795001239316159,0.15729920997081737,1.5374597939680894E-12]
```

API Summary

Jython Syntax
<y>=ERFC(<x>)

Properties
Number or array x [INPUT, MANDATORY, default=no default value]
Number or array y [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Number or array x [INPUT, MANDATORY, default=no default value]
The value or values of which to compute the complementary error function.

Number or array y [OUTPUT, MANDATORY, default=no default value]
The complementary error function value or values.

See also

- ERF
- Developers Manual: herschel.ia.numeric.toolbox.basic.Erfc

1.123. ERF

Full Name:	herschel.ia.numeric.toolbox.basic.Erf
Alias:	ERF
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Erf
Category	Mathematics/Statistics

Description

Returns the error function of a number or array.

For more information on the error function, see for instance the MathWorld website.

Example

Example 1: Apply ERF to a Double1d

```
x = Double1d([-5.0,-1.0,-0.5,0.0,0.5,1.0,5.0])
er = ERF(x)
print er
# [-0.9999999999984626,-0.8427007900291826,-0.5204998760683841,
#  0.0,0.5204998760683841,0.8427007900291826,0.9999999999984626]
```

API Summary

Jython Syntax

```
&lt;y&gt;=ERF(&lt;x&gt;)
```

Properties

Number or array x [INPUT, MANDATORY, default=no default value]

Number or array y [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Number or array x [INPUT, MANDATORY, default=no default value]

The value or values of which to compute the error function.

Number or array y [OUTPUT, MANDATORY, default=no default value]

The error function value or values.

See also

- ERFC
- Developers Manual: herschel.ia.numeric.toolbox.basic.Erf

1.124. est_attitude_new

Full Name:	herschel.ia.toolbox.pointing.est_attitude_new
Type:	Jython Task -
Import:	from herschel.ia.toolbox.pointing import est_attitude_new
Category	Toolboxes/Pointing

Description

A new version of the function to estimate the star tracker attitude.

This version uses the improvement in the p-value with the star excluded with respect to the p-value with the star present to decide whether a star should be deemed bad and eliminated. The maximum of two stars which may be eliminated has been increased to five.

Example

Example 1: num_stars = 5

```
stars_brf    = Double2d(18,3)
stars_meq    = Double2d(18,3)
stars_id     = Int1d([1, 2, 3, 4, 5])
measerr_est  = 1.0e-5
prob_thresh  = 1.0e-4
prob_frac    = 100.0
for star in range(num_stars):
  stars_brf[star,:] =...
  stars_meq[star,:] =...
(strquat, loss, TASTE, prob, num_bad, id_bad, status) = \\
      est_attitude(num_stars, stars_brf, stars_meq, stars_id, measerr_est,
 prob_thresh, prob_fac)
```

API Summary

Jython Syntax

```
(strquat, loss, TASTE, prob, num_bad, id_bad, status) = \
est_attitude_new (num_stars, stars_brf, stars_meq, stars_id,
measerr_est, prob_thresh, prob_fac)
```

Properties

int **num_stars** [INPUT, MANDATORY, default=NO default value]
Double2d **stars_brf** [INPUT, MANDATORY, default=NO default value]
Double2d **stars_meq** [INPUT, MANDATORY, default=NO default value]
Int1d(18) **stars_id** [INPUT, MANDATORY, default=NO default value]
Double **measerr_est** [INPUT, MANDATORY, default=NO default value]
Double **prob_thresh** [INPUT, MANDATORY, default=NO default value]
Double **prob_fac** [INPUT, MANDATORY, default=NO default value]
Quaternion **strquat** [OUTPUT, MANDATORY, default=NO default value]
Double **loss** [OUTPUT, MANDATORY, default=NO default value]
Double **TASTE** [OUTPUT, MANDATORY, default=NO default value]
Double **prob** [OUTPUT, MANDATORY, default=NO default value]

Properties
Int **num_bad** [OUTPUT, MANDATORY, default=NO default value]
Int(5) **id_bad** [OUTPUT, MANDATORY, default=NO default value]
Double **status** [OUTPUT, MANDATORY, default=NO default value]

API details

Properties

int num_stars [INPUT, MANDATORY, default=NO default value]

No. of stars, N, with which to determine attitude

Double2d stars_brf [INPUT, MANDATORY, default=NO default value]

Measured star vectors (BRF)

Double2d stars_meq [INPUT, MANDATORY, default=NO default value]

Catalogue star vectors (MEQ2000)

Int1d(18) stars_id [INPUT, MANDATORY, default=NO default value]

IDs in star catalogue

Double measerr_est [INPUT, MANDATORY, default=NO default value]

Best estimate of star vector measurement error (1 sigma, rad.)

Double prob_thresh [INPUT, MANDATORY, default=NO default value]

Probability threshold used for deciding when fit is bad

Double prob_fac [INPUT, MANDATORY, default=NO default value]

A star is considered bad iff the p-value with the star excluded is more than prob_fac times greater than the p-value with it present

Quaternion strquat [OUTPUT, MANDATORY, default=NO default value]

Estimated star tracker attitude quaternion

Double loss [OUTPUT, MANDATORY, default=NO default value]

Minimized loss function, J(A*) (with a_k = 1/N) in RD.1

Double TASTE [OUTPUT, MANDATORY, default=NO default value]

Shuster's TASTE variable

Double prob [OUTPUT, MANDATORY, default=NO default value]

Probability of obtaining such a large value of TASTE at random

Int num_bad [OUTPUT, MANDATORY, default=NO default value]

Number of bad stars eliminated from estimation (0-5)

Int(5) id_bad [OUTPUT, MANDATORY, default=NO default value]

IDs of bad stars (max. 5)

Double status [OUTPUT, MANDATORY, default=NO default value]
Return status (0 = success, 1 = too few stars, 2 = other error)

History

- 2015-08-12 - CAS: ---------- Initial version

1.125. est_attitude

Full Name:	herschel.ia.toolbox.pointing.est_attitude
Type:	Jython Task -
Import:	from herschel.ia.toolbox.pointing import est_attitude
Category	Toolboxes/Pointing

Description

Estimate the star tracker attitude, eliminating bad stars if necessary.

At present this routine handles at most two bad stars (which should be sufficient).

Example

Example 1: num_stars = 5

```
stars_brf    = Double2d(18,3)
stars_meq    = Double2d(18,3)
stars_id     = Int1d([1, 2, 3, 4, 5])
measerr_est  = 1.0e-5
taste_thresh = 100
for star in range(num_stars):
  stars_brf[star,:] =...
  stars_meq[star,:] =...
(strquat, loss, TASTE, prob, num_bad, id_bad, status) = \\
      est_attitude(num_stars, stars_brf, stars_meq, stars_id, measerr_est,
 prob_thresh)
```

API Summary

Jython Syntax

```
(strquat, loss, TASTE, prob, num_bad, id_bad, status) = \
est_attitude (num_stars, stars_brf, stars_meq, stars_id, measer-
r_est, prob_thresh)
```

Properties
int **num_stars** [INPUT, MANDATORY, default=NO default value]
Double2d **stars_brf** [INPUT, MANDATORY, default=NO default value]
Double2d **stars_meq** [INPUT, MANDATORY, default=NO default value]
Int(18) **stars_id** [INPUT, MANDATORY, default=NO default value]
Double **measerr_est** [INPUT, MANDATORY, default=NO default value]
Double **prob_thresh** [INPUT, MANDATORY, default=NO default value]
Quaternion **strquat** [OUTPUT, MANDATORY, default=NO default value]
Double **loss** [OUTPUT, MANDATORY, default=NO default value]
Double **TASTE** [OUTPUT, MANDATORY, default=NO default value]
Double **prob** [OUTPUT, MANDATORY, default=NO default value]
Int **num_bad** [OUTPUT, MANDATORY, default=NO default value]
Int(2) **id_bad** [OUTPUT, MANDATORY, default=NO default value]
Double **status** [OUTPUT, MANDATORY, default=NO default value]

API details

Properties

int num_stars [INPUT, MANDATORY, default=NO default value]

No. of stars, N, with which to determine attitude

Double2d stars_brf [INPUT, MANDATORY, default=NO default value]

Measured star vectors (BRF)

Double2d stars_meq [INPUT, MANDATORY, default=NO default value]

Catalogue star vectors (MEQ2000)

Int(18) stars_id [INPUT, MANDATORY, default=NO default value]

IDs in star catalogue

Double measerr_est [INPUT, MANDATORY, default=NO default value]

Best estimate of star vector measurement error (1 sigma, rad.)

Double prob_thresh [INPUT, MANDATORY, default=NO default value]

Probability threshold used for deciding when fit is bad

Quaternion strquat [OUTPUT, MANDATORY, default=NO default value]

Estimated star tracker attitude quaternion

Double loss [OUTPUT, MANDATORY, default=NO default value]

Minimized loss function, J(A*) (with a_k = 1/N) in RD.1

Double TASTE [OUTPUT, MANDATORY, default=NO default value]

Shuster's TASTE variable

Double prob [OUTPUT, MANDATORY, default=NO default value]

Probability of obtaining such a large value of TASTE at random

Int num_bad [OUTPUT, MANDATORY, default=NO default value]

Number of bad stars eliminated from estimation (0, 1 or 2)

Int(2) id_bad [OUTPUT, MANDATORY, default=NO default value]

IDs of bad stars (max. 2)

Double status [OUTPUT, MANDATORY, default=NO default value]

Return status (0 = success, 1 = too few stars, 2 = other error)

History

- 2014-06-04 - CAS: Initial version
- 2014-06-06 - CAS: Import some much-needed modules.

- 2014-06-11 - CAS: HCSS-19366 Split the assignment of the index for the second candidate bad star into two steps (as Jython complains when an integer is added to a list).
- 2014-06-20 - CAS: HCSS-18566 Stars are now excluded based on a probability test rather than a direct test of TASTE. This is better as the probability also depends upon the number of stars used.
- 2014-11-17 - CAS: HCSS-19703 A little bit of a re-write to handle the cases where it is not HCSS-19764 possible to estimate the attitude (too few stars). Also noticed that the best value of TASTE was not being properly updated, so that the 'worst' star or pair of stars was not always being excluded.
- 2015-05-27 - CAS: HCSS-19398 Arrays resized to allow a maximum of 18 stars.
- 2015-08-12 - CAS: HCSS-19267 Input variable (and its description) modified.

1.126. EXP10

Full Name:	herschel.ia.numeric.toolbox.basic.Exp10
Alias:	EXP10
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Exp10
Category	Mathematics/General functions

Description

Returns the base 10 exponential function applied to a number or array.

The base 10 exponential function of a number *x* is 10 elevated to the power *x*. If the input is an array, the output is an array of the same size, with the computed exponentials instead of the original elements.

Example

Example 1: Applying EXP10 to a Double1d

```
x = Double1d([1,2,3,4])
print EXP10(x) #  [10.0, 100.0, 1000.0, 10000.0]
```

API Summary

Jython Syntax

```
&lt;y&gt; = EXP10(&lt;x&gt;)
```

Properties

Number or array **x** [INPUT, MANDATORY, default=no default value]

Number or array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Number or array x [INPUT, MANDATORY, default=no default value]

The value or values of which to compute the exponential. Complex numbers are not allowed.

Number or array y [OUTPUT, MANDATORY, default=no default value]

The exponential value or values.

See also

- LOG10
- EXP
- ExpN
- Developers Manual: herschel.ia.numeric.toolbox.basic.Exp10

1.127. EXP

Full Name:	herschel.ia.numeric.toolbox.basic.Exp
Alias:	EXP
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Exp
Category	Mathematics/General functions

Description

Returns the natural exponential function applied to a number or array.

The natural exponential function of a number *x* is Euler's number *e* elevated to the power *x*. If the input is an array, the output is an array of the same size, with the computed exponentials instead of the original elements.

Example

Example 1: Applying EXP to a Double1d

```
x = Double1d([0,1])
print LOG(EXP(x)) #  [0.0,1.0]
```

API Summary

Jython Syntax

```
&lt;y&gt; = EXP(&lt;x&gt;)
```

Properties

Number or array **x** [INPUT, MANDATORY, default=no default value]

Number or array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Number or array x [INPUT, MANDATORY, default=no default value]

The value or values of which to compute the exponential. Complex numbers are not allowed.

Number or array y [OUTPUT, MANDATORY, default=no default value]

The exponential value or values.

See also

- LOG
- EXP10
- ExpN
- Developers Manual: herschel.ia.numeric.toolbox.basic.Exp

1.128. ExpModel

Full Name:	herschel.ia.numeric.toolbox.fit.ExpModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import ExpModel
Category	Mathematics/Fitting

Description

Exponential Model.

$$f(x:p) = p_0 * \exp(p_1 * x)$$

where p_0 = amplitude, p_1 = slope and As always x = input.

The parameters are initialized at {1.0, -1.0}. It is a non-linear model.

Beware of a positive 2nd parameter. It is going off to Infinity quickly.

See example

Example

Example 1: ExpModel

```
em = ExpModel()              # exponential
print em.getNumberOfParameters()      # 2
```

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.ExpModel

1.129. ExpN

Full Name:	herschel.ia.numeric.toolbox.basic.ExpN
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import ExpN
Category	Mathematics/General functions

Description

Returns the base *n* exponential function applied to a number or array.

The base *n* exponential function of a number *x* is *n* elevated to the power *x*. If the input is an array, the output is an array of the same size, with the computed exponentials instead of the original elements.

Example

Example 1: Applying ExpN to a Double1d

```
x = Double1d([1,2,3,4])
print ExpN(2)(x) # [2.0,4.0,8.0,16.0]
```

API Summary

Jython Syntax

```
&lt;y&gt; = ExpN(&lt;base&gt;)(&lt;x&gt;)
```

Properties

Number or array **x** [INPUT, MANDATORY, default=no default value]

Number **base** [INPUT, MANDATORY, default=no default value]

Number or array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Number or array x [INPUT, MANDATORY, default=no default value]

The value or values of which to compute the exponential. Complex numbers are not allowed.

Number base [INPUT, MANDATORY, default=no default value]

The base of the exponential.

Number or array y [OUTPUT, MANDATORY, default=no default value]

The exponential value or values.

See also

- LogN
- EXP

- EXP10
- Developers Manual: herschel.ia.numeric.toolbox.basic.ExpN

1.130. ExponentialPrior

Full Name:	herschel.ia.numeric.toolbox.fit.ExponentialPrior
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import ExponentialPrior
Category	Mathematics/Fitting

Description

Exponential prior distribution.

Pr(x) = exp(-x / scale) By default scale = 1. The domain is [0,+Inf]

domain2unit: u = exp(- d / scale)

unit2domain: d = -log(u) * scale

By default: scale = 1.

Optionally one can set a zero fraction as the fraction of u where the unit2Domain method returns a 0.

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.ExponentialPrior

1.131. exportObservation

Full Name:	herschel.ia.toolbox.util.ExportObservationTask
Alias:	exportObservation
Type:	Java Task -
Import:	from herschel.ia.toolbox.util import ExportObservationTask
Category	Data access

Description

Exports observations from a pool to an HSA hierarchical directory structure.

The exportObservation task is used to export observations to an HSA hierarchical directory structure, so that they can be easily inspected, used and shared outside of HIPE.

Please note that only LocalStores and HsaReadPools can export their observations.

Example

Example 1: Export the first observation in "myPool" pool to the exportdir directory

```
mypool = PoolManager.getInstance().get('myPool')
exportObservation(pool=mypool,
 urn='urn:myPool:herschel.ia.obs.ObservationContext:0', dirout='exportdir',
 warn=False)
```

API Summary

Jython Syntax

```
exportObservation(&lt;pool&gt;, &lt;urn&gt;, &lt;dirout&gt; [,
&lt;warn&gt;=True])
```

Properties

- `ProductPool pool [INPUT, MANDATORY, default=No default value]`
- `String urn [INPUT, MANDATORY, default=No default value]`
- `String dirout [INPUT, MANDATORY, default=No default value]`
- `Boolean warn [INPUT, OPTIONAL, default=True]`

Limitations

- Only LocalStores and HsaReadPools can export their observations.
- By default, the dirout should not exist (you cannot safely export to the same dir in the general case), use `warn=False` if you want to export without being asked.
- This task will not delete any file it does not create itself.

API details

Properties

`ProductPool pool [INPUT, MANDATORY, default=No default value]`

The pool to export products from (should be a local store or HSA read pool)

String urn [INPUT, MANDATORY, default=No default value]
The urn that defines what to export

String dirout [INPUT, MANDATORY, default=No default value]
The directory where the observations will be exported to an HSA hierarchical directory structure. Will not be deleted but can be moved via moveTo.

Boolean warn [INPUT, OPTIONAL, default=True]
If true, asks confirmation if the dirout directory already exists and it is not empty.

See also

- Developers Manual: herschel.ia.toolbox.util.ExportObservationTask

History

- 2009-01-16 - JDS: initial version

1.132. exportSpectrumToAscii

Full Name:	herschel.ia.toolbox.spectrum.ExportSpectrumToAsciiTask
Alias:	exportSpectrumToAscii
Type:	Java Task -
Import:	from herschel.ia.toolbox.spectrum import ExportSpectrumToAsciiTask
Category	Spectra/Analysis

Description

Task for exporting spectrum data to ASCII.

The flux and wavescale data of the different spectra will end up in different columns of the ascii table. Furthermore, the task can be configured to export also flag and weight data. Note that in the PACS and SPIRE context, rather error and mask are used as opposed to flag and weight. The user will always find weights and flag in the output product (whether PACS, HIFI or SPIRE). Typically, flags and mask are equal whereas the weights will typically contain inverse square error. See, however, the implementation details of the data structures containing the spectra.

By default, the values in the output file will be comma-separated (*.csv). Further configuration possibilities are provided by the 'formatter' parameter.

Example

Example 1: export data to ASCII formats

```
global spectra # defined elsewhere
# export all the spectra found in the spectrum data structure:
exportSpectrumToAscii(ds=spectra, file="spectra.csv")
# export the segments with id's 0,1 of the spectra contained in the spectrum
 data structure:
exportSpectrumToAscii(ds=spectra, segments=[0,1], file="spectra.csv")
# export the point spectra with id's 1,4,5 found in the spectrum data structure
# (e.g. rows 1,4,5 in a HifiSpectrumDataset, or spaxels with index 1,4,5 in a
 cube)
# if there are multiple segments one can specify an array of tuples where the
 first index in the tuples refers to
# the point spectrum id. e.g. selection=[(2,1),(3,2)]
exportSpectrumToAscii(ds=spectra, selection=[0], file="spectra.csv")
# export all spectra to ASCII with space separated values
formatter = CsvFormatter()
formatter.delimiter = ' '
exportSpectrumToAscii(ds=spectra, file="spectra.csv", formatter=formatter)
```

API Summary

Properties
SpectrumContainer **ds** [INPUT, MANDATORY, default=no default value.]
Object **selection** [INPUT, OPTIONAL, default=no default value.]
PyArray **segments** [INPUT, OPTIONAL, default=no default value.]
String **file** [INPUT, OPTIONAL, default=False.]
Boolean **meta** [INPUT, OPTIONAL, default=False.]
Boolean **flags** [INPUT, OPTIONAL, default=False.]

Properties
Boolean **weights** [INPUT, OPTIONAL, default=False.]
Boolean **concat** [INPUT, OPTIONAL, default=False.]
AsciiFormatter **formatter** [INPUT, OPTIONAL, default=no default value.]

API details

Properties

SpectrumContainer ds [INPUT, MANDATORY, default=no default value.]

Spectrum data to be exported to ASCII. Examples of `SpectrumContainer` are `Spectrum1d, Spectrum2d, SpectralSimpleCube`.

Object selection [INPUT, OPTIONAL, default=no default value.]

Using the selection parameter you can specify a subset of individual spectra (from the given structure) that should be exported. In case it is not specified all spectra found in the spectrum data structure are exported. See the examples below and also the SelectSpectrumTask for how to specify selections.

PyArray segments [INPUT, OPTIONAL, default=no default value.]

Using the segments parameter you can specify the segments in the spectra to be exported. Note that PACS and SPIRE cubes only have a single segment so that this parameter will not be used for such data.

String file [INPUT, OPTIONAL, default=False.]

The location and name of the file to write the output to.

Boolean meta [INPUT, OPTIONAL, default=False.]

Specifies whether the meta data found in the input data is also exported (included as header information).

Boolean flags [INPUT, OPTIONAL, default=False.]

Specifies whether the the flags that can be specified on a per wavescale channel (in PACS / SPIRE spectra often referred to as mask) should also be exported.

Boolean weights [INPUT, OPTIONAL, default=False.]

Specifies whether the the weights that can be specified on a per wavescale channel (in PACS / SPIRE spectra often referred to as error) should also be exported.

Boolean concat [INPUT, OPTIONAL, default=False.]

Specifies whether segments are concatenated before exporting them. In general the spectra may be organised in several, possibly disjoint ranges. When setting the concat flag to True it means that all these segments are just glued. This may lead to frequencies not monotonously in- or decreasing or with gaps. Note that PACS and SPIRE cubes only have a single segment so that this parameter will not be used for such data.

AsciiFormatter formatter [INPUT, OPTIONAL, default=no default value.]

AsciiFormatter object that can be used to specify the value separator in the output file (see `AsciiTableWriterTask` for further details).

See also

- select
- Developers Manual: herschel.ia.toolbox.spectrum.ExportSpectrumToAsciiTask

History

- `2011-08-08 - melchior`: renamed from ExportSpectrumToASCII

1.133. extract

Full Name:	herschel.ia.toolbox.spectrum.ExtractSpectrumTask
Alias:	extract
Type:	Java Task -
Import:	from herschel.ia.toolbox.spectrum import ExtractSpectrumTask
Category	Spectra/Analysis

Description

Task for extracting a wavescale range (or ranges) from a spectrum or set of spectra.

It will select the specified range out of the input spectrum/a, wherever this range is within the input. Different possibilities are available for specifying the ranges (see the `ranges` -parameter below. Note that in presence of frequency drift, the ranges extracted from the different spectra may contain different number of bins (pixels, channels). To include that in a new spectrum data structure (of the same type as provided as input, a `SpectrumContainer` an 'enveloping' range is constructed so that the all the PointSpectra included in the result container have again the same width of the segments.

The number of frequency ranges specified in the `ranges` -parameter and the number of segments provided in the input data does not need to be the same. Each of the segments with a non-trivial overlap with one of the frequency intervals will lead to a segment in the result container. The spectra included in the input DO NOT need to entirely cover the specified range and they DO NOT need to have the all the same wavescale sampling - but, as part of a `SpectrumContainer` they DO need to have the same length.

The task should work with whatever unit the wavescale is expressed in (frequency, wavelength, wavenumber, velocity). The unit used to specify the `ranges` should however be consistent with the wavescale unit.

The range extraction can be restricted to a sub-selection of point spectra and/or segments. The parameters and values passed to these parameters are the same as used in the `SelectSpectrum` -task. See there for further details.

Example

Example 1: Extracting spectra from a SpectrumContainer

```
global spectra # defined elsewhere
slices = extract(ds=spectra,segments=[1,3])
# extract just some point spectra - keep the same wavescale ranges and segments
 of the input
slices = extract(ds=spectra, selection=[1,3,5,7])
# extract frequency ranges - from all the point spectra / segments found in the
 input spectra
# here providing first all the lower bounds and then all the upper bounds of
 the different intervals to be extrated
slices = extract(ds=spectra, ranges=([4100.0,4600.0,5100.0,6100.0,7100.0],
[4400.0,4900.0,5900.0,6900.0,7900.0]))
# extract frequency ranges - from all the point spectra / segments found in the
 input spectra
# here providing an array of intervals - each interval specified in a form
 <code>(lower,upper)</code>
slices = extract(ds=spectra,ranges=[(4100.0,4400.0),(4600.0,4900.0),
(5100.0,5900.0),(6100.0,6900.0),(7100.0,7900.0)])
# the same as above but now restricting to a sub-selection of point spectra
# (see <code>SelectSpectrum</code> for further examples of how to specify
 selections)
slices = extract(ds=spectra, \
                 ranges=[(4100.0,4400.0),(4600.0,4900.0),(5100.0,5900.0),
(6100.0,6900.0),(7100.0,7900.0)],selection=[0,1,2,3])
```

Example 1: Extracting spectra from a SpectrumContainer

```
# an alternative (deprecated) way to specify the selection:
min = Double1d([4100.0,4600.0,5100.0,6100.0,7100.0])
max = Double1d([4400.0,4900.0,5900.0,6900.0,7900.0])
slices = extract(ds=spectra, minFreq=min, maxFreq=max)
# for cubes you can specify the selection by spaxel coordinates - just pass a
 list of tuples (col,row)
# where 'col' is the column and 'row' the row index:
spectraOut = extract(ds=spectra, selection=[1,3,5,7])
```

API Summary

Properties
Object **ds** [INPUT, OPTIONAL, default=no default value.]
SpectrumContainer **result** [OUTPUT, OPTIONAL, default=no default value.]
Object **ranges** [INPUT, OPTIONAL, default=no default value.]
Double1d **minFreq** [INPUT, OPTIONAL, default=no default value.]
Double1d **maxFreq** [INPUT, OPTIONAL, default=no default value.]
Object **segments** [INPUT, OPTIONAL, default=no default value.]
Object **selection** [INPUT, OPTIONAL, default=None.]
PyDictionary **filter_meta** [INPUT, OPTIONAL, default=No default value.]

API details

Properties

Object ds [INPUT, OPTIONAL, default=no default value.]

Input data to be processed by the task. Several types are possible:

- SpectrumContainer
- Array of SpectrumContainer, i.e. SpectrumContainer[] (e.g. `[ds1,ds2,ds3]`)
- List of SpectrumContainer, i.e. List<SpectrumContainer>
- (Any product with implementations of `SpectrumContainer` inside.)

Examples of `SpectrumContainer` are `Spectrum1d, Spectrum2d, Spectral-SimpleCube`.

SpectrumContainer result [OUTPUT, OPTIONAL, default=no default value.]

The resulting container created by this task.

Object ranges [INPUT, OPTIONAL, default=no default value.]

Specify the range to be extracted - either by specifying ranges of channel numbers or frequency ranges:

- Type Range[]: Range of channel numbers for each sub-segment to be extracted.
- Type Range: Range of channel numbers to be extracted (if only one segment in the data).

Object ranges [INPUT, OPTIONAL, default=no default value.]

- Type PyList or PyTuples: Each PyTuple defines an interval on the frequency scale.
- Type PyTuple of two PyLists: The first list with the minima and the second with the maxima of the intervals on the frequency scale.

Double1d minFreq [INPUT, OPTIONAL, default=no default value.]

For each of the ranges to be extracted the minimum value (in the units the data have).

Double1d maxFreq [INPUT, OPTIONAL, default=no default value.]

For each of the ranges to be extracted the maximum value (in the units the data have).

Object segments [INPUT, OPTIONAL, default=no default value.]

Specify what segments to restrict on. There are two options available:

- Specify a PyList of segment indices or
- pass an instance of a SegmentSelection which gives the information on what segments for each point spectrum included in the container.

Object selection [INPUT, OPTIONAL, default=None.]

Specify what point spectra to restricted to. Different ways to specify these selections are possible:

- Specify a list of indices (in jython) of the point spectra for which the processing should be applied. For cubes, you can alternatively also pass a list of spaxel coordinates (a list of integer tuples `(row,col)`).
- Use a dictionary (in jython) to specify a selection model (lookup for attributes, filter attributes to be included in ranges or intervals).
- Pass a string representation of a jython list or a jython dictionary. The string will be parsed and the prcessing delegated to the two cases above.
- Pass any java instance that implements the SelectionModel interface (for the advanced user).

See the examples below or the SelectSpectrumTask for how to specify selections.

PyDictionary filter_meta [INPUT, OPTIONAL, default=No default value.]

Restrict the operation on spectrum containers with meta data that match given values. This only applies in case more than one container is passed as input data to the task.

See also

- SpectrumContainer
- extract
- select
- Developers Manual: herschel.ia.toolbox.spectrum.ExtractSpectrumTask

History

- `2011-08-08 - MM`: renamed from ExtractFreqRanges

- `2011-09-28 - MM`: add selection by spaxels - tuples (row,col).

1.134. extractRegionSpectrum

Full Name:	herschel.ia.toolbox.cube.ExtractRegionSpectrumTask
Alias:	extractRegionSpectrum
Type:	Java Task -
Import:	from herschel.ia.toolbox.cube import ExtractRegionSpectrumTask
Category	Data cubes/Analysis

Description

Extracts a spectrum from either the whole image, a single pixel, or an ellipsoid, rectangular or slit region.

Other regions are not yet available. The result can be a sum or average of the region. Fractional pixels at the border of the ellipse are accurately taken into account by computing the analytical integral of the part of the ellipse which covers the corresponding pixel. The integral values are used as weights.

API Summary

Properties
SimpleCube **cube** [INPUT, MANDATORY, default=No default value.]
String **regionType** [INPUT, MANDATORY, default=default WHOLE_IMAGE]
Double **centerCol** [INPUT, OPTIONAL, default=default None]
Double **centerRow** [INPUT, OPTIONAL, default=default None]
Double **radiusCol** [INPUT, OPTIONAL, default=default None]
Double **radiusRow** [INPUT, OPTIONAL, default=default None]
Double **row1** [INPUT, OPTIONAL, default=default None]
Double **col1** [INPUT, OPTIONAL, default=default None]
Double **row2** [INPUT, OPTIONAL, default=default None]
Double **col2** [INPUT, OPTIONAL, default=default None]
Double **width** [INPUT, OPTIONAL, default=default None]
String **arithmetics** [INPUT, OPTIONAL, default=default AVG]
Boolean **doAreaConversion** [INPUT, OPTIONAL, default=default false]
SimpleSpectrum **spectrum** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

SimpleCube cube [INPUT, MANDATORY, default=No default value.]

The Cube containing the spectra to extract

String regionType [INPUT, MANDATORY, default=default WHOLE_IMAGE]

Region type. One of WHOLE_IMAGE, ELLIPSE, SINGLE_PIXEL, RECTANGLE or SLIT

Double centerCol [INPUT, OPTIONAL, default=default None]

Column of center of ellipse (X). Also used in single pixel selection.

Double centerRow [INPUT, OPTIONAL, default=default None]

Row of center of ellipse (Y). Also used in single pixel selection.

Double radiusCol [INPUT, OPTIONAL, default=default None]

Radius of ellipse in the column direction (X) (pixels)

Double radiusRow [INPUT, OPTIONAL, default=default None]

Radius of ellipse in the row direction (Y) (pixels)

Double row1 [INPUT, OPTIONAL, default=default None]

Row (Y) of first coordinate of rectangle or slit

Double col1 [INPUT, OPTIONAL, default=default None]

Column (X) of first coordinate of rectangle or slit

Double row2 [INPUT, OPTIONAL, default=default None]

Row (Y) of second coordinate of rectangle or slit

Double col2 [INPUT, OPTIONAL, default=default None]

Column (X) of second coordinate of rectangle or slit

Double width [INPUT, OPTIONAL, default=default None]

Width of the slit in pixels

String arithmetics [INPUT, OPTIONAL, default=default AVG]

a value defining the kind of arithmetic to use for the resulting spectra: choice between SUM or AVERAGE.

Boolean doAreaConversion [INPUT, OPTIONAL, default=default false]

If true convert the flux from whatever unit (e.g. K or Jy) per pixel/spaxel/beam to the same unit per Steradian. This done by dividing the flux through the pixel/spaxel area in Steradian (obtained from WCS and converted to Steradian using 1 arcsec^2 = 2.35044e-11 Steradian). When this option is set, then the task assumes that the fluxes are per pixel/spaxel/beam, irrespective to the provided unit. "/sr" will be appended to the output unit (e.g. K becomes K/sr, Jy becomes Jy/sr).

SimpleSpectrum spectrum [OUTPUT, MANDATORY, default=no default value]

The spectrum extracted at the given position as a SimpleSpectrum

See also

- Developers Manual: herschel.ia.toolbox.cube.ExtractRegionSpectrumTask

1.135. Factor

Full Name:	herschel.ia.numeric.toolbox.xform.Factor
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.xform import Factor

Description

Contains a function that returns the prime factors of a given long integer.

See also

- Developers Manual: herschel.ia.numeric.toolbox.xform.Factor

1.136. Factory

Full Name:	herschel.ia.toolbox.fit.Factory
Type:	Java Class -
Import:	from herschel.ia.toolbox.fit import Factory

Description

Helper class for defining the GUI components for AccessDataFrameTask

See also

- Developers Manual: herschel.ia.toolbox.fit.Factory

History

- `2006-11-15 - DK`: creation.

1.137. fFT2d

Full Name:	herschel.ia.toolbox.image.FFT2dTask
Alias:	fFT2d
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import FFT2dTask
Category	Mathematics/Signal processing

Description

This is a task to calculate the Fast Fourier Transform and the power spectrum of an image.

This is a task to calculate the Fast Fourier Transform and the power spectrum of an image.

Example

Example 1: Example of how you can calculate the Fast Fourier Transform and the power spectrumof an image :

```
transform = fFT2d(image = myImage)
spectrum = fFT2d.spectrum
```

API Summary

Properties
Image **image** [INPUT, MANDATORY, default=None]
Complex2d **transform** [OUTPUT, MANDATORY, default=None]
Double2d **spectrum** [OUTPUT, MANDATORY, default=None]

API details

Properties

Image image [INPUT, MANDATORY, default=None]

This is the input image.

Complex2d transform [OUTPUT, MANDATORY, default=None]

This is the Fast Fourier Transform.

Double2d spectrum [OUTPUT, MANDATORY, default=None]

This is the power spectrum.

See also

- Developers Manual: herschel.ia.toolbox.image.FFT2dTask

1.138. FFT_AUTO

Full Name:	herschel.ia.numeric.toolbox.xform.FFT_AUTO
Alias:	FFT_AUTO
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.xform import FFT_AUTO
Category	Mathematics/Signal processing

Description

Gives the Fast Fourier Transform, automatically choosing what should generally be the fastest implementation.

The implementation is chosen by:

1. If length is 2^N, use FFT or IFFT
2. If the maximum prime factor of the length of the array is greater than 700, use FFT or IFFT
3. If the array is real only, has even length, and has even symmetry x[k] = x[n-k], use FFT_PACK_EVEN or IFFT_PACK_EVEN
4. If the array is real only (or imaginary only for the inverse), has even length, and has odd symmetry x[k] = -x[n-k] and x[0] = 0, use FFT_PACK_ODD or IFFT_PACK_ODD
5. If the maximum prime factor of the length of the array is greater than 500, use FFT or IFFT
6. If the array is real only, use RealDoubleFFT
7. If the maximum prime factor of the length of the array is greater than 250, use FFT or IFFT
8. Otherwise, use FFT_PACK

Note that all of the methods are adapted and normalised to give identical results.

The inverse transform is IFFT_AUTO. It transforms complex input to complex output. If input has length N then output has length N. The output of IFFT_AUTO is normalized: signal = IFFT_AUTO(FFT(signal))

Examples

Example 1: Apply FFT_AUTO

```
from java.lang.Math import PI
# Frequency modulated signal: parameters
ts = 1E-6          # Sampling period (sec)
fc = 200000        # Carrier frequency (Hz)
fm = 2000          # Modulation frequency (Hz)
beta = .0003       # Modulation index (Hz)
n = 5000           # Number of samples
# Create signal in complex form
t = Double1d.range(n) * ts
signal = SIN(2 * PI * fc * t * (1 + beta * COS(2 * PI * fm * t)))
z_signal=Complex1d(signal)
# Get spectrum
spectrum = ABS(FFT_AUTO(z_signal))
# Repeat with apodizing
z_signal=Complex1d(HAMMING(signal))
spectrum2 = ABS(FFT_AUTO(z_signal))
```

Example 2: Recreate signal with IFFT

```
# Create signal
N = 100
signal = Complex1d(Double1d.range(N), Double1d.range(N))
# Recreate signal from spectrum using IFFT
recreatedSignal = IFFT_AUTO(FFT_AUTO(signal))
# Print difference in signal and recreated signal
print "The maximum abs difference between the signal and recreated signal is:"
print MAX(ABS(signal-recreatedSignal))
```

API Summary

Jython Syntax
<y>=FFT_AUTO(<x>)

Properties
Complex1d **x** [INPUT, MANDATORY, default=no default value]
Complex1d **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Complex1d x [INPUT, MANDATORY, default=no default value]

Complex1d arrays only.

Complex1d y [OUTPUT, MANDATORY, default=no default value]

Returns a Complex1d

See also

- Section 5.3 in *Scripting Guide*
- FFT
- FFT_PACK
- RealDoubleFFT
- FFT_PACK_EVEN
- FFT_PACK_ODD
- Developers Manual: herschel.ia.numeric.toolbox.xform.FFT_AUTO

1.139. FFT_PACK_EVEN

Full Name:	herschel.ia.numeric.toolbox.xform.FFT_PACK_EVEN
Alias:	FFT_PACK_EVEN
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.xform import FFT_PACK_EVEN
Category	Mathematics/Signal processing

Description

Gives the Discrete Cosine Transform (DCT) of real data.

FFT_PACK_EVEN implements the Discrete Cosine Transform (DCT) FFT_PACK_EVEN transforms real input into real output. It is equivalent to the DFT of data with even symmetry of the form (x_0, x_1, ..., x_{N-2}, x_{N-1}, x_{N-2}, ..., x_1).

DCT(x_0, ...,x_{N-1}) is equivalent to FFT(x_0, x_1, ..., x_{N-2}, x_{N-1}, x_{N-2}, ..., x_1).

If input has length N then output has length N. FFT_PACK_EVEN is fast for input lengths for which the input length minus 1 can be decomposed into small factors.

Use FFT_PACK_EVEN to quickly calculate the FFT of evenly symmetric data. However, if the length of input minus 1 has large prime factors, use FFT instead.

The inverse transform is IFFT_PACK_EVEN. It transforms real input to real output. If input has length N then output has length N. Note that the output of IFFT_PACK_EVEN is not normalized. To normalize, divide by the size of the extended sequence: signal = IFFT_PACK_EVEN(FFT_PACK_EVEN(signal))/(signal.size*2-2)

Example

Example 1: Transform a symmetric double-sided interferogram into a spectrum using FFT_PACK and FFT_PACK_EVEN.

```
from java.lang.Math import PI
# Create double-sided interferogram with even symmetry and even length.
# For the DCT to be equivalent to the DFT, the extended signal must
# have even symmetry of the form (x0, x1, ....,x(N-2), x(N-1), x(N-2), ...,
 x1).
# That is: DCT(x0, ...,x(N-1)) equals DFT(x0, x1, ...., x(N-2), x(N-1),
 x(N-2), ..., x1).
# Note that this means that the extended (symmetric) sequence has even length.
dOpd = 0.25                 # Spatial interval between samples
wingLength = 100            # Number elements in each wing of the interferogram
c = 2*Math.PI*0.1           # Opd multiplier
amplitudeAtZpd = 10.0       # The amplitude of the sinc at opd =0
opd = Double1d.range(2*wingLength+2)*dOpd- wingLength*dOpd # Optical path
 difference abscissa
extendedSignal = SIN(opd.copy()*c)/opd # Sinc function
extendedSignal = extendedSignal*(amplitudeAtZpd /c)
extendedSignal[wingLength] = amplitudeAtZpd      # Set value at ZPD to 1.0
 (instead of NaN)
extendedSignal = SHIFT(extendedSignal,-wingLength) # Shift signal ZPd is the
 first element
# Calculate FT of extendedSignal using FFT (and discard negative frequencies)
spec_fft = FFT(Complex1d(extendedSignal))[0:(extendedSignal.size/2+1)]
# Calculate spectrum from half of the extended signal using FFT_EVEN
# (the other part of the signal is redundant given the DCT's implicit
 assumption of even symmetry)
signal = extendedSignal[:(wingLength+2)]
spec_even = FFT_PACK_EVEN(signal)
```

Example 1: Transform a symmetric double-sided interferogram into a spectrum using FFT_PACK and FFT_PACK_EVEN.

```
# Compare FFT output with FFT_PACK_EVEN output
print "Maximum difference between FFT and FFT_PACK_ODD:"
print MAX(ABS(spec_fft -spec_even))
# Recreate signal from spectrum using inverse DCT
# Note that the recreated signal is normalized by the length
# of the *extended* signal.
recreatedSignal = IFFT_PACK_EVEN(spec_even)/(extendedSignal.size)
print "Maximum absolute difference between signal and recreated signal:"
print MAX(ABS(signal-recreatedSignal))
# Compare extended signals (original versus one recreated from a spectrum)
extendedRecreatedSignal = SHIFT(recreatedSignal.append( \
    REVERSE(recreatedSignal[1:(recreatedSignal.size-1)])), -
recreatedSignal.size)
print "Maximum absolute difference between extended signal and extended
 recreated signal:"
print MAX(ABS(extendedSignal-extendedRecreatedSignal))
```

API Summary

Jython Syntax
<y> = FFT_PACK_EVEN(<x>)

Properties
Double1d **x** [INPUT, MANDATORY, default=no default value]
Double1d **y** [OUTPUT, MANDATORY, default=no default value]

Limitations

FFT_PACK_EVEN only transforms Double1d input.

Miscellaneous

The discrete cosine transformation produces output that assumes the input, if extended, has even symmetry. The DCT of an input of size N is equivalent to a discrete Fourier transform (DFT) of the same input extended (with even symmetry) to have 2N - 2 real numbers.

API details

Properties

Double1d x [INPUT, MANDATORY, default=no default value]

The input for the discrete cosine transform.

Double1d y [OUTPUT, MANDATORY, default=no default value]

The discrete cosine transform of the input array.

See also

- Section 5.3 in *Scripting Guide*
- FFT
- RealDoubleFFT

- FFT_PACK
- FFT_PACK_ODD
- FFT_AUTO
- Developers Manual: herschel.ia.numeric.toolbox.xform.FFT_PACK_EVEN

1.140. FFT_PACK_ODD

Full Name:	herschel.ia.numeric.toolbox.xform.FFT_PACK_ODD
Alias:	FFT_PACK_ODD
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.xform import FFT_PACK_ODD
Category	Mathematics/Signal processing

Description

Gives the Discrete Sine Transform (DST) of real data.

FFT_PACK_ODD transforms real input into imaginary output. It is equivalent to the DFT of data with odd symmetry of the form (0, x_0, x_1, ..., x_{N-1}, 0, -x_{N-1}, ..., -x_0).

DST(x_0, ...,x_{N-1}) is equivalent to -FFT (0, x_0, x_1, ..., x_{N-1}, 0, -x_{N-1}, ..., -x_1).

If input has length N then output has length N. FFT_PACK_ODD is fast for input lengths for which the input length plus 1 can be decomposed into small factors.

Use FFT_PACK_ODD to quickly calculate the FFT of data with odd symmetry. However, if the length of input plus 1 has large prime factors, use FFT instead.

The inverse transform is IFFT_PACK_ODD. It transforms imaginary input to real output. If input has length N then output has length N. Note that the output of IFFT_PACK_ODD is not normalized. To normalize, divide by the size of the extended sequence: signal = IFFT_PACK_ODD(FFT_PACK_ODD(signal))/(signal.size*2+2)

Example

Example 1: A comparison of the FFT of odd real input with the FFT_PACK_ODD of (approximately) half the input.

```
from java.lang.Math import PI
# Create an extended signal of beats with odd symmetry and even length.
# For the DST to be equivalent to the DFT, the extended signal must
# have odd symmetry of the form (0, x0, x1, ..., x(N-1), 0, -x(N-1), ..., -x0).
# Equivalence between DST and DFT: DST(x0, ...,x(N-1)) equals -DFT(0, x0,
 x1, ..., x(N-1), 0, -x(N-1), ..., -x0).
# Note that this means that the extended (odd) sequence has even length.
deltaX = 0.25                # Spatial interval between samples
N_extended = 60              # Length of extended signal (must have even
 length)
N = N_extended/2 -1          # Length of input to FFT_PACK_ODD
periods = 2                  # Number of periods in signal
beats = 20                   # Secondary oscillations
amplitude = 100.0            # amplitude of signal
x = Double1d.range(N_extended)*deltaX  # abscissa
extendedSignal = amplitude*SIN(2*PI*periods/N_extended/
deltaX*x)*COS(2*PI*periods*beats/deltaX/N_extended*x)
# Calculate FT of extendedSignal using FFT (discard zero and negative
 frequencies)
spec_fft = FFT(Complex1d(extendedSignal))[1:N+1]
# Calculate spectrum from half of the extended signal using FFT_ODD
# Note that the zeros (at element 0 and element N+1) from the extended signal
 are not used as input to the DST.
# A completely imaginary spectrum has no zero frequency term.
signal = extendedSignal[1:N+1]
spec_odd = FFT_PACK_ODD(signal)
# Compare FFT output with FFT_PACK_ODD output
print "Maximum sum between FFT and FFT_PACK_ODD:"
```

Example 1: A comparison of the FFT of odd real input with the FFT_PACK_ODD of (approximately) half the input.

```
print MAX(ABS(spec_fft.imag + spec_odd))
# Recreate signal from spectrum
# Note that the recreated signal is normalized by the length
# of the *extended* signal (which is 2*N+2)
recreatedSignal = IFFT_PACK_ODD(spec_odd)/(extendedSignal.size)
print "Maximum absolute difference between signal and recreated signal:"
print MAX(ABS(signal-recreatedSignal))
# Compare extended signals
extendedRecreatedSignal =
 Double1d([0]).append(recreatedSignal).append(0).append(-
REVERSE(recreatedSignal))
print "Maximum absolute difference between extended signal and extended
 recreated signal:"
print MAX(ABS(extendedSignal-extendedRecreatedSignal))
```

API Summary

Jython Syntax
<y> = FFT_PACK_ODD(<x>)

Properties
Double1d **x** [INPUT, MANDATORY, default=no default value]
Double1d **y** [OUTPUT, MANDATORY, default=no default value]

Limitations

FFT_PACK_ODD only transforms Double1d input.

Miscellaneous

The discrete sine transformation produces output that assumes the input, if extended, has odd symmetry. This DST with input size N is equivalent to a discrete Fourier transform (DFT) of the same input extended to have 2N + 2 real numbers with odd symmetry.

API details

Properties

Double1d x [INPUT, MANDATORY, default=no default value]

The input for the discrete sine transform.

Double1d y [OUTPUT, MANDATORY, default=no default value]

The discrete sine transform of the input array.

See also

- Section 5.3 in *Scripting Guide*
- FFT
- RealDoubleFFT
- FFT_PACK

- FFT_PACK_EVEN
- FFT_AUTO
- Developers Manual: herschel.ia.numeric.toolbox.xform.FFT_PACK_ODD

1.141. FFT_PACK

Full Name:	herschel.ia.numeric.toolbox.xform.FFT_PACK
Alias:	FFT_PACK
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.xform import FFT_PACK
Category	Mathematics/Signal processing

Description

Gives the Fast Fourier Transform of complex data using an FFT_PACK algorithm.

FFT_PACK transforms complex input into complex output. The forward transform of FFT_PACK is equivalent to that of FFT but it may require less time to execute.

If input has length N then output has length N. FFT_PACK is fast for input for which the input length can be decomposed into small factors.

Use FFT_PACK when the input has a length which can be decomposed into small factors. However, if the length of input has large prime factors, use FFT instead. Also, if the input length is a power of two, use FFT instead.

The inverse transform is IFFT_PACK. It transforms complex input to complex output. If input has length N then output has length N. Note that the output of IFFT_PACK is not normalized. To normalize, divide by the length of the input: signal = IFFT_PACK(FFT_PACK(signal))/(signal.size)

Examples

Example 1: Apply FFT_PACK to an asymmetric signal

```
from java.lang.Math import PI
# Frequency modulated signal: parameters
ts = 1E-6          # Sampling period (sec)
fc = 200000        # Carrier frequency (Hz)
fm = 2000          # Modulation frequency (Hz)
beta = .0003       # Modulation index (Hz)
n = 5000           # Number of samples
# Create signal in complex form
t = Double1d.range(n) * ts
signal = SIN(2 * PI * fc * t * (1 + beta * COS(2 * PI * fm * t)))
z_signal = Complex1d(signal)
# Get spectrum
spectrum = ABS(FFT_PACK(z_signal))
# Repeat with apodizing
z_signal = Complex1d(HAMMING(signal))
spectrum2 = ABS(FFT_PACK(z_signal))
```

Example 2: Show how to get equivalent results with FFT and FFT_PACK, IFFT and IFFT_PACK

```
N = 1000
signal = Complex1d(Double1d.range(N), Double1d.range(N))
# Transform signal using FFT and FFT_PACK
spec_fft = FFT(signal)
spec_fft_pack = FFT_PACK(signal)
# Compare results of transformations
print "The maximum absolute difference between FFT and FFT_PACK spectra: "
print MAX(ABS(spec_fft-spec_fft_pack))
# Recreate signal with inverse transforms
recreatedSignal_fft = IFFT(spec_fft)
```

Example 2: Show how to get equivalent results with FFT and FFT_PACK, IFFT and IFFT_PACK

```
# Normalize recreated signal from FFT_PACK for comparison with FFT
recreatedSignal_fft_pack = IFFT_PACK(spec_fft_pack)/N
# Compare results of inverse transformations
print "The maximum absolute difference signals recreated using IFFT and
 IFFT_PACK: "
print MAX(ABS(recreatedSignal_fft-recreatedSignal_fft_pack))
```

API Summary

Jython Syntax
`y = FFT_PACK(x)`

See also

- Section 5.3 in *Scripting Guide*
- FFT
- RealDoubleFFT
- FFT_PACK_EVEN
- FFT_PACK_ODD
- FFT_AUTO
- Developers Manual: herschel.ia.numeric.toolbox.xform.FFT_PACK

1.142. FFT

Full Name:	herschel.ia.numeric.toolbox.xform.FFT
Alias:	FFT
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.xform import FFT
Category	Mathematics/Signal processing

Description

Gives the Fast Fourier Transform.

Note that greater performance may be achieved using the FFT_PACK classes if input has odd or even symmetry, or if input is strictly real, or if the input length can be decomposed into small factors.

The inverse transform is IFFT. It transforms complex input to complex output. If input has length N then output has length N. The output of IFFT is normalized: signal = IFFT(FFT(signal))

Examples

Example 1: Apply FFT

```
from java.lang.Math import PI
# Frequency modulated signal: parameters
ts = 1E-6          # Sampling period (sec)
fc = 200000        # Carrier frequency (Hz)
fm = 2000          # Modulation frequency (Hz)
beta = .0003       # Modulation index (Hz)
n = 5000           # Number of samples
# Create signal in complex form
t = Double1d.range(n) * ts
signal = SIN(2 * PI * fc * t * (1 + beta * COS(2 * PI * fm * t)))
z_signal=Complex1d(signal)
# Get spectrum
spectrum = ABS(FFT(z_signal))
# Repeat with apodizing
z_signal=Complex1d(HAMMING(signal))
spectrum2 = ABS(FFT(z_signal))
```

Example 2: Recreate signal with IFFT

```
# Create signal
N = 100
signal = Complex1d(Double1d.range(N), Double1d.range(N))
# Recreate signal from spectrum using IFFT
recreatedSignal = IFFT(FFT(signal))
# Print difference in signal and recreated signal
print "The maximum abs difference between the signal and recreated signal is:"
print MAX(ABS(signal-recreatedSignal))
```

API Summary

Jython Syntax

```
<y> = FFT(<x>)
```

Properties

Complex1d **x** [INPUT, MANDATORY, default=no default value]

Properties
Double1d **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Complex1d x [INPUT, MANDATORY, default=no default value]
Complex1d arrays only.

Double1d y [OUTPUT, MANDATORY, default=no default value]
Returns a Double1d

See also

- Section 5.3 in *Scripting Guide*
- FFT_PACK
- RealDoubleFFT
- FFT_PACK_EVEN
- FFT_PACK_ODD
- FFT_AUTO
- Developers Manual: herschel.ia.numeric.toolbox.xform.FFT

1.143. filterSpectrum

Full Name:	herschel.ia.toolbox.cube.FilterSpectrumTask
Alias:	filterSpectrum
Type:	Java Task -
Import:	from herschel.ia.toolbox.cube import FilterSpectrumTask
Category	Spectra/Analysis

Description

A Task which apply a filter on a spectrum (Double1d).

API Summary

Properties
Double1d **rawSpectrum** [INPUT, MANDATORY, default=no default value]
String nodefault value **spectralUnit** [INPUT, MANDATORY, default=no default value]
Double1d **specIndex** [INPUT, MANDATORY, default=no default value]
Integer **sizeOfSpectrum** [INPUT, MANDATORY, default=no default value]
String **modelFilter** [INPUT, MANDATORY, default=no default value]
Integer **widthFilter** [INPUT, MANDATORY, default=no default value]
Double1d **filteredSpectrum** [OUTPUT, MANDATORY, default=no default value]
Double **maxValue** [OUTPUT, MANDATORY, default=no default value]
Integer **maxPosition** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Double1d rawSpectrum [INPUT, MANDATORY, default=no default value]

The spectralCube as a SimpleCube containing the spectral and velocities information

String nodefault value spectralUnit [INPUT, MANDATORY, default=no default value]

the unit of the spectrum as a string

Double1d specIndex [INPUT, MANDATORY, default=no default value]

The spectral values corresponding to each bin of the spectrum

Integer sizeOfSpectrum [INPUT, MANDATORY, default=no default value]

The size of the spectrum (number of bin)

String modelFilter [INPUT, MANDATORY, default=no default value]
The name of the model to use for the filter

Integer widthFilter [INPUT, MANDATORY, default=no default value]
the width of the filter (fwhm of the gaussian wiodth of the box car ...)

Double1d filteredSpectrum [OUTPUT, MANDATORY, default=no default value]
The filtered spectrum as a double1d array

Double maxValue [OUTPUT, MANDATORY, default=no default value]
The value of the maximum of the filtered signal

Integer maxPosition [OUTPUT, MANDATORY, default=no default value]
The position of the maximum of the filtered position

See also

- Developers Manual: herschel.ia.toolbox.cube.FilterSpectrumTask

1.144. FitFringeData

Full Name:	herschel.ia.toolbox.spectrum.standingwaves.FitFringeData
Type:	Java Class -
Import:	from herschel.ia.toolbox.spectrum.standingwaves import FitFringeData
Category	Spectra/Analysis

Description

Class that wraps Double1d's into a SpectralSegment.

It can be used to generate a data object used with the FitFringe and SmoothBaselie in the standingwaves package.

Example

Example 1: some examples using FitFringeData

```
ffData1  = FitFringeData(myFreq,myFlux)
ffData2  = FitFringeData(myFreq,myFlux,myFlag,myWeight)
# WbsSpectrumDataSet, 6 = point spectrum index, 0 = segment index,
# 0 = do not convert GHz to micron
ffData3 = FitFringeData(spectrumContainer,6,0,0)
```

API Summary

Properties
Double1d **freq** [INPUT, MANDATORY, default=no default value]
Double1d **flux** [INPUT, MANDATORY, default=no default value]
Int1d **flag** [INPUT, OPTIONAL, default=no default value]
Double1d **weight** [INPUT, OPTIONAL, default=no default value]
int **scindex** [INPUT, MANDATORY, default=point spectrum index]
int **segmentindex** [INPUT, MANDATORY, default=segment index]
Boolean **convertGHzToMicron** [INPUT, MANDATORY, default=convert GHz or MHz to micron]

API details

Properties

Double1d freq [INPUT, MANDATORY, default=no default value]

- 1d array of wavelength.

Double1d flux [INPUT, MANDATORY, default=no default value]

- 1d array of flux.

Int1d flag [INPUT, OPTIONAL, default=no default value]

- 1d array of flag.

Double1d weight [INPUT, OPTIONAL, default=no default value]

- 1d array of weight.

int scindex [INPUT, MANDATORY, default=point spectrum index]

int segmentindex [INPUT, MANDATORY, default=segment index]

Boolean convertGHzToMicron [INPUT, MANDATORY, default=convert GHz or MHz to micron]

See also

- Developers Manual: herschel.ia.toolbox.spectrum.standingwaves.FitFringeData

1.145. fitFringe

Full Name:	herschel.ia.toolbox.spectrum.standingwaves.FitFringeTask
Alias:	fitFringe
Type:	Java Task -
Import:	from herschel.ia.toolbox.spectrum.standingwaves import FitFringeTask
Category	Spectra/Analysis

Description

Task that fits and removes fringes from spectra.

Fringes are detected in wavenumber (1/micron) space, where they have a constant `wavelength', in cycles per wavenumber (C/Wn [==micron]).

To find fringes we fit a cosine function to the data f(n) = alfa.cos(p.n + phi) == A.cos(p.n) + B.sin(p.n) A, B and p are parameters to be estimated. p the the 'wavelength' of the fringe. For band 3 the main fringe is located at 3250 cycle/wavenumber. For a given p f(n) is a linear function which can easily be fitted to the data. So we search all wavelengths from CYCLE (=3000) to CYCLE+NCYCLE*CYSTEP (=3600), fit A and B and find the minimum in chi-sq.

In automatic mode we calculate the (bayesian) evidence for this (new) fringe. We keep the fringe if this evidence is larger than the evidence carried by the previous (set of) fringe(s). The process of identifying new fringes continues as long as the evidence increases with each new fringe. For all fringes the amplitudes are deterimined and the total fringe-set is subtracted. This is the advised mode.

In non-automatic mode fringes are extracted until the reduction in chi-sq is less than TOL or the number of fringes is larger than NFRINGES.

Before the fitting is started a smoothed version of the data is subtracted from itself, so that all the data are around zero. This smoothed version is at the end added in again. The smoothing, which is done in the frin_smooth procedure, is somewhat of an art; different options may have to be tried to get the best result.

We can get the evidence for the solution from the last value in the column `LogPr'. The evidence of solution for fringe 1 w.r.t fringe 2 equals 10 * (LogPr1 - LogPr2) dB.

Large outliers in the background subtracted data are given less weight, unless /WEIGHT is set. When there are still points that should not be taken along in the calculations they can be given zero weight with keywords SUPPRESS and SPUWID.

See Kester etal. 2001 "SWS Fringes and Models" in ISO Legacy Conference Proceedings.

For 3 reasons this procedure is superior to the use of FFTs. 1. The wavelength of the fringe can be estimated much better than in an FFT where it is limited to the Nyquist frequency, which in turn is determined by the number of datapoints. 2. It works on unevenly spaced data just as well as on a regularly gridded data set. Even when there are large gaps in the data it works fine. 3. It automatically determines how many fringe components are present, avoiding possible overfitting and underfitting.

Example

Example 1: Description

```
from herschel.ia.toolbox.spectrum.standingwaves import FitFringeData
myFreq = Double1d(freq_column.data)
myFlux = Double1d(flux_column.data)
myFlag = Int1d(flag_column.data)
```

Example 1: Description

```
myWeight = Double1d(weight_column.data)
ffData = FitFringeData(myFreq,myFlux,myFlag,myWeight)
improvedData = fitFringe(inputdata=ffData,nfringes=2)
baseline     = fitFringe.baseline
fringelist   = fitFringe.fringelist
```

API Summary

Jython Syntax

```
improvedData = fitFringe(myData,nfringes=2)
```

Properties

FitFringeData **data** [INPUT, MANDATORY, default=no default value]

Integer **nfringes** [INPUT, OPTIONAL, default="1"]

Boolean **mhz** [INPUT, OPTIONAL, default="false"]

Double **cycle** [INPUT, OPTIONAL, default="1100000.0"]

Double **cystep** [INPUT, OPTIONAL, default="9000.0"]

Integer **ncycle** [INPUT, OPTIONAL, default="450"]

Double **midcycle** [INPUT, OPTIONAL, default="1700000.0"]

Double **tolerance** [INPUT, OPTIONAL, default="0.0"]

Boolean **weight** [INPUT, OPTIONAL, default="false"]

Boolean **auto** [INPUT, OPTIONAL, default="false"]

Integer **plot** [INPUT, OPTIONAL, default="3(both)"]

Boolean **expert** [INPUT, OPTIONAL, default="false"]

Object **usermask** [INPUT, OPTIONAL, default="true"]

Double1d **fixfreq** [INPUT, OPTIONAL, default=no default value]

Boolean **automask** [INPUT, OPTIONAL, default="true"]

Object **flags** [INPUT, OPTIONAL, default=None]

FitFringeData **improvedData** [OUTPUT, MANDATORY, default=no default value]

FitFringeData **baseline** [OUTPUT, MANDATORY, default=no default value]

Double1d **mask** [OUTPUT, MANDATORY, default=no default value]

TableDataset **fringelist** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

FitFringeData data [INPUT, MANDATORY, default=no default value]

- Input data which contains four 1d-array of wavelength, flux, flag(optional), weight(optional).

Integer nfringes [INPUT, OPTIONAL, default="1"]

- Number of fringes to be removed.

Boolean mhz [INPUT, OPTIONAL, default="false"]

- Unit in mhz.

Double cycle [INPUT, OPTIONAL, default="1100000.0"]

- Start of cycle to search for fringes.

Double cystep [INPUT, OPTIONAL, default="9000.0"]

- Step in the cycles.

Integer ncycle [INPUT, OPTIONAL, default="450"]

- Number of cycles to check.

Double midcycle [INPUT, OPTIONAL, default="1700000.0"]

- Typical cycle frequency used for smoothing.

Double tolerance [INPUT, OPTIONAL, default="0.0"]

- Reduce the chisq until reduction is less than tol.

Boolean weight [INPUT, OPTIONAL, default="false"]

- Set all weights to 1.

Boolean auto [INPUT, OPTIONAL, default="false"]

- Automatic determination of the number of fringes to extract.

Integer plot [INPUT, OPTIONAL, default="3(both)"]

- Plot the result.

Boolean expert [INPUT, OPTIONAL, default="false"]

- Expert plot option.

Object usermask [INPUT, OPTIONAL, default="true"]

- Input frequencies of a set of wavelengths to disregard during fringe fitting (eg. at emission lines).

Double1d fixfreq [INPUT, OPTIONAL, default=no default value]

- Fix periods to these values.

Boolean automask [INPUT, OPTIONAL, default="true"]

- Automatically mask data points DEFAULT: TRUE. The automatically defined mask is added to any user-defined mask. - If set to FALSE, only the user-defined mask is used.

Object flags [INPUT, OPTIONAL, default=None]

Data points with these flag values will be excluded from the calculations. If no values are entered, ALL data points with non-zero flag values will be excluded.

FitFringeData improvedData [OUTPUT, MANDATORY, default=no default value]

- Fringe removed data.

FitFringeData baseline [OUTPUT, MANDATORY, default=no default value]

- Smoothed background.

Double1d mask [OUTPUT, MANDATORY, default=no default value]

- Output mask.

TableDataset fringelist [OUTPUT, MANDATORY, default=no default value]

- Table dataset which contains a cumulative list of extracted fringes. fringeNum - fringe number. cycle - fringe cycle frequency. cycle_in_MHz - fringe cycle in MHz. sinAmp - amplitude of sine component. cosAmp - amplitude of cosine component. chisq - chisq. chiRed - total chisq reduction.

See also

- Developers Manual: herschel.ia.toolbox.spectrum.standingwaves.FitFringeTask

1.146. FitsArchive

Full Name:	herschel.ia.io.fits.FitsArchive
Alias:	FitsArchive
Type:	Java Class -
Import:	from herschel.ia.io.fits import FitsArchive
Category	Input-output

Description

A class for reading and writing FITS files.

The FitsArchive provides a transparent way to store and retrieve Products as defined within the Herschel Data Processing software. FitsArchive is thread tolerant. It is not thread safe: do not share a FitsArchive object between several threads.

Examples

Example 1: default usage:

```
# write/read a Product in HCSS decorated format.
product = Product()
fits=FitsArchive()
fits.save(path,product)
product=fits.load(path)
```

Example 2: reading a HCSS FITS file which contains a removed class:

```
fits=FitsArchive()
product=fits.load(path,FitsArchive.HANDLE_MISSING_CLASSES)
```

Example 3: reading a externally generated FITS file into a Product:

```
# setup a new FitsArchive, but change the reader type
from herschel.ia.io.fits.reader.standard import StandardFitsReader
fits = FitsArchive(reader = StandardFitsReader())
product=fits.load(pathSrc)
# Saving the product will be done in HCSS decorated format.
fits.save(pathDst,product)
```

Example 4: reading a gzipped product:

```
fits=FitsArchive()
product=fits.load(path)
```

Example 5: reading a zipped product:

```
from java.util.zip import ZipFile, ZipEntry
fits = FitsArchive()
f = ZipFile(path)
entry = f.entries().nextElement() # the only entry
product = fits.load(f.getInputStream(entry))
```

Example 6: saving an empty ArrayDataset is allowed:

```
fits=FitsArchive()
ads=ArrayDataset()
product=Product("with empty ArrayDataset")
product['ads'] = ads
fits.save(pathDst, product) # works: image extension without image
```

Example 7: saving an empty column in a TableDataset is NOT allowed:

```
fits=FitsArchive()
tds=TableDataset()
tds.addColumn(Column())
product=Product("with empty Column in TableDataset")
product['tds'] = tds
try:
    fits.save(pathDst, product) # fails: binary table extensions require data
except:
    print "Exception raised"
pass
```

Example 8: saving the product minimising product metadata keyword translations and making keywords duplicity:

```
fits=FitsArchive()
product=Product()
fits.save(pathDst, product, 1, 1)
```

Example 9: saving a gzipped product

```
from java.util.zip import GZIPOutputStream
from java.io import FileOutputStream
product = Product()
fits=FitsArchive()
fits.save(GZIPOutputStream(FileOutputStream(pathDst)), product)
```

API Summary

Fields
FitsReader **HCSS_READER** HCSS FITS Reader
FitsReader **STANDARD_READER** Generic FITS Reader
FitsReader **RAW_READER**

Methods
Product **load** (String name) Loads a Product from a FITS file.
Product **load** (String name, boolean handleMissingClasses) Loads a Product from a FITS file.
Product **load** (String file) Loads a Product from a FITS file.
Product **load** (String file, boolean handleMissingClasses) Loads a Product from a FITS file.
Product **load** (InputStream is) Loads a Product from a FITS InputStream.
Product **load** (InputStream is, boolean handleMissingClasses) Loads a Product from a FITS InputStream.
save (String name, [Optionally derived] Product. product) Saves a Product to a FITS file.

Methods
save (boolean compress, String name, [Optionally derived] Product. product) Saves a Product to a FITS file.
save (String name, [Optionally derived] Product. product, Boolean. minimizeTranslations, [Optionally derived] Product. keysDuplicity) Saves a Product to a FITS file.
save (boolean compress, String name, [Optionally derived] Product. product, Boolean. minimizeTranslations, [Optionally derived] Product. keysDuplicity) Saves a Product to a FITS file.
save (File file, [Optionally derived] Product. product) Saves a Product to a FITS file.
save (boolean compress, File file, [Optionally derived] Product. product) Saves a Product to a FITS file.
save (File file, [Optionally derived] Product. product, Boolean. minimizeTranslations, [Optionally derived] Product. keysDuplicity) Saves a Product to a FITS file.
save (boolean compress, File file, [Optionally derived] Product. product, Boolean. minimizeTranslations, [Optionally derived] Product. keysDuplicity) Saves a Product to a FITS file.
save (OutputStream os, [Optionally derived] Product. product) Saves a Product to an OutputStream (creating a FITS file).
save (OutputStream os, [Optionally derived] Product. product, Boolean. minimizeTranslations, [Optionally derived] Product. keysDuplicity) Saves a Product to an OutputStream (creating a FITS file).
MetaData **loadMeta** (String name) Loads a MetaData product object from a FITS file.
MetaData **loadMeta** (File file) Loads a MetaData product object from a FITS file.
MetaData **loadMeta** (InputStream is) Loads a MetaData product object from a FITS file.
MetaData **loadMeta** (String name, integer hduIndex) Loads a MetaData object from a FITS file at the specified HDU index.
MetaData **loadMeta** (String name, integer hduIndex, integer commentExtraSpacesToRemove) Loads a MetaData object from a FITS file at the specified HDU index.
MetaData **loadMeta** (File file, integer hduIndex) Loads a MetaData object from a FITS file at the specified HDU index.
MetaData **loadMeta** (File file, integer hduIndex, integer commentExtraSpacesToRemove) Loads a MetaData object from a FITS file at the specified HDU index.
MetaData **loadMeta** (InputStream is, integer hduIndex)

Methods
Loads a MetaData object from a FITS file at the specified HDU index.
MetaData **loadMeta** (InputStream is, integer hduIndex, integer commentExtraSpacesToRemove)
Loads a MetaData object from a FITS file at the specified HDU index.
MetaData **loadMeta** (String name, integer hduIndex, boolean handleMissingClasses)
Loads a MetaData object from a FITS file at the specified HDU index.
MetaData **loadMeta** (String name, integer hduIndex, integer commentExtraSpacesToRemove, boolean handleMissingClasses)
Loads a MetaData object from a FITS file at the specified HDU index.
MetaData **loadMeta** (File file, integer hduIndex, boolean handleMissingClasses)
Loads a MetaData object from a FITS file at the specified HDU index.
MetaData **loadMeta** (File file, integer hduIndex, integer commentExtraSpacesToRemove, boolean handleMissingClasses)
Loads a MetaData object from a FITS file at the specified HDU index.
MetaData **loadMeta** (InputStream is, integer hduIndex, boolean handleMissingClasses)
Loads a MetaData object from a FITS file at the specified HDU index.
MetaData **loadMeta** (InputStream is, integer hduIndex, integer commentExtraSpacesToRemove, boolean handleMissingClasses)
Loads a MetaData object from a FITS file at the specified HDU index.

API Details

Fields

FitsReader **HCSS_READER**

HCSS FITS Reader
 	
A FITS reader that expects a FITS format that was produced by this archive implementation. This is a shared object among any FitsArchive instance. In a multithreaded environment, you should use a new instance of a reader.
 	
It can handle nested structures, derived datasets and quantities of the data within FITS.
 	
This is the default reader when you create a FitsArchive.
Examples
creating a new FitsArchive
`fits=FitsArchive() # default reader=FitsArchive.HCSS_READER`
creating a new FitsArchive
`fits=FitsArchive(reader=FitsArchive.HCSS_READER)`

FitsReader **STANDARD_READER**
Generic FITS Reader

***FitsReader* STANDARD_READER**

A generic FITS reader for FITS files that are not created by the HCSS FitsArchive. This is a shared object among any FitsArchive instance. In a multithreaded environment, you should use a new instance of a reader.

This reader can read FITS files that were generated by other software as long as it complies to the FITS standard and translates the contents into a Product.

Examples

Import data from an externally generated FITS file

```
fits=FitsArchive(reader=FitsArchive.STANDARD_READER)
product=fits.load(path) # Where path is e.g. "/home/joe/external.fits"
```

Reusing this FitsArchive but changing the reader

```
fits.reader=fits.HCSS_READER
product=fits.load(path) # Where path is e.g. "/home/joe/hcss.fits"
```

***FitsReader* RAW_READER**

Example

creating a new FitsArchive with a RawFitsReader

```
fits = FitsArchive()
fits.setReader(FitsArchive.RAW_READER)
rules = fits.getRules()
rules.setDictionary(None)
meta = fits.loadMeta(path)
```

Methods

***Product* load (String name)**

Loads a Product from a FITS file.

Loads a Product from a FITS file using the current reader.

Argument

`String` **name** `[INPUT, MANDATORY, default=no default value]`

Name of the FITS file

Return

Product

An object of the herschel.ia.dataset.Product family.

***Product* load (String name, boolean handleMissingClasses)**

Loads a Product from a FITS file.

Loads a Product from a FITS file using the current reader.

Arguments

`String` **name** `[INPUT, MANDATORY, default=no default value]`

Name of the FITS file

`boolean` **handleMissingClasses** `[INPUT, MANDATORY, default=no default value]`

For creating dummy classes when an HCSS class is not found.

Return

Product

Product load (String name, boolean handleMissingClasses)

An object of the herschel.ia.dataset.Product family.

Product load (String file)

Loads a Product from a FITS file.

Loads a Product from a FITS file using the current reader.

Argument

`String` **file** [INPUT, MANDATORY, default=no default value]

FITS file

Return

Product

An object of the herschel.ia.dataset.Product family.

Product load (String file, boolean handleMissingClasses)

Loads a Product from a FITS file.

Loads a Product from a FITS file using the current reader.

Arguments

`String` **file** [INPUT, MANDATORY, default=no default value]

FITS file

boolean **handleMissingClasses** [INPUT, MANDATORY, default=no default value]

For creating dummy classes when an HCSS class is not found.

Return

Product

An object of the herschel.ia.dataset.Product family.

Product load (InputStream is)

Loads a Product from a FITS InputStream.

Loads a Product from a FITS InputStream using the current reader.

Argument

InputStream **is** [INPUT, MANDATORY, default=no default value]

InputStream to the FITS file

Return

Product

An object of the herschel.ia.dataset.Product family.

Product load (InputStream is, boolean handleMissingClasses)

Loads a Product from a FITS InputStream.

Loads a Product from a FITS InputStream using the current reader. The user must close the InputStream.

Arguments

InputStream **is** [INPUT, MANDATORY, default=no default value]

InputStream to the FITS file

Product load (InputStream is, boolean handleMissingClasses)

boolean **handleMissingClasses** [INPUT, MANDATORY, default=no default value]

For creating dummy classes when an HCSS class is not found.

Return

Product

An object of the herschel.ia.dataset.Product family.

save (String name, [Optionally derived] Product. product)

Saves a Product to a FITS file.

Saves a Product to a FITS file with sufficient information to preserve the quantities of the data as well as the original dataset type and contents.

Arguments

String **name** [INPUT, MANDATORY, default=no default value]

Name of the FITS file

[Optionally derived] Product. **product** [INPUT, MANDATORY, default=no default value]

An object of the herschel.ia.dataset.Product family.

save (boolean compress, String name, [Optionally derived] Product. product)

Saves a Product to a FITS file.

Saves a Product to a FITS file with sufficient information to preserve the quantities of the data as well as the original dataset type and contents.

Arguments

boolean **compress** [INPUT, MANDATORY, default=no default value]

'true' for writing a gzip FITS file.

String **name** [INPUT, MANDATORY, default=no default value]

Name of the FITS file

[Optionally derived] Product. **product** [INPUT, MANDATORY, default=no default value]

An object of the herschel.ia.dataset.Product family.

save (String name, [Optionally derived] Product. product, Boolean. minimizeTranslations, [Optionally derived] Product. keysDuplicity)

Saves a Product to a FITS file.

Saves a Product to a FITS file with sufficient information to preserve the quantities of the data as well as the original dataset type and contents.

Arguments

String **name** [INPUT, MANDATORY, default=no default value]

Name of the FITS file

[Optionally derived] Product. **product** [INPUT, MANDATORY, default=no default value]

An object of the herschel.ia.dataset.Product family.

Boolean. **minimizeTranslations** [INPUT, MANDATORY, default=no default value]

save (String name, [Optionally derived] Product. product, Boolean. minimizeTranslations, [Optionally derived] Product. keysDuplicity)

Specifies if some keywords that are not in any dictionary will be transalted without using 'META' following some rules.

`[Optionally derived] Product. keysDuplicity [INPUT, MANDATORY, default=no default value]`

Specifies if a keyword will be translated to several FITS keywords.

save (boolean compress, String name, [Optionally derived] Product. product, Boolean. minimizeTranslations, [Optionally derived] Product. keysDuplicity)

Saves a Product to a FITS file.

Saves a Product to a FITS file with sufficient information to preserve the quantities of the data as well as the original dataset type and contents.

Arguments

`boolean compress [INPUT, MANDATORY, default=no default value]`

'true' for writting a gzip FITS file.

`String name [INPUT, MANDATORY, default=no default value]`

Name of the FITS file

`[Optionally derived] Product. product [INPUT, MANDATORY, default=no default value]`

An object of the herschel.ia.dataset.Product family.

`Boolean. minimizeTranslations [INPUT, MANDATORY, default=no default value]`

Specifies if some keywords that are not in any dictionary will be transalted without using 'META' following some rules.

`[Optionally derived] Product. keysDuplicity [INPUT, MANDATORY, default=no default value]`

Specifies if a keyword will be translated to several FITS keywords.

save (File file, [Optionally derived] Product. product)

Saves a Product to a FITS file.

Saves a Product to a FITS file with sufficient information to preserve the quantities of the data as well as the original dataset type and contents.

Arguments

`File file [INPUT, MANDATORY, default=no default value]`

FITS file

`[Optionally derived] Product. product [INPUT, MANDATORY, default=no default value]`

An object of the herschel.ia.dataset.Product family.

save (boolean compress, File file, [Optionally derived] Product. product)

Saves a Product to a FITS file.

Saves a Product to a FITS file with sufficient information to preserve the quantities of the data as well as the original dataset type and contents.

Arguments

save (boolean compress, File file, [Optionally derived] Product. product)

`boolean` **`compress`** `[INPUT, MANDATORY, default=no default value]`

'true' for writting a gzip FITS file.

`File` **`file`** `[INPUT, MANDATORY, default=no default value]`

FITS file

`[Optionally derived] Product.` **`product`** `[INPUT, MANDATORY, default=no default value]`

An object of the herschel.ia.dataset.Product family.

save (File file, [Optionally derived] Product. product, Boolean. minimizeTranslations, [Optionally derived] Product. keysDuplicity)

Saves a Product to a FITS file.

Saves a Product to a FITS file with sufficient information to preserve the quantities of the data as well as the original dataset type and contents.

Arguments

`File` **`file`** `[INPUT, MANDATORY, default=no default value]`

FITS file.

`[Optionally derived] Product.` **`product`** `[INPUT, MANDATORY, default=no default value]`

An object of the herschel.ia.dataset.Product family.

`Boolean.` **`minimizeTranslations`** `[INPUT, MANDATORY, default=no default value]`

Specifies if some keywords that are not in any dictionary will be transalted without using 'META' following some rules.

`[Optionally derived] Product.` **`keysDuplicity`** `[INPUT, MANDATORY, default=no default value]`

Specifies if a keyword will be translated to several FITS keywords.

save (boolean compress, File file, [Optionally derived] Product. product, Boolean. minimizeTranslations, [Optionally derived] Product. keysDuplicity)

Saves a Product to a FITS file.

Saves a Product to a FITS file with sufficient information to preserve the quantities of the data as well as the original dataset type and contents.

Arguments

`boolean` **`compress`** `[INPUT, MANDATORY, default=no default value]`

'true' for writting a gzip FITS file.

`File` **`file`** `[INPUT, MANDATORY, default=no default value]`

FITS file.

`[Optionally derived] Product.` **`product`** `[INPUT, MANDATORY, default=no default value]`

An object of the herschel.ia.dataset.Product family.

`Boolean.` **`minimizeTranslations`** `[INPUT, MANDATORY, default=no default value]`

Specifies if some keywords that are not in any dictionary will be transalted without using 'META' following some rules.

save (boolean compress, File file, [Optionally derived] Product. product, Boolean. minimizeTranslations, [Optionally derived] Product. keysDuplicity)

`[Optionally derived] Product. keysDuplicity [INPUT, MANDATORY, default=no default value]`

Specifies if a keyword will be translated to several FITS keywords.

save (OutputStream os, [Optionally derived] Product. product)

Saves a Product to an OutputStream (creating a FITS file).

Saves a Product to an OutputStream, creating a FITS file with sufficient information to preserve the quantities of the data as well as the original dataset type and contents. The OutputStream is closed by this method.

Arguments

`OutputStream os [INPUT, MANDATORY, default=no default value]`

OutputStream.

`[Optionally derived] Product. product [INPUT, MANDATORY, default=no default value]`

An object of the herschel.ia.dataset.Product family.

save (OutputStream os, [Optionally derived] Product. product, Boolean. minimizeTranslations, [Optionally derived] Product. keysDuplicity)

Saves a Product to an OutputStream (creating a FITS file).

Saves a Product to an OutputStream, creating a FITS file with sufficient information to preserve the quantities of the data as well as the original dataset type and contents. The OutputStream is closed by this method.

Arguments

`OutputStream os [INPUT, MANDATORY, default=no default value]`

OutputStream.

`[Optionally derived] Product. product [INPUT, MANDATORY, default=no default value]`

An object of the herschel.ia.dataset.Product family.

`Boolean. minimizeTranslations [INPUT, MANDATORY, default=no default value]`

Specifies if some keywords that are not in any dictionary will be transalted without using 'META' following some rules.

`[Optionally derived] Product. keysDuplicity [INPUT, MANDATORY, default=no default value]`

Specifies if a keyword will be translated to several FITS keywords.

***MetaData* loadMeta (String name)**

Loads a MetaData product object from a FITS file.

Loads a MetaData product object using the current reader.

Argument

`String name [INPUT, MANDATORY, default=no default value]`

Name of the FITS file

Return

***MetaData* loadMeta (String name)**

MetaData

A MetaData product object.

***MetaData* loadMeta (File file)**

Loads a MetaData product object from a FITS file.

Loads a MetaData product object using the current reader.

Argument

`File` **file** `[INPUT, MANDATORY, default=no default value]`

Name of the FITS file

Return

MetaData

A MetaData product object.

***MetaData* loadMeta (InputStream is)**

Loads a MetaData product object from a FITS file.

Loads a MetaData product object using the current reader. The user must close the InputStream.

Argument

`InputStream` **is** `[INPUT, MANDATORY, default=no default value]`

InputStream to the FITS file.

Return

MetaData

A MetaData product object.

***MetaData* loadMeta (String name, integer hduIndex)**

Loads a MetaData object from a FITS file at the specified HDU index.

Loads a MetaData object using the current reader for HDU index 0. Loads a MetaData object using STANDARD_READER for HDU index greater than 0.

Arguments

`String` **name** `[INPUT, MANDATORY, default=no default value]`

Name of the FITS file

`integer` **hduIndex** `[INPUT, MANDATORY, default=no default value]`

HDU index.

Return

MetaData

A MetaData object at the specified HDU index.

***MetaData* loadMeta (String name, integer hduIndex, integer commentExtraSpacesToRemove)**

Loads a MetaData object from a FITS file at the specified HDU index.

Loads a MetaData object using the current reader for HDU index 0. Loads a MetaData object using STANDARD_READER for HDU index greater than 0.

***MetaData* loadMeta (String name, integer hduIndex, integer commentExtraSpacesToRemove)**

Arguments

`String` **name** [INPUT, MANDATORY, default=no default value]

Name of the FITS file

`integer` **hduIndex** [INPUT, MANDATORY, default=no default value]

HDU index.

`integer` **commentExtraSpacesToRemove** [INPUT, MANDATORY, default=no default value]

Comment extra spaces to be removed.

Return

MetaData

A MetaData object at the specified HDU index.

***MetaData* loadMeta (File file, integer hduIndex)**

Loads a MetaData object from a FITS file at the specified HDU index.

Loads a MetaData object using the current reader for HDU index 0. Loads a MetaData object using STANDARD_READER for HDU index greater than 0.

Arguments

`File` **file** [INPUT, MANDATORY, default=no default value]

Name of the FITS file

`integer` **hduIndex** [INPUT, MANDATORY, default=no default value]

HDU index.

Return

MetaData

A MetaData object at the specified HDU index.

***MetaData* loadMeta (File file, integer hduIndex, integer commentExtraSpacesToRemove)**

Loads a MetaData object from a FITS file at the specified HDU index.

Loads a MetaData object using the current reader for HDU index 0. Loads a MetaData object using STANDARD_READER for HDU index greater than 0.

Arguments

`File` **file** [INPUT, MANDATORY, default=no default value]

Name of the FITS file

`integer` **hduIndex** [INPUT, MANDATORY, default=no default value]

HDU index.

`integer` **commentExtraSpacesToRemove** [INPUT, MANDATORY, default=no default value]

Comment extra spaces to be removed.

Return

MetaData

A MetaData object at the specified HDU index.

***MetaData* loadMeta (InputStream is, integer hduIndex)**

Loads a MetaData object from a FITS file at the specified HDU index.

Loads a MetaData object using the current reader for HDU index 0. Loads a MetaData object using STANDARD_READER for HDU index greater than 0. The user must close the InputStream.

Arguments

`InputStream` **`is`** `[INPUT, MANDATORY, default=no default value]`

InputStream to the FITS file.

`integer` **`hduIndex`** `[INPUT, MANDATORY, default=no default value]`

HDU index.

Return

MetaData

A MetaData object at the specified HDU index.

***MetaData* loadMeta (InputStream is, integer hduIndex, integer commentExtraSpacesToRemove)**

Loads a MetaData object from a FITS file at the specified HDU index.

Loads a MetaData object using the current reader for HDU index 0. Loads a MetaData object using STANDARD_READER for HDU index greater than 0. The user must close the InputStream.

Arguments

`InputStream` **`is`** `[INPUT, MANDATORY, default=no default value]`

InputStream to the FITS file.

`integer` **`hduIndex`** `[INPUT, MANDATORY, default=no default value]`

HDU index.

`integer` **`commentExtraSpacesToRemove`** `[INPUT, MANDATORY, default=no default value]`

Comment extra spaces to be removed.

Return

MetaData

A MetaData object at the specified HDU index.

***MetaData* loadMeta (String name, integer hduIndex, boolean handleMissingClasses)**

Loads a MetaData object from a FITS file at the specified HDU index.

Loads a MetaData object using the current reader for HDU index 0. Loads a MetaData object using STANDARD_READER for HDU index greater than 0.

Arguments

`String` **`name`** `[INPUT, MANDATORY, default=no default value]`

Name of the FITS file

`integer` **`hduIndex`** `[INPUT, MANDATORY, default=no default value]`

HDU index.

`boolean` **`handleMissingClasses`** `[INPUT, MANDATORY, default=no default value]`

For creating dummy classes when an HCSS class is not found.

Return

***MetaData* loadMeta (String name, integer hduIndex, boolean handleMissingClasses)**

MetaData

A MetaData object at the specified HDU index.

***MetaData* loadMeta (String name, integer hduIndex, integer commentExtraSpacesToRemove, boolean handleMissingClasses)**

Loads a MetaData object from a FITS file at the specified HDU index.

Loads a MetaData object using the current reader for HDU index 0. Loads a MetaData object using STANDARD_READER for HDU index greater than 0.

Arguments

`String` **name** `[INPUT, MANDATORY, default=no default value]`

Name of the FITS file

`integer` **hduIndex** `[INPUT, MANDATORY, default=no default value]`

HDU index.

`integer` **commentExtraSpacesToRemove** `[INPUT, MANDATORY, default=no default value]`

Comment extra spaces to be removed.

`boolean` **handleMissingClasses** `[INPUT, MANDATORY, default=no default value]`

For creating dummy classes when an HCSS class is not found.

Return

MetaData

A MetaData object at the specified HDU index.

***MetaData* loadMeta (File file, integer hduIndex, boolean handleMissingClasses)**

Loads a MetaData object from a FITS file at the specified HDU index.

Loads a MetaData object using the current reader for HDU index 0. Loads a MetaData object using STANDARD_READER for HDU index greater than 0.

Arguments

`File` **file** `[INPUT, MANDATORY, default=no default value]`

Name of the FITS file

`integer` **hduIndex** `[INPUT, MANDATORY, default=no default value]`

HDU index.

`boolean` **handleMissingClasses** `[INPUT, MANDATORY, default=no default value]`

For creating dummy classes when an HCSS class is not found.

Return

MetaData

A MetaData object at the specified HDU index.

***MetaData* loadMeta (File file, integer hduIndex, integer commentExtraSpacesToRemove, boolean handleMissingClasses)**

Loads a MetaData object from a FITS file at the specified HDU index.

***MetaData* loadMeta (File file, integer hduIndex, integer commentExtraSpacesToRemove, boolean handleMissingClasses)**

Loads a MetaData object using the current reader for HDU index 0. Loads a MetaData object using STANDARD_READER for HDU index greater than 0.

Arguments

`File` **file** [INPUT, MANDATORY, default=no default value]

Name of the FITS file

`integer` **hduIndex** [INPUT, MANDATORY, default=no default value]

HDU index.

`integer` **commentExtraSpacesToRemove** [INPUT, MANDATORY, default=no default value]

Comment extra spaces to be removed.

`boolean` **handleMissingClasses** [INPUT, MANDATORY, default=no default value]

For creating dummy classes when an HCSS class is not found.

Return

MetaData

A MetaData object at the specified HDU index.

***MetaData* loadMeta (InputStream is, integer hduIndex, boolean handleMissingClasses)**

Loads a MetaData object from a FITS file at the specified HDU index.

Loads a MetaData object using the current reader for HDU index 0. Loads a MetaData object using STANDARD_READER for HDU index greater than 0. The user must close the InputStream.

Arguments

`InputStream` **is** [INPUT, MANDATORY, default=no default value]

InputStream to the FITS file.

`integer` **hduIndex** [INPUT, MANDATORY, default=no default value]

HDU index.

`boolean` **handleMissingClasses** [INPUT, MANDATORY, default=no default value]

For creating dummy classes when an HCSS class is not found.

Return

MetaData

A MetaData object at the specified HDU index.

***MetaData* loadMeta (InputStream is, integer hduIndex, integer commentExtraSpacesToRemove, boolean handleMissingClasses)**

Loads a MetaData object from a FITS file at the specified HDU index.

Loads a MetaData object using the current reader for HDU index 0. Loads a MetaData object using STANDARD_READER for HDU index greater than 0. The user must close the InputStream.

Arguments

`InputStream` **is** [INPUT, MANDATORY, default=no default value]

InputStream to the FITS file.

`integer` **hduIndex** [INPUT, MANDATORY, default=no default value]

***MetaData* loadMeta (InputStream is, integer hduIndex, integer commentExtraSpacesToRemove, boolean handleMissingClasses)**

HDU index.

`integer` **`commentExtraSpacesToRemove`** `[INPUT, MANDATORY, default=no default value]`

Comment extra spaces to be removed.

`boolean` **`handleMissingClasses`** `[INPUT, MANDATORY, default=no default value]`

For creating dummy classes when an HCSS class is not found.

Return

MetaData

A MetaData object at the specified HDU index.

See also

- Developers Manual: herschel.ia.io.fits.FitsArchive

1.147. fitsReader

Full Name:	herschel.ia.toolbox.util.FitsReaderTask
Alias:	fitsReader
Type:	Java Task -
Import:	from herschel.ia.toolbox.util import FitsReaderTask
Category	Input-output

Description

Task to read and import arbitrary products in FITS files into HIPE.

Creates a "suitable" product (or dataset) from a FITS file, not necessarily from Herschel (it is capable of importing external complex products).

Currently it deals with images, (wave based) spectra, cubes and spectral cubes.

It supports compressed (zip and gzip) fits files.

Examples

Example 1: Import (or read) guessing the product stored in a FITS file

```
filepath = "path_to_file/filename"
product=fitsReader(file=filepath)
```

Example 2: Import (or read) the image stored in a FITS file

```
filepath = "path_to_file/filename"
product=fitsReader(file=filepath, fitsType=FitsReaderTask.FitsType.IMAGE)
```

Example 3: Same as above but using strings for the fitsType

```
filepath = "path_to_file/filename"
product=fitsReader(file=filepath, fitsType="image")
```

API Summary

Jython Syntax

```
data = fitsReader(&lt;file&gt; [, &lt;fitsType&gt;="guess"])
```

Properties

String **file** [INPUT, MANDATORY, default=no default value]
Object **fitsType** [INPUT, OPTIONAL, default="guess"]
Object **data** [OUTPUT, MANDATORY, default=null]

Limitations

- There is no importer for Spectra, thus this task can only import a small subset of.

API details

Properties

String file [INPUT, MANDATORY, default=no default value]

The path of the FITS file to be read (can be gzipped or zipped).

Object fitsType [INPUT, OPTIONAL, default="guess"]

The type of object the fits file contains (default is unknown).

Possible values (either FitsType or String):

- **FitsReaderTask.FitsType.UNKNOWN** or "guess": (default) try to guess the type of object
- **FitsReaderTask.FitsType.IMAGE** or "image": import as an Image
- **FitsReaderTask.FitsType.SPECTRUM** or "spectrum": import as a Spectrum
- **FitsReaderTask.FitsType.CUBE or "cube"** : import as a Cube
- **FitsReaderTask.FitsType.SPECTRAL_CUBE or "spectral cube"** : import as an Spectral Cube

Object data [OUTPUT, MANDATORY, default=null]

The Product (or dataset for spectra) read from the FITS file.

See also

- simpleFitsReader
- importImage
- importCube
- importSpectralCube
- Developers Manual: herschel.ia.toolbox.util.FitsReaderTask

History

- 2009-09-04 - JDS: initial release (only images can be displayed, graphical editors fail to show)
- 2009-10-22 - JDS: release complete (no general solution for Spectra in Herschel)
- 2010-08-11 - JDS: extended to compressed files (via streams)
- 2012-07-20 - JDS: replaces SimpleFitsReader,

1.148. FitterFunction

Full Name:	herschel.ia.toolbox.fit.FitterFunction
Alias:	FitterFunction
Type:	Java Class -
Import:	from herschel.ia.toolbox.fit import FitterFunction
Category	Mathematics/Fitting

Description

Specialization of RealFunction that fits some given data.

Mostly for use in the ia.numeric.toolbox.integr package.

Example

Example 1: Implicit fitting

```
x = Double1d([ 0,0.5,1,2, 3, 4, 5,  6,  10,      12])
y = Double1d([-3,  2,4,5,12,37,92,189,1237,2325])
f = FitterFunction(x, y, PolynomialModel(3))
i = SimpsonIntegrator(1, 10)
print i.integrate(f)  # 2668.499999999972
# Explicit fitting
x = Double1d([ 0,0.5,1,2, 3, 4, 5,  6,  10,      12])
y = Double1d([-3,  2,4,5,12,37,92,189,1237,2325])
model = SplinesModel(3,3,x)
fitter = Fitter(x, model)
fitter.fit(y)
f = FitterFunction(model)  # or f = FitterFunction(fitter)
# remaining code like before
i = SimpsonIntegrator(1, 10)
print i.integrate(f)  # 2668.499999999972
```

API Summary

Jython Syntax
<f>=FitterFunc- tion(<x>,<y>,<m>[,<c>])
<f>=FitterFunction(<m>)
<f>=FitterFunction(<F>)

Properties
Double1d of abscissas **x** [INPUT, MANDATORY, default=no default value]
Double1d of values **y** [INPUT, MANDATORY, default=no default value]
AbstractModel **m** [INPUT, MANDATORY, default=no default value]
Class of Fitter **c** [INPUT, OPTIONAL, default=Fitter.class]
Fitter **F** [INPUT, MANDATORY, default=no default value]
RealFunction **f** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Double1d of abscissas x [INPUT, MANDATORY, default=no default value]

Abscissas; it is only mandatory for the constructors where it appears in the synopsis.

Double1d of values y [INPUT, MANDATORY, default=no default value]

Values corresponding to the abscissas; it is only mandatory for 1st constructors.

AbstractModel m [INPUT, MANDATORY, default=no default value]

Unitialized model for 1st constructors, or already customized model for 2nd constructor.

Class of Fitter c [INPUT, OPTIONAL, default=Fitter.class]

Allows specifying the fitter class to be used in 1st constructor. FitterFunction provides some useful constants: LINEAR (for Fitter.class), AMOEBA (for AmoebaFitter.class) and LEVENBERG (for LevenbergMarquardtFitter.class).

Fitter F [INPUT, MANDATORY, default=no default value]

It is only mandatory for 3rd constructor.

RealFunction f [OUTPUT, MANDATORY, default=no default value]

Function object that fits the provided data with the specified model.

See also

- Developers Manual: herschel.ia.toolbox.fit.FitterFunction

1.149. Fitter

Full Name:	herschel.ia.numeric.toolbox.fit.Fitter
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import Fitter
Category	Mathematics/Fitting

Description

Fitter for linear models.

The Fitter class is to be used in conjunction with *Model classes.

The Fitter class and its descendants fit data to a model. Fitter itself is the variant for linear models, ie. models linear in its parameters.

For both linear and nonlinear models it holds that once the optimal estimate of the parameters is found, a variety of calculations is exactly the same: standard deviations, noise scale, evidence and model errors. They all derive more or less from the inverse Hessian matrix (aka the covariance matrix). All these calculations are in this Fitter class. Other Fitter classes relegate their calculation in these issues to this one.

An introduction to the use of fit package can be found here.

At least once have a look at the background on the organization and structure of the package. The fit/demo directory contains worked examples on almost all aspects of of the package.

Example

Example 1: Fitter (for models which are linear in its parameters.)

```
# assume x and y are Double1d data arrays:
x = Double1d.range(100)
y = Double1d( Int1d.range(100).divide( 4 ) )         # digitization noise
poly = PolynomialModel( 1 )            # line
fitter = Fitter( x, poly )
param = fitter.fit( y )
stdev = fitter.getStandardDeviation()    # stdevs on the parameters
chisq = fitter.getChiSquared()
scale = fitter.getScale()            # noise scale
yfit  = fitter.getResult()            # fitted values
yfit  = poly( x )                # fitted values (same as previous)
yband = fitter.monteCarloError()     # 1 sigma confidence region
```

API Summary

Constructor

`Fitter (NumericData input, AbstractModel model)`

Create a new Fitter, providing inputs and model.

Method

`fit (Double1d data, Double1d weights)`

Return model parameters fitted to the data, including weights.

Limitations

1. The Fitter does **not** work with limits.

2. The calculation of the evidence is an Gaussian approximation which is only exact for linear models with a fixed scale.

Miscellaneous

In case of problems look at the trouble shooting section.

API Details

Constructor

`Fitter (NumericData input, AbstractModel model)`

Create a new Fitter, providing inputs and model.

A Fitter class is defined by its model and the input vector (the independent variable). When a fit to another model and/or another input vector is needed a new object should be created.

Arguments

`NumericData` **`input`** `[INPUT, MANDATORY, default=no default value]`

- Array of independent input values

`AbstractModel` **`model`** `[INPUT, MANDATORY, default=no default value]`

- The model function to be fitted

Method

`fit (Double1d data, Double1d weights)`

Return model parameters fitted to the data, including weights.

For Linear models the matrix equation

$$H * p = \beta$$

is solved for p. H is the Hessian matrix ($D * w * D^T$) and β is the inproduct of the data with the D, design matrix.

$$\beta = y * w * D^T$$

Arguments

`Double1d` **`data`** `[INPUT, MANDATORY, default=no default value]`

The data vector to be fitted.

`Double1d` **`weights`** `[INPUT, OPTIONAL, default=no default value ]`

Weights pertaining to the data (= 1.0 / sigma^2)

Error

`IllegalArgumentException`

when data/weights contain a NaN

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.Fitter

1.150. FixedMask

Full Name:	herschel.ia.numeric.toolbox.mask.FixedMask
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.mask import FixedMask
Category	Mathematics/Masks

Description

FixedMask represents the more traditional mask form, where a mask is defined at a fixed bit offset position.

The class represents only the definition, and does not allocate any data array itself. Defining a single mask array requires a minimum of 8 bits per data element (if byte is used) up to 64 bits (if long is used). All integer types are supported (byte/short/int/long). However, masks can be combined in the same array, so if 64 masks were stored in a long, it would be fully packed, with no memory wastage. This also means that there is a practical limit of 64 on the number of masks that can be defined. This mask still makes perfect sense when exported to an external format. It also integrates well with standard numeric library functionality. Note that the mask data itself is not stored in this class.

Example

Example 1: Examples of FixedMask functionalities

```
#(1)Define some mask bits/setters
MASTER = FixedMask (0, "Master")
NOISY  = FixedMask (2, "Noisy")
#where 0 and 2 are bit offsets
#Since a long integer can only holds 64 bits,an exception will be thrown out
 if
#TOOBIG=FixedMask(100,"Too Big")
#This will trigger java.lang.IllegalArgumentException: bit offset must be &gt;=
 0 and less than 64, not 100
#
#(2)A mask (both single bit and muti-bit) can be defined as
MASK=FixedMask(10)
#(3)Mask bits can be printed as "&lt;name&gt; @ &lt;bit offset&gt; =
 &lt;value&gt;":
print MASTER
# Master @ 0 = 1
print NOISY
# Noisy @ 2 = 4
print MASK
#multi-bit @ -1 = 10
print NOISY.getName()
#Noisy
print NOISY.getValue()
#4L
#
#(4)Set/Unset mask data:
#A mask data x can be set/unset by different mask setters:
x = 0
x = NOISY.set (x)
print x
#4L
#
#(5)Check mask data:
#The status of mask data x can be checked for different mask setters:
print NOISY.isSet(x)
# True
print MASTER.isSet(x)
# False
# The mask checking can be ignored:
NOISY.setIgnore(1)
print NOISY.isSet(x)
```

Example 1: Examples of FixedMask functionalities

```
#False
# The mask checking can be resumed:
NOISY.setIgnore(0)
print NOISY.isSet(x)
#True
#
#(6)Use mask bits on a mask data array:
a = Int1d([1,2,3,4,5,6,7,8,9])
print a.where (MASTER)
#[0,2,4,6,8]  (Note. The method "where" returns the indices where the mask
 MASTER is set.)
print MASTER.isSet (a)
#[true,false,true,false,true,false,true,false,true]
print MASTER.unset (a)
#[0,2,2,4,4,6,6,8,8]
print MASTER.set (a)
#[1,3,3,5,5,7,7,9,9]
MASTER.setIgnore(1)
print MASTER.isSet(a)
#[false,false,false,false,false,false,false,false,false]
print a.where(MASTER)
#[]
MASTER.setIgnore(0)
#
#(7)More examples to show mask is set, is not set, and the combined masks are
 set:
b = Int1d.range(20)
#Check where Master is set
print b.where(MASTER.isSet(b))
#[1,3,5,7,9,11,13,15,17,19]
#Check where Noisy or Master are set
print b.where(NOISY.isSet(b) | MASTER.isSet(b))
#[1,3,4,5,6,7,9,11,12,13,14,15,17,19]
#
#(8)The mask data array can be in high dimension:
aa = Int2d([[1,2,3],[4,5,6]])
print NOISY.isSet(aa)
#[ [false,false,false], [true,true,true] ]
print aa.where(NOISY)
#[3,4,5]
#
#(9)Mask bits/setters can be combined:
M_or_N = MASTER|NOISY
print M_or_N
#Master | Noisy @ -1 = 5
b=Int1d([0,0,0,0,0,0])
print M_or_N.set(b)
#[5,5,5,5,5,5]
#which is equivalent to set MASTER and then NOISY
M_or_N.unset(b)
#
M_and_N=MASTER &amp;amp; NOISY
print M_and_N
#Master &amp;amp; Noisy @ -1 = 0
print M_and_N.set(b)
#[5,5,5,5,5,5]
#Nothing is set in b, because M_and_N's value is 0.
M_invert=~MASTER
print M_invert
#~Master @ -1 = -2
#(9)Convenience method for creating a combination mask by <em>or</em>ing the
 input masks.
# print FixedMask.combine(MASTER,NOISY)
# Master | Noisy @ -1 = 5
# multi_bit @ multi-bit = 10
# (10)Convenience method for creating a combination mask by specifying which
 masks to
# include and which to exclude.
#  The first argument is the masks to include and the 2nd the mask to exclude.
#  If the same mask is present in both it is excluded.
```

Example 1: Examples of FixedMask functionalities

```
print FixedMask.combine([MASTER,NOISY],[NOISY])
#Master @ 0 = 1
#
#End of jexample
```

See also

- Developers Manual: herschel.ia.numeric.toolbox.mask.FixedMask

History

- `2006-06-01 - SG`: Original prototype
- `2006-09-01 - SG`: second go not using enum, subclass AbstractArrayPredicate
- `2007-02-01 - SG`: third iteration
- `2007-07-01 - SG`: fourth iteration
- `2009-05-01 - SG`: SPR-6833, SCR-4734
- `2009-11-01 - SG`: SCR-6541: registration
- `2011-04-28 - YM`: HCSS-13018: formalisation of multi-bit mask behaviour
- `2012-03-14 - YM`: HCSS-15344: code review fixes

1.151. fixedSkyAperturePhotometry

Full Name:	herschel.ia.toolbox.image.FixedSkyAperturePhotometryTask
Alias:	fixedSkyAperturePhotometry
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import FixedSkyAperturePhotometryTask
Category	Astronomical utilities/Photometry

Description

This is a task for aperture photometry for a circular target aperture.

It is meant for cases where you have retrieved a value for the sky intensity from somewhere else. You need to specify the target location (i.e. the center of the circular aperture) either in pixel coordinates (with the *centerX* and *centerY* parameters) or sky coordinates (with the *centerRa* and *centerDec* parameters), and the value for the sky intensity. The latter is given in the same unit as the one attached to the image.

The algorithm integrates the flux within the circular target aperture, taking only the pixels into account that are not flagged (in case the image has no flag attached to it, all pixels within the aperture will be taken into account). The unit of the integrated flux is the unit of the image, multiplied with "pixel". So aperture photometry on an image in Jy/pixel will yield an integrated flux in Jy. For images in Jy/beam, the integrated flux will be expressed in Jy/beam*pixel, if you don't convert the image to Jy/pixel yourself beforehand. This can be done with the ConvertImageUnitTask. Additionally, if the input unit is MJy/sr, the output will still be in Jy (the task does the conversion).

In the output product you will find the value of the integrated target flux before and after background subtraction, as well as the number of pixels in the target aperture that were taken into account, and the error on these fluxes. The background subtracted target flux is calculated as the flux integrated over the target aperture (omitting the flagged pixels), minus the product of the fixed sky intensity value and the number of pixels that were taken into account to calculate the flux within the target aperture.

The calculation of the error on the fluxes is done in the same fashion as in the aper.pro routine of the IDL Astronomy Library. This kind of errors are suited for CCDs, but a new error calculation will be added in the future. We will now explain how the error is calculated for the moment. In general, three factors contribute to the error on the target flux :

- scatter in sky values
- random photon noise
- uncertainty in mean sky brightness

In case you are using a fixed value for the sky intensity, the error on the integrated target flux (with and without background subtracted) is the sqrt of the (absolute value of the) total flux in the target aperture, before subtraction of the background.

Example

Example 1: This is an example how you can use the

```
fixedSkyAperturePhotometry : result =
fixedSkyAperturePhotometry(image = myImage, centerX =
429.17, centerY=312.89, radiusArcsec = 5.0, sky = 0.0,
centroid = True) result = fixedSkyAperturePhotometry(image =
myImage, centerRa = "05:47:03.282",
```

Example 1: This is an example how you can use the

```
centerDec="-51:03:45.56", radiusPixels = 5.0, sky = 0.0)
```

API Summary

Properties
Image **image** [INPUT, MANDATORY, default=None This is the image on which]
Double **centerX** [INPUT, OPTIONAL, default=NaN This is the]
Double **centerY** [INPUT, OPTIONAL, default=NaN This is the]
String **centerRa** [INPUT, OPTIONAL, default="centerRa" This is the right]
String **centerDec** [INPUT, OPTIONAL, default="centerDec" This is]
Double **radiusPixels** [INPUT, OPTIONAL, default=NaN This is the target]
Double **radiusArcsec** [INPUT, OPTIONAL, default=NaN This the target radius]
boolean **centroid** [INPUT, OPTIONAL, default=false This parameter]
Double **sky** [INPUT, OPTIONAL, default=0.0 This is the value for the sky]
AperturePhotometryProduct **result** [OUTPUT, MANDATORY, default=None This]

API details

Properties

Image image [INPUT, MANDATORY, default=None This is the image on which]

to perform aperture photometry.

Double centerX [INPUT, OPTIONAL, default=NaN This is the]

x-pixel-coordinate of the target center.

Double centerY [INPUT, OPTIONAL, default=NaN This is the]

y-pixel-coordinate of the target center.

String centerRa [INPUT, OPTIONAL, default="centerRa" This is the right]

ascension of the target center. Using this only makes sense if the input image has a valid Wcs attached to it. The input format for the right ascension is "hh mm ss.s", "hh:mm:ss.s" or "hh.hh-h".

String centerDec [INPUT, OPTIONAL, default="centerDec" This is]

declination of the target center. Using this only makes sense if the input image has a valid Wcs attached to it. The input format for the declination is "[+/-]dd mm ss.s", "[+/-]dd:mm:ss.s" or "[+/-]dd.ddd".

Double radiusPixels [INPUT, OPTIONAL, default=NaN This is the target]

radius (i.e. the radius of the circular target aperture) in pixels.

Double radiusArcsec [INPUT, OPTIONAL, default=NaN This the target radius]

in arcsec. Using this only makes sense if the input image has a valid Wcs attached to it and the pixel scaling is the same along both axis (in absolute value).

boolean centroid [INPUT, OPTIONAL, default=false This parameter]

indicates whether centroiding on the target is needed.

Double sky [INPUT, OPTIONAL, default=0.0 This is the value for the sky]

intensity (in the same unit as the image). This value has been retrieved from somewhere else, but the origin is unimportant.

AperturePhotometryProduct result [OUTPUT, MANDATORY, default=None This]

is the result of the aperture photometry. Here you can find the flux in the target aperture (with and without backgroud contribution), the number of pixels in the aperture and the error on the flux. From the curve of growth the user can judge whether a good value for the target radius was chosen (under the assumption that the given sky value was accurate enough).

See also

- Developers Manual: herschel.ia.toolbox.image.FixedSkyAperturePhotometryTask

1.152. FixedSkyAperturePhotometryProduct

Full Name:	herschel.ia.dataset.image.FixedSkyAperturePhotometryProduct
Type:	Java Class -
Import:	from herschel.ia.dataset.image import FixedSkyAperturePhotometryProduct
Category	Astronomical utilities/Photometry

Description

This is a class to deal with the results of aperture photometry with a circular target aperture and a fixed value for the sky intensity.

This class is used to store the output of the {@link herschel.ia.toolbox.image.FixedSkyAperturePhotometryTask FixedSkyAperturePhotometryTask}. Aperture specific information is stored in metadata, all other data (results of calculations) are stored in TableDatasets.

Example

Example 1: Script demonstrating the use of a FixedSkyAperturePhotometryProduct.

```
# get hold of the pixel coordinates of the target center:
coords = product.getTargetCenterPixelCoordinates()
row = coords[1]
column = coords[0]
# get hold of the sky coordinates of the target center:
coords = product.getTargetCenterSkyCoordinates()
ra = coords[0]
dec = coords[1]
# get hold of the target radius in pixels:
print product.getTargetRadiusPixels()
# get hold of the target radius in arcsec:
print product.getTargetRadiusArcsec()
# get hold of the fixed value for the sky intensity:
print product.getSkyValue()
# get hold of the intensity unit (i.e. the unit of the image):
print product.getUnit()
# get hold of the (integrated) flux unit:
print product.getFluxUnit()
# get hold of the results table (you can save it with the asciiTableWriter):
table = product.getTable()
# get hold of the total flux in the target aperture incl. background (you can
 save it with the asciiTableWriter):
print product.getTargetPlusSkyTotal()
# get hold of the total flux in the target aperture (background subtracted):
print product.getTargetTotal()
# get hold of the curve of growth as a table (you can save it with the
 asciiTableWriter) :
table = product.getCurveOfGrowth()
# plot the curve of growth :
plot = PlotXY(product.getGrowthRadius(), product.getGrowthFlux())
plot.setXtitle("Target radius [pixels]");
plot.setYtitle("Target flux (sky subtr.) [" + product.getFluxUnit() + "]")
plot.getSubPlot(0).getLayer(0).getStyle().setLine(0)
```

API Summary

Constructor
FixedSkyAperturePhotometryProduct
The constructor of a new FixedSkyAperturePhotometryProduct.

Methods
setSkyValue (double sky) Setting the value for the sky intensity.
setResultsTable (Double2d resultsTable) Setting the results table.
double **getSkyValue** Returns the fixed value for the sky intensity.
double **getIntensityPerSkyPixel** Returns the fixed value for the sky intensity.
double **getTargetTotal** Returns the total target flux (sky subtracted).
double **getNbOfTargetPixels** Returns the number of target pixels.
double **getIntensityPerTargetPixel** Returns the intensity per target pixel (sky subtracted).
double **getTargetError** Returns the error on the target flux (sky subtracted).

API Details

Constructor

FixedSkyAperturePhotometryProduct

The constructor of a new FixedSkyAperturePhotometryProduct.

A new AperturePhotometryProduct is constructed.

Methods

setSkyValue (double sky)

Setting the value for the sky intensity.

Sets the sky value for this FixedSkyAperturePhotometryProduct to the given sky value.

Argument

double **sky** [INPUT, MANDATORY, default=no default value]

The given sky value, as double

setResultsTable (Double2d resultsTable)

Setting the results table.

Sets the results table for this AperturePhotometryProduct to the given table.

Argument

Double2d **resultsTable** [INPUT, MANDATORY, default=no default value]

The results table, as Double2d

***double* getSkyValue**

Returns the fixed value for the sky intensity.

***double* getSkyValue**

Returns the sky value for this FixedSkyAperturePhotometryProduct.

Return

double

The sky value for this FixedSkyAperturePhotometryProduct.

***double* getIntensityPerSkyPixel**

Returns the fixed value for the sky intensity.

Returns the intensity per pixel for the sky for this FixedSkyAperturePhotometryProduct.

Return

double

The intensity per pixel for the sky for this FixedSkyAperturePhotometryProduct.

***double* getTargetTotal**

Returns the total target flux (sky subtracted).

Returns the total flux for the target (sky subtracted) for this FixedSkyAperturePhotometryProduct.

Return

double

The total flux for the target (sky subtracted) for this FixedSkyAperturePhotometryProduct.

***double* getNbOfTargetPixels**

Returns the number of target pixels.

Returns the number of pixels for the target (sky subtracted) for this FixedSkyAperturePhotometryProduct.

Return

double

The number of pixels for the target (sky subtracted) for this FixedSkyAperturePhotometryProduct.

***double* getIntensityPerTargetPixel**

Returns the intensity per target pixel (sky subtracted).

Returns the intensity per pixel for the target (sky subtracted) for this FixedSkyAperturePhotometryProduct.

Return

double

The intensity per pixel for the target (sky subtracted) for this FixedSkyAperturePhotometryProduct.

***double* getTargetError**

Returns the error on the target flux (sky subtracted).

Returns the error on the target flux (sky subtracted) for this FixedSkyAperturePhotometryProduct.

Return

double **getTargetError**
double Returns the error for the target flux (sky subtracted) for this FixedSkyAperturePhotometryProduct.

See also

- Developers Manual: herschel.ia.dataset.image.FixedSkyAperturePhotometryProduct

1.153. FIX

Full Name:	herschel.ia.numeric.toolbox.basic.Fix
Alias:	FIX
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Fix
Category	Arrays and datasets/Manipulation

Description

For each element of an array, returns the largest integer less than or equal to the element.

Returns a long array. Only float or double arrays are allowed as input argument.

Example

Example 1: Apply FIX to a Float1d

```
x = Float1d([1.0,2.1,3.5,4.9])
print FIX(x) # [1,2,3,4]
# To change the returned array type into an Int1d array:
i = Int1d(FIX(x))
```

API Summary

Jython Syntax

```
&lt;y&gt;=FIX(&lt;x&gt;)
```

Properties

Float or double array **x** [INPUT, MANDATORY, default=no default value]

LongNd **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Float or double array x [INPUT, MANDATORY, default=no default value]

The input array.

LongNd y [OUTPUT, MANDATORY, default=no default value]

The output array.

See also

- CEIL
- FLOOR
- ROUND

- Developers Manual: herschel.ia.numeric.toolbox.basic.Fix

1.154. Flag

Full Name:	herschel.ia.dataset.image.Flag
Alias:	Flag
Type:	Java Class -
Import:	from herschel.ia.dataset.image import Flag
Category	Images

Description

A class to describe Flags for images (SimpleImage and SimpleCube).

This class represents different kinds of flags in n-dimensions. The Flag is defined as an ArrayDataset. The different flag types are defined in the metadata of the ArrayDataset, a Shortnd describes the flags. Only the "UNVALID" flag type is available after constructing the Flag.

Example

Example 1: A basic example on how to create a Flag for an SimpleImage with dimensions 310 x 200.

```
flag = Flag(200, 310)
# Only the "UNVALID" flag type is available after constructing the Flag.
# Adding an extra "GLITCH" flag type.
flag.addFlagType("GLITCH", "Flag for signals affected by a cosmic ray hit")
# Adding information about the flagged pixels.
# validFlagData is a Bool2d, with the same dimensions of the flag (310 x 200)
flag.setFlag("UNVALID", validFlagData)
flag.setFlag("GLITCH", glitchFlagData)
# Getting information on Flagged pixels
print flag.getFlag("GLITCH")
```

API Summary

Constructors
Flag The standard constructor for a Flag.
Flag (int k) Constructor for a 1 dimensional flag.
Flag (int k, int l) Constructor for a 2 dimensional flag.
Flag (int k, int l, int m) Constructor for a 3 dimensional flag.
Flag (int k, int l, int m, int n) A constructor for a four-dimensional flag of given dimensions.
Flag (int k, int l, int m, int n, int o) Constructor for a 5 dimensional flag.
Flag (Flag flag) The copy constructor.

Methods
Flag **copy** The copy method returns a new Flag which is a copy of the original flag.
addFlagType (String flagType, String description) Defines a new flag type.
String1d **getFlagTypes** Give a list with the defined flag types.
boolean **hasFlag** (String name) Checks if the flag exists.
AbstractArrayData **getFlag** (String flagType) Gives the flag for this specific flag type.
AbstractArrayData **getFlag** Gives the flag.
AbstractArrayData **getFlagAsShortnd** Gives the flag.
setFlag (String flagType, AbstractArrayData flag) Sets the flag for this specific flag type.
setFlag (String flagType, int index, boolean flag) Sets the flag of a certain pixel.
setFlag (String flagType, int row, int column, boolean flag) Sets the flag of a certain pixel.
setFlag (String flagType, int index1, int index2, int index3, boolean flag) Sets the flag of a certain pixel.
setFlag (String flagType, int index1, int index2, int index3, int index4, boolean flag) Sets the flag of a certain pixel.
setFlag (String flagType, int index1, int index2, int index3, int index4, int index5, boolean flag) Sets the flag of a certain pixel.

API Details

Constructors

Flag
The standard constructor for a Flag. A constructor for a flag, after construction, 1 flagtype is available : UNVALID

Flag (int k)
Constructor for a 1 dimensional flag. A constructor for a one-dimensional flag of given dimension. A Flag of k elements in 1 dimension is made. No pixels are masked out. After construction, 1 flagtype is available : UNVALID **Argument**

Flag (int k)

`int k [INPUT, MANDATORY, default=no default value]`

Number of elements in the flag, as int

Example

Typical example on how to create a one-dimensional Flag.

```
flag = Flag(100)
```

Flag (int k, int l)

Constructor for a 2 dimensional flag.

A constructor for a two-dimensional flag of given dimensions. A Flag of (k, l) elements in 2 dimensions is made. No pixels are masked out. After construction, 1 flagtype is available : UNVALID

Arguments

`int k [INPUT, MANDATORY, default=no default value]`

Number of rows in the flag, as int

`int l [INPUT, MANDATORY, default=no default value]`

Number of columns in the flag, as int

Example

Typical example on how to create a 2-dimensional Flag.

```
flag = Flag(100, 50)
```

Flag (int k, int l, int m)

Constructor for a 3 dimensional flag.

A constructor for a three-dimensional flag of given dimensions. A Flag of (k, l, m) elements in 3 dimensions is made. No pixels are masked out. After construction, 1 flagtype is available : UNVALID

Arguments

`int k [INPUT, MANDATORY, default=no default value]`

Depth of the flag, as int

`int l [INPUT, MANDATORY, default=no default value]`

Number of rows in the flag, as int

`int m [INPUT, MANDATORY, default=no default value]`

Number of columns in the flag, as int

Example

Typical example on how to create a 3-dimensional Flag.

```
flag = Flag(100, 50, 75)
```

Flag (int k, int l, int m, int n)

A constructor for a four-dimensional flag of given dimensions.

A Flag of (k, l, m, n) elements in 4 dimensions is made. No pixels are masked out. After construction, 1 flagtype is available : UNVALID

Arguments

`int k [INPUT, MANDATORY, default=no default value]`

Flag (int k, int l, int m, int n)

Number of elements in the first dimension of the flag, as int

`int l [INPUT, MANDATORY, default=no default value]`

Number of elements in the second dimension of the flag, as int

`int m [INPUT, MANDATORY, default=no default value]`

Number of elements in the third dimension of the flag, as int

`int n [INPUT, MANDATORY, default=no default value]`

Number of elements in the fourth dimension of the flag, as int

Example

Typical example on how to create a 4-dimensional Flag.

```
flag = Flag(100, 50, 75, 125)
```

Flag (int k, int l, int m, int n, int o)

Constructor for a 5 dimensional flag.

A constructor for a five-dimensional flag of given dimensions. A Flag of (k, l, m, n, o) elements in 5 dimensions is made. No pixels are masked out. After construction, 1 flagtype is available : UNVALID

Arguments

`int k [INPUT, MANDATORY, default=no default value]`

Number of elements in the first dimension of the flag, as int

`int l [INPUT, MANDATORY, default=no default value]`

Number of elements in the second dimension of the flag, as int

`int m [INPUT, MANDATORY, default=no default value]`

Number of elements in the third dimension of the flag, as int

`int n [INPUT, MANDATORY, default=no default value]`

Number of elements in the fourth dimension of the flag, as int

`int o [INPUT, MANDATORY, default=no default value]`

Number of elements in the fifth dimension of the flag, as int

Example

Typical example on how to create a 5-dimensional Flag.

```
flag = Flag(100, 50, 75, 125, 25)
```

Flag (Flag flag)

The copy constructor.

The copy constructor makes an exact copy of the flag.

Argument

`Flag flag [INPUT, MANDATORY, default=no default value]`

The flag to be copied, as Flag

Methods

***Flag* copy**

The copy method returns a new Flag which is a copy of the original flag.

Flag copy

Return

Flag

Dataset A new Flag which is a copy of the current Flag.

addFlagType (String flagType, String description)

Defines a new flag type.

Define a new flag type : a temporary flag type that can be used at the user's discretion in the processing. This method guarantees that the identifier used does not interfere with existing identifiers. There can be no more than 16 different flag types.

Arguments

String **flagType** [INPUT, MANDATORY, default=no default value]

The name of the flag type. Should be unique in the list of flag types defined on this dataset, as String

String **description** [INPUT, MANDATORY, default=no default value]

Textual description of the flag type, as String

Errors

ImageProductException

Exception thrown if the number of flagTypes exceed 16.

IllegalArgumentException

Exception thrown if the flagType is not unique.

String1d getFlagTypes

Give a list with the defined flag types.

A String1d with all defined flag types is returned.

Return

String1d

The list of the mask type identifiers

boolean hasFlag (String name)

Checks if the flag exists.

Returns true if the flag has the given flag type.

Argument

String **name** [INPUT, MANDATORY, default=no default value]

Name of the flag type to check, as String

Return

boolean

The list of the mask type identifiers Returns true if the flag has the given flag type.

AbstractArrayData getFlag (String flagType)

Gives the flag for this specific flag type.

Gives the flag for this specific flag type as an AbstractArrayData. Depending on the dimensions of the flag, a Boolnd is returned with the flag for a given flag type.

AbstractArrayData getFlag (String flagType)

Argument

String **flagType** [INPUT, MANDATORY, default=no default value]

The flag type to check, as String

Return

AbstractArrayData

A Boolnd (depending on the dimension of the flag), describing the flag of the given flagType.

Error

IllegalArgumentException

if the maskType is not known.

AbstractArrayData getFlag

Gives the flag.

Gives the flag as an AbstractArrayData. Depending on the dimensions of the flag, a Boolnd is returned with the flag.

Return

AbstractArrayData

A Boolnd (depending on the dimension of the flag), describing the flag

AbstractArrayData getFlagAsShortnd

Gives the flag.

Gives the flag as an AbstractArrayData. Depending on the dimensions of the flag, a Shortnd is returned with the flag.

Return

AbstractArrayData

A Shortnd (depending on the dimension of the flag), describing the flag

setFlag (String flagType, AbstractArrayData flag)

Sets the flag for this specific flag type.

Sets the flag for this specific flag type as an AbstractArrayData. A boolnd can be given for a flag type. If the boolnd does not have the same dimension as the dimension of the flag, an IllegalArgumentException is thrown.

Arguments

String **flagType** [INPUT, MANDATORY, default=no default value]

The flag type to set, as String

AbstractArrayData **flag** [INPUT, MANDATORY, default=no default value]

A Boolean array that is True where the flag is set, as AbstractArrayData

Error

IllegalArgumentException

if the flagType is not known.

setFlag (String flagType, int index, boolean flag)

Sets the flag of a certain pixel.

setFlag (String flagType, int index, boolean flag)

Set a certain flag of a certain flagType to a certain value

Arguments

`String` **flagType** [INPUT, MANDATORY, default=no default value]

The flag type to set, as String

`int` **index** [INPUT, MANDATORY, default=no default value]

The index to change, as int

`boolean` **flag** [INPUT, MANDATORY, default=no default value]

The new value for the flag, as boolean

setFlag (String flagType, int row, int column, boolean flag)

Sets the flag of a certain pixel.

Set a certain flag of a certain flagType to a certain value

Arguments

`String` **flagType** [INPUT, MANDATORY, default=no default value]

The flag type to set, as String

`int` **row** [INPUT, MANDATORY, default=no default value]

The row to change, as int

`int` **column** [INPUT, MANDATORY, default=no default value]

The column to change, as int

`boolean` **flag** [INPUT, MANDATORY, default=no default value]

The new value for the flag, as boolean

setFlag (String flagType, int index1, int index2, int index3, boolean flag)

Sets the flag of a certain pixel.

Set a certain flag of a certain flagType to a certain value

Arguments

`String` **flagType** [INPUT, MANDATORY, default=no default value]

The flag type to set, as String

`int` **index1** [INPUT, MANDATORY, default=no default value]

The first index to change, as int

`int` **index2** [INPUT, MANDATORY, default=no default value]

The second index to change, as int

`int` **index3** [INPUT, MANDATORY, default=no default value]

The third index to change, as int

`boolean` **flag** [INPUT, MANDATORY, default=no default value]

The new value for the flag, as boolean

setFlag (String flagType, int index1, int index2, int index3, int index4, boolean flag)

Sets the flag of a certain pixel.

Set a certain flag of a certain flagType to a certain value

Arguments

setFlag (String flagType, int index1, int index2, int index3, int index4, boolean flag)

String **flagType** [INPUT, MANDATORY, default=no default value]

The flag type to set, as String

int **index1** [INPUT, MANDATORY, default=no default value]

The first index to change, as int

int **index2** [INPUT, MANDATORY, default=no default value]

The second index to change, as int

int **index3** [INPUT, MANDATORY, default=no default value]

The third index to change, as int

int **index4** [INPUT, MANDATORY, default=no default value]

The fourth index to change, as int

boolean **flag** [INPUT, MANDATORY, default=no default value]

The new value for the flag, as boolean

setFlag (String flagType, int index1, int index2, int index3, int index4, int index5, boolean flag)

Sets the flag of a certain pixel.

Set a certain flag of a certain flagType to a certain value

Arguments

String **flagType** [INPUT, MANDATORY, default=no default value]

The flag type to set, as String

int **index1** [INPUT, MANDATORY, default=no default value]

The first index to change, as int

int **index2** [INPUT, MANDATORY, default=no default value]

The second index to change, as int

int **index3** [INPUT, MANDATORY, default=no default value]

The third index to change, as int

int **index4** [INPUT, MANDATORY, default=no default value]

The fourth index to change, as int

int **index5** [INPUT, MANDATORY, default=no default value]

The fifth index to change, as int

boolean **flag** [INPUT, MANDATORY, default=no default value]

The new value for the flag, as boolean

See also

- SimpleImage
- SimpleCube
- Developers Manual: herschel.ia.dataset.image.Flag

1.155. flagPixels

Full Name:	herschel.ia.toolbox.spectrum.FlagPixelsTask
Alias:	flagPixels
Type:	Java Task -
Import:	from herschel.ia.toolbox.spectrum import FlagPixelsTask
Category	Spectra/Analysis

Description

Task for flagging pixels in spectra.

By setting the 'mask' parameter (and possibly restricting the selection of point spectra and segments to consider for flagging) you specify the wavescale bins to be flagged. Optionally, you can specify a flag (by default `2 power 30` is used). Then, the task sets the flags by modifying the input data. Please be aware that when the task is applied to cubes, flags values above 32767 will not be honoured. Also note that a pre-condition for the task to work is that the data structure already contains flag values (also referred to as mask for SPIRE and PACS). To learn more about the meaning and usage of flags/masks for each instrument, please see the HIFI User Manual, the PACS Data Reduction Guide: Spectroscopy or the SPIRE Data Reduction Guide as appropriate.

You have different alternatives to specify a 'mask' with the pixel indices to be flagged. For the following, it is helpful, to understand the data structure defined with `SpectrumContainer`, `PointSpectrum` and `SpectralSegment`. Please see the associated URM entries.

1. General mask: For each segment within a specific point spectrum specify an entry in a dictionary, with keys given by tuples (i,j) (where i stands for the point spectrum index and j for the segment index) and values specifying the list of pixel to be flagged.

2. Uniform masks: For all the point spectra (possibly restricted by a suitable selection model - see the 'selection' parameter below) the same mask apply. This is given again by a dictionary where the keys specify the segment index (an integer) and the values specify the list of pixel indices to be flagged. Alternatively, you can directly specify a java.util.Map with Integer keys and herschel.ia.dataset.Selection as values. The values specified as Selection's contain the indices of the pixels to be flagged. Alternatively, instead of a Selection you can also specify an Int1d cointaining the indices or a Bool1d with 'true' at the positions the flag should be set.

3. Uniform masks, uniform for several segments: Specify a list of pixel indices to be flagged. This mask applies to all the segments and point spectra - possibly restricted by a suitable 'selection' and 'segments' indices.

As input spectra you can specify not only a single spectrum container but also a list of spectrum containers. Alternatively, you can specify a list of tuples with the first argument in the tuple specifying the spectrum container and the second argument the mask with the pixels to be flagged.

Example

Example 1: various ways to flag pixels in spectra

```
global spectra,  spectra1,  spectra2 # defined elsewhere
# For the following, assume 'spectra' to be a SpectrumContainer with several
 point spectra containing
# several segments.
# Set flag 2 in point spectrum 0 / segments 1 at the pixel indices 1,2,667 and
# in point spectrum 0 / segment 3 at the pixel indices 30,690 - see the
 'General mask' (1.) above.
flagPixels(ds=spectra, mask={(0,1):[1,20,667], (0,3):[30,690]}, flag=2)
```

Example 1: various ways to flag pixels in spectra

```
# Set flag 2 in segments 1 and 3 at the indices 1,2,667 and 30,690
 respectively.
# The same scheme is applied to all point spectra - see the 'Uniform mask' (2.)
 above.
flagPixels(ds=spectra, mask={1:[1,20,667], 3:[30,690]}, flag=2)
# Set flag 2 power 30 in segments 1 and 3 at the indices 1,2,667 and 30,690
 respectively.
# The same scheme is applied to all point spectra - see the 'Uniform mask' (2.)
 above.
flagPixels(ds=spectra, mask={1:[1,20,667], 3:[30,690]})
# Set flag 2 in segments 1 and 3 at the indices 1,2,667 and 30,690
 respectively.
# In contrast to the example above, here restricting to the point spectra with
 indices [0]
# (again a 'Uniform mask' (2.) combined with the selection parameter).
# For further ways of how to specify selections see e.g. the AverageSpectrum-
task
flagPixels(ds=spectra, mask={1:[1,20,667], 3:[30,690]}, flag=2, selection=[0])
# For spectra1 and spectra2 two SpectrumContainers:
# Set flag 2 at the pixels 1,20,667 in the segments 1,2,3 for spectra1 and
# for spectra2 in point spectrum 0 and segments 1 at the pixel indices 1,2,667
 and
# in point spectrum 0 and segment 3 at the pixel indices 30,690.
flagPixels(ds=[(spectra1,[1,20,667]),(spectra2,{(0,1):[1,20,667], (0,3):
[30,690]})], segments=[1,2,3], flag=2)
# With a single container you can just specify one tuple - which is equivalent
 to the very first example above:
flagPixels(ds=(spectra,{(0,1):[1,20,667], (0,3):[30,690]}), flag=2)
```

API Summary

Properties
Object **ds** [INOUT, MANDATORY, default=no default value.]
Object **mask** [INPUT, OPTIONAL, default=None.]
Object **selection** [INPUT, OPTIONAL, default=None.]
Object **segments** [INPUT, OPTIONAL, default=no default value.]
int **flag** [INPUT, OPTIONAL, default=2 up to the power of 30 = 1073741824.]
Boolean **setFluxToNaN** [INPUT, OPTIONAL, default=False.]
PyDictionary **filter_meta** [INPUT, OPTIONAL, default=No default value.]

API details

Properties

Object ds [INOUT, MANDATORY, default=no default value.]

The input container(s) in which pixels should be flagged. See examples below. Examples of `SpectrumContainer` are `Spectrum1d, Spectrum2d, SpectralSimpleCube`.

Object mask [INPUT, OPTIONAL, default=None.]

Specification of the pixels to flag. See the description above and the examples below.

Object selection [INPUT, OPTIONAL, default=None.]

Specification of what point spectra the flagging mask should be applied to. Different ways to specify these selections are possible:

Object selection [INPUT, OPTIONAL, default=None.]

- Specify a list of indices (in jython) of the point spectra for which the mask should be applied.
- Use a dictionary (in jython) to specify a selection model (lookup for attributes, filter attributes to be included in ranges or intervals).
- Pass a string representation of a jython list or a jython dictionary. The string will be parsed and the processing delegated to the two cases above.
- Pass any java instance that implements the SelectionModel interface (for the advanced user).

See the examples below and the SelectSpectrumTask for how to specify selections.

Object segments [INPUT, OPTIONAL, default=no default value.]

Specify what segments to restrict on. There are two options available:

- Specify a PyList of segment indices or
- pass an instance of a SegmentSelection which gives the information on what segments for each point spectrum included in the container.

int flag [INPUT, OPTIONAL, default=2 up to the power of 30 = 1073741824.]

The flag value to set for the specified pixel.

Boolean setFluxToNaN [INPUT, OPTIONAL, default=False.]

Boolean to indicate that the flux value of the pixels to flag should be set to NaN. This will typically remove the pixel from plots.

PyDictionary filter_meta [INPUT, OPTIONAL, default=No default value.]

Restrict the operation on spectrum containers with meta data that match given values. This only applies in case more than one container is passed as input data to the task.

See also

- select
- Developers Manual: herschel.ia.toolbox.spectrum.FlagPixelsTask

History

- `2011-08-08 - melchior`: renamed from FlagPixels

1.156. flagSaturatedPixelsCube

Full Name:	herschel.ia.toolbox.cube.FlagSaturatedPixelsCubeTask
Alias:	flagSaturatedPixelsCube
Type:	Java Task -
Import:	from herschel.ia.toolbox.cube import FlagSaturatedPixelsCubeTask
Category	Data cubes/Analysis

Description

A Task to flag saturated pixels in a cube.

API Summary

Properties
`Cube cube [INPUT, MANDATORY, default=No default value]`
`Double value [INPUT, MANDATORY, default=No default value]`
`Cube result [OUTPUT, MANDATORY, default=No default value]`

API details

Properties

`Cube cube [INPUT, MANDATORY, default=No default value]`

The cube.

`Double value [INPUT, MANDATORY, default=No default value]`

The value of saturation.

`Cube result [OUTPUT, MANDATORY, default=No default value]`

The resulting cube.

See also

- Developers Manual: herschel.ia.toolbox.cube.FlagSaturatedPixelsCubeTask

1.157. flagSaturatedPixels

Full Name:	herschel.ia.toolbox.image.FlagSaturatedPixelsTask
Alias:	flagSaturatedPixels
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import FlagSaturatedPixelsTask
Category	Images/Analysis

Description

This is a task to flag saturated pixels in an image.

This is a task to flag saturated pixels in an image. You must only specify the value at which saturation occurs. All pixels for which the intensity value is not smaller than this value will be flagged with the SATURATED flag.

API Summary

Properties
`Image image [INPUT, MANDATORY, default=None]`
`Double value [INPUT, OPTIONAL, default=0.0]`
`Image flaggedImage [OUTPUT, MANDATORY, default=None]`

API details

Properties

`Image image [INPUT, MANDATORY, default=None]`

This is the input image.

`Double value [INPUT, OPTIONAL, default=0.0]`

This is the value at which saturation occurs.

`Image flaggedImage [OUTPUT, MANDATORY, default=None]`

This is the resulting image in which all saturated pixels have been flagged with the SATURATED flag.

See also

- Developers Manual: herschel.ia.toolbox.image.FlagSaturatedPixelsTask

1.158. Float1d

Full Name:	herschel.ia.numeric.Float1d
Type:	Java Class -
Import:	from herschel.ia.numeric import Float1d
Category	Arrays and datasets

Description

A rectangular numeric float array of rank 1.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Float1d

1.159. Float2d

Full Name:	herschel.ia.numeric.Float2d
Type:	Java Class -
Import:	from herschel.ia.numeric import Float2d
Category	Arrays and datasets

Description

A rectangular numeric float array of rank 2.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Float2d

1.160. Float3d

Full Name:	herschel.ia.numeric.Float3d
Type:	Java Class -
Import:	from herschel.ia.numeric import Float3d
Category	Arrays and datasets

Description

A rectangular numeric float array of rank 3.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Float3d

1.161. Float4d

Full Name:	herschel.ia.numeric.Float4d
Type:	Java Class -
Import:	from herschel.ia.numeric import Float4d
Category	Arrays and datasets

Description

A rectangular numeric float array of rank 4.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Float4d

1.162. Float5d

Full Name:	herschel.ia.numeric.Float5d
Type:	Java Class -
Import:	from herschel.ia.numeric import Float5d
Category	Arrays and datasets

Description

A rectangular numeric float array of rank 5.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Float5d

1.163. FLOOR

Full Name:	herschel.ia.numeric.toolbox.basic.Floor
Alias:	FLOOR
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Floor
Category	Mathematics/Nearest integer

Description

Returns the largest integer less than or equal to the input.

If the input is an array, the output is an array of the same type and size, with the computed values instead of the original elements.

Example

Example 1: Applying FLOOR to a Float1d

```
x = Float1d([0.1, 0.5, 0.9])
print FLOOR(x) # [0.0,0.0,0.0]
```

API Summary

Jython Syntax

```
&lt;y&gt; = FLOOR(&lt;x&gt;)
```

Properties

Number or array **x** [INPUT, MANDATORY, default=no default value]

Number or array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Number or array x [INPUT, MANDATORY, default=no default value]

The value or values of which to compute the floor. It cannot be a Complex array.

Number or array y [OUTPUT, MANDATORY, default=no default value]

The floor value or values.

See also

- CEIL
- ROUND
- Developers Manual: herschel.ia.numeric.toolbox.basic.Floor

1.164. fold

Full Name:	herschel.ia.toolbox.spectrum.FoldSpectrumTask
Alias:	fold
Type:	Java Task -
Import:	from herschel.ia.toolbox.spectrum import FoldSpectrumTask
Category	Spectra/Analysis

Description

Task for folding frequency-switched spectra.

The folded spectra are constructed by averaging the original spectra with a shifted and inverted copy. This algorithm corresponds to the most simple scheme described in article {\it "Recovering line profiles from frequency-switched spectra}, H.Liszt, Astron. Astrophys. Suppl. Ser. 124, 183-188 (1997)}. In addition to modifying the segment data the integration time is doubled to reflect the total integration time from the two overlayed phases. It is assumed that the integration time is available as an attribute named 'integration time'. By default, the task modifies the input spectra. Note that the operation is performed on a per segment basis. For that reason you may consider stitching the segments before folding.

Example

Example 1: res=fold(ds=spectra, freqThrow=-200., overwrite=False) # unit implicit --> the same as of the wave scale

```
res=fold(ds=spectra, freqThrow=-200., unit="MHz", overwrite=False)
res=fold(ds=spectra, freqThrow=-0.2, unit="GHz", overwrite=False)
# shift the resulting spectra by minus half the throw distance to the right
res=fold(ds=spectra, freqThrow=-200, overwrite=False, shift=True)
# Take the throw found in the dataset.
res=fold(ds=spectra, overwrite=False) # by default look for a 'LoThrow'-
attribute in the data.
res=fold(ds=spectra, freqThrow="loThrow", overwrite=False)
```

API Summary

Properties
SpectrumContainer **ds** [INPUT, MANDATORY, default=no default value.]
SpectrumContainer **result** [OUTPUT, MANDATORY, default=no default value.]
Object **freqThrow** [INPUT, MANDATORY, default='LoThrow']
String **unit** [INPUT, OPTIONAL, default=no default value.]
Boolean **shift** [INPUT, OPTIONAL, default=False.]
Boolean **overwrite** [INPUT, OPTIONAL, default=True.]

API details

Properties

SpectrumContainer ds [INPUT, MANDATORY, default=no default value.]

Input container with the spectra to be folded. Examples of `SpectrumContainer` are `Spectrum1d`, `Spectrum2d`, `SpectralSimpleCube`.

SpectrumContainer result [OUTPUT, MANDATORY, default=no default value.]

Output container with the folded spectra.

Object freqThrow [INPUT, MANDATORY, default='LoThrow']

Either you can pass here directly the frequency throw to be used when folding the spectra. Or, alternatively, specify the name of the attribute (e.g. column name) where the frequency throw can be found. By default, 'LoThrow' (or 'loThrow') is used as value if an attribute with this name or a meta data item with with key is available. If a double value is provided or the throw taken from the meta data, the same double value is applied to all the spectra in the container. If the throw is specified as an attribute name (or in the default) it is typically dependent on the point spectrum.

String unit [INPUT, OPTIONAL, default=no default value.]

Specify the unit the frequency throw is expressed in. If no unit is specified it is assumed that the unit of the frequency throw is assumed to be the same as the one found on the wave scale. Typical values are "GHz" or "MHz".

Boolean shift [INPUT, OPTIONAL, default=False.]

If true, the spectrum is shifted in frequency scale by half the throw distance so that the resulting spectrum is centered between the original and the "switched" spectrum.

Boolean overwrite [INPUT, OPTIONAL, default=True.]

Specify whether the input data container can be reused - the values found therein are overwritten. If set to false, a new container is created and the spectral segments are reduced in shape by the frequency throw distance.

See also

- Developers Manual: herschel.ia.toolbox.spectrum.FoldSpectrumTask

History

- `2011-08-08 - melchior`: renamed from FoldSpectrum

1.165. FreeShapeModel

Full Name:	herschel.ia.numeric.toolbox.fit.sample.FreeShapeModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit.sample import FreeShapeModel
Category	Mathematics/Fitting

Description

Free Shape Model.

f(x:p) = p (expanded)

where p is a array of amplitudes of the same size as the input x divided by the number of pixels per bin (ppb). When ppb > 1, each p has a shape which is used to fill the pixels of the bin. Initially the shape is a top-hat, which can be autoconvolved. The input x needs to be an Int1d with indices pointing into the (expanded) p-array.

By default ppb = 5.

The parameters are initialized at {0}.

Although this is a LinearModel it will not work very well with the (linear) Fitter. It will be a very ill-posed problem. Its exponential prior ensures that all parameters are kept positive.

Using NestedSampler, its exponential prior will ensure that all parameters are kept positive.

Example

Example 1: FreeShapeModel

```
F = DataFormatter()
nn = 100
x = Int1d.range( nn )
fsm = FreeShapeModel( nn )
fsm.autoConvolve( 2 )
print F.p( fsm.getPixelsPerBin() ) + F.p( fsm.getShape(), 20 )
```

See also

- Fitter
- NestedSampler
- Developers Manual: herschel.ia.numeric.toolbox.fit.sample.FreeShapeModel

1.166. FullQuery

Full Name:	herschel.ia.pal.query.FullQuery
Type:	Java Class -
Import:	from herschel.ia.pal.query import FullQuery
Category	Data access

Description

A data mining query formulates a query the full interface of a Product.

As such, one can query any aspect of a Product. Typically this type of query is quite slow and should be used in combination with query refinements.

Example

Example 1: Example of a data mining query

```
q = FullQuery(SimpleCube, "p", "ANY(p['array'].data&lt;2)")
```

See also

- Developers Manual: herschel.ia.pal.query.FullQuery

1.167. GAMMALN

Full Name:	herschel.ia.numeric.toolbox.basic.GammaLn
Alias:	GAMMALN
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import GammaLn
Category	Mathematics/Statistics

Description

Returns the natural logarithm of the Gamma function.

For more information on the Gamma function, see for instance the MathWorld website.

Example

Example 1: Apply GAMMALN to a Double1d

```
x = Double1d([1.0,2.0,3.0,4.0,5.0,6.0])
gn = GAMMALN(x)
print gn
# [0.0,-4.440892098500626E-16,0.6931471805599443,
#  1.791759469228055,3.1780538303479453,4.787491742782044]
```

API Summary

Jython Syntax
<y>=GAMMALN(<x>)

Properties
Number or array **x** [INPUT, MANDATORY, default=no default value]
Number or array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Number or array x [INPUT, MANDATORY, default=no default value]

The value or values of which to compute the natural logarithm of the Gamma function.

Number or array y [OUTPUT, MANDATORY, default=no default value]

The natural logarithm value or values of the Gamma function.

See also

- GammaP
- GammaQ
- Developers Manual: herschel.ia.numeric.toolbox.basic.GammaLn

1.168. GammaP

Full Name:	herschel.ia.numeric.toolbox.basic.GammaP
Alias:	GammaP
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import GammaP
Category	Mathematics/Statistics

Description

Returns the incomplete Gamma function P(a,x).

For more information on the incomplete Gamma function, see for instance the MathWorld website.

Example

Example 1: Apply GammaP to a Double1d

```
a = 0.5
x = Double1d([0.0316228,0.0707107,5.0,1.04881,2.44949,25.4951])
gp = GammaP(a)(x)
print gp
# [0.19856221732013887,0.2931279825202761,0.9984345977771615,
#  0.8524713739638958,0.9731274396553844,0.9999999999990717]
```

API Summary

Jython Syntax

```
&lt;y&gt;=GammaP(a)(&lt;x&gt;)
```

Properties

Double **a** [INPUT, MANDATORY, default=no default value]

Number or array **x** [INPUT, MANDATORY, default=no default value]

Number or array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Double a [INPUT, MANDATORY, default=no default value]

The parametric exponent of the integrand.

Number or array x [INPUT, MANDATORY, default=no default value]

The value or values of which to compute the incomplete Gamma function.

Number or array y [OUTPUT, MANDATORY, default=no default value]

The incomplete Gamma function value or values.

See also

- GammaQ

- GAMMALN
- Developers Manual: herschel.ia.numeric.toolbox.basic.GammaP

1.169. GammaQ

Full Name:	herschel.ia.numeric.toolbox.basic.GammaQ
Alias:	GammaQ
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import GammaQ
Category	Mathematics/Statistics

Description

Returns the complement of the incomplete Gamma function Q(a,x).

Q(a,x) is defined as 1 - P(a,x), where P(a,x) is the incomplete Gamma function. For more information on the incomplete Gamma function, see for instance the MathWorld website.

Example

Example 1: Apply GammaQ to a Double1d

```
a = 0.5
x = Double1d([0.5,0.0316228,0.0707107,5.0,1.04881,2.44949,25.4951])
gq = GammaQ(a)(x)
print gq
# [0.31731050863888255,0.8014377826798611,0.7068720174797238,
#
 0.001565402222838555,0.14752862603610417,0.026872560344615565,9.282826956361127E-13]
```

API Summary

Jython Syntax

<y>=GammaQ(a)(<x>)

Properties

Double **a** [INPUT, MANDATORY, default=no default value]

Number or array **x** [INPUT, MANDATORY, default=no default value]

Number or array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Double a [INPUT, MANDATORY, default=no default value]

The parametric exponent of the integrand.

Number or array x [INPUT, MANDATORY, default=no default value]

The value or values of which to compute the complement of the incomplete Gamma function.

Number or array y [OUTPUT, MANDATORY, default=no default value]

The value or values of the complement of the incomplete Gamma function.

See also

- GammaP
- GAMMALN
- Developers Manual: herschel.ia.numeric.toolbox.basic.GammaQ

1.170. Gauss2DModel

Full Name:	herschel.ia.numeric.toolbox.fit.Gauss2DModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import Gauss2DModel
Category	Mathematics/Fitting

Description

Rotationally symmetric two dimensional Gaussian Model.

The model has 4 parameters yielding a "round" 2d-Gaussian.

$$f(x,y:p) = p_0 * \exp(-0.5 * ((x - p_1) / p_3)^2) * \exp(-0.5 * ((y - p_2) / p_3)^2)$$

p_0 = amplitide p_1 = x-shift p_2 = y-shift p_3 = sigma

The parameters are initialized at {1.0/Math.sqrt(Math.PI), 0.0, 0.0, 1.0}, normalised to 1.0. Parameter 3 (sigma) is always kept positive (>=0).

See example

For an asymmetric, rotated Gaussian2D model see Gauss2DRotModel

Example

Example 1: Gauss2DModel

```
gauss = Gauss2DModel()
print gauss
print gauss.getNumberOfParameters()
print gauss.getDimension()
# ... fitter etc. see LevenbergMarquardtFitter
```

Limitations

It is silently assumed that the Units on both x-axes are the same. Otherwise the reported Units on the parameters are not OK.

See also

- Gauss2DRotModel
- Developers Manual: herschel.ia.numeric.toolbox.fit.Gauss2DModel

1.171. Gauss2DRotModel

Full Name:	herschel.ia.numeric.toolbox.fit.Gauss2DRotModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import Gauss2DRotModel
Category	Mathematics/Fitting

Description

Rotated asymmetric 2 dimensional Gaussian Model.

The model has 6 parameters yielding a arbitrarily oriented elliptical 2d-Gaussian.

f(x,y:p) = p_0 * exp(-0.5 * ((aa * cc + bb * ss) * xx + (aa * ss + bb * cc) * yy + 2 * x * y * c * s * (bb - aa)))

where

- x = x - p_1
- y = y - p_2
- a = 1 / p_3 = 1 / σ_x
- b = 1 / p_4 = 1 / σ_y
- c = cos(p_5) = cosine of rotational angle
- s = sin(p_5) = sine of rotational angle
- aa = a * a. Etc.

The parameters are resp. amplitude, x-shift, y-shift, x-width, y-width and rotational angle (in radians). They are initialized at {1.0/PI, 0.0, 0.0, 1.0, 2.0, 0.0}. Parameters 3 & 4 (sigmas) are always kept positive (>=0).

Do **not** initialize both widths at the same value as the model then degenerates: the rotational angle does not have a direction anymore.

See example

For a circular symmetric Gaussian2D model see Gauss2DModel

Example

Example 1: Gauss2DRotModel

```
gauss = Gauss2DRotModel()
print gauss
print gauss.getNumberOfParameters()
# ... fitter etc. see LevenbergMarquardtFitter
```

Limitations

It is silently assumed that the Units on both x-axes are the same. Also, the units of the rotational angle are radians. Otherwise the reported Units on the parameters are not OK.

See also

- Gauss2DModel
- Developers Manual: herschel.ia.numeric.toolbox.fit.Gauss2DRotModel

1.172. GaussErrorDistribution

Full Name:	herschel.ia.numeric.toolbox.fit.sample.GaussErrorDistribution
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit.sample import GaussErrorDistribution
Category	Mathematics/Fitting

Description

GaussErrorDistribution models the fitting error using a Gauss distribution.

Gauss distr: $f(x) = (2 \pi s^2)^{-0.5} \exp(- 0.5 (x / s)^2)$

where s is the scale and x is the residual

For a simple way to choose the distribution in the NestedSampler use NestedSampler.setGaussDistribution().

It is the default.

See example

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.sample.GaussErrorDistribution

1.173. GaussianFilter

Full Name:	herschel.ia.numeric.toolbox.filter.GaussianFilter
Alias:	GaussianFilter
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.filter import GaussianFilter
Category	Mathematics/Signal processing

Description

Creates a Gaussian filter, that can be applied to numeric arrays of rank 1 and 2.

Example

Example 1: Apply GaussianFilter on a Int1d

```
x=Int1d([1,3,2,4,6,5])
f=GaussianFilter(3.0)
print f(x) # [1.539953486945335,1.9597782894453124,
           #  2.295455625723868,2.4710872510431194,
           #  2.4399273453142873,2.204343169997656]
```

API Summary

Jython Syntax

```
<f>=GaussianFilter(<sigma> [,<center>=true|false] [,<edge>=Convolution.ZEROES|CIRCULAR|REPEAT])
<x>=<f>(<x>)
```

See also

- BoxCarFilter
- Convolution
- Developers Manual: herschel.ia.numeric.toolbox.filter.GaussianFilter

History

- 2010-12-14 - AS: Add URM category; correct URM example.

1.174. gaussianSmoothing

Full Name:	herschel.ia.toolbox.image.GaussianSmoothingTask
Alias:	gaussianSmoothing
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import GaussianSmoothingTask
Category	Images/Analysis

Description

This is a task to smooth/filter an image using a convolution with a gaussian.

This is a task to smooth/filter an image using a convolution with a gaussian, for which the user must specify the standard deviation.

Example

Example 1: This is an example of how to use the gaussianSmoothing :

```
smoothed = gaussianSmoothing(image = myImage, sigma = 2.5)
```

API Summary

Properties
Image **image** [INPUT, MANDATORY, default=None]
Double **sigma** [INPUT, OPTIONAL, default=Double(3.0)]
Image **smoothed** [OUTPUT, MANDATORY, default=None]

API details

Properties

Image image [INPUT, MANDATORY, default=None]

This is the input image.

Double sigma [INPUT, OPTIONAL, default=Double(3.0)]

This is the standard deviation of the gaussian in pixels.

Image smoothed [OUTPUT, MANDATORY, default=None]

This is the resulting smoothed/filtered image.

See also

- Developers Manual: herschel.ia.toolbox.image.GaussianSmoothingTask

1.175. GaussModel

Full Name:	herschel.ia.numeric.toolbox.fit.GaussModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import GaussModel
Category	Mathematics/Fitting

Description

Gaussian Model.

$$f(x:p) = p_0 * \exp(-0.5 * ((x - p_1) / p_2)^2)$$

p_0 = amplitide p_1 = x-shift p_2 = sigma

The parameters are initialized at {1.0, 0.0, 1.0}. Parameter 2 (sigma) is always kept positive (>=0).

See example

Example

Example 1: GaussModel

```
gauss = GaussModel()
print gauss
print gauss.getNumberOfParameters()
print gauss( Double1d.range(11)-5 )        # gauss between [-5,5]
# ... fitter etc. see LevenbergMarquardtFitter
```

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.GaussModel

1.176. GEOMEAN

Full Name:	herschel.ia.numeric.toolbox.stat.GeoMean
Alias:	GEOMEAN
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.stat import GeoMean
Category	Mathematics/Statistics

Description

Yields the geometric mean value of the elements in the input array.

Example

Example 1: Apply GEOMEAN on a Float1d

```
x=Float1d([1,2,3])
print GEOMEAN(x) # 1.8171205928321397
```

API Summary

Jython Syntax
<y>=GEOMEAN(<x>)

Properties
any scalar array **x** [INPUT, MANDATORY, default=no default value]
double **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

any scalar array x [INPUT, MANDATORY, default=no default value]

The input array is integral or floating-point array; the former implicitly transformed to a double array.

double y [OUTPUT, MANDATORY, default=no default value]

Returns a double

See also

- MEAN
- MEDIAN
- STDDEV
- VARIANCE
- Developers Manual: herschel.ia.numeric.toolbox.stat.GeoMean

1.177. getObservation

Full Name:	herschel.ia.toolbox.util.GetObsTask
Alias:	getObservation
Type:	Java Task -
Import:	from herschel.ia.toolbox.util import GetObsTask
Category	Input-output

Description

Task to download, browse, or import an Observation.

Allows to efficiently open expanded observations.

Examples

Example 1: Open an Observation from a descriptor

```
obs = getObservation(path="/home/jadiaz/loadObs/from_hsa/12345678-
herschel.ia.obs.ObservationContext-100.xml");
```

Example 2: Open an Observation from the archive

```
obsid = 1342180475
obs = getObservation(obsid, useHsa=True)
```

Example 3: Open an Observation from the archive using a specific SPG Version

```
obsid = 1342180475
pool = MyHSAPool.getNewInstance(herschel.ia.pal.pool.hsa.MyHSAConnection.ON)
ps = ProductStorage([pool])
query = MetaQuery(herschel.ia.obs.ObservationContext, "p",
 "p.meta.containsKey('obsid') "+
"and p.meta['obsid'].value=="+str(obsid), True)
result = ps.select(query)
meta = result[0].meta
creator = meta['creator'].getValue()
obs = getObservation(obsid, useHsa=True, spgVersion=creator[5:])
spgVersionFromMeta = obs.meta['creator'].string
expectedSpgVersion = creator
assert expectedSpgVersion == spgVersionFromMeta, ("The SPG Versions should
 match: Expected "+
expectedSpgVersion +" but got "+spgVersionFromMeta)
```

API Summary

Jython Syntax

```
obs = getObservation(
obsid=&lt;obsid&gt; | path=&lt;path&gt; | urn=&lt;urn&gt; |
tag=&lt;tag&gt;
[, instrument=&lt;instrument&gt; ]
[, od=&lt;operationalDay&gt; ]
[, poolName=&lt;poolName&gt; ]
[, poolLocation=&lt;poolLocation&gt; ]
[, useHsa=True|False ]
```

Jython Syntax
[, save=True|False]
[, console=True|False]
[, allVersions=True|False]
[, version=<version>]
[, spgVersion=<spgVersion>]
[, verbose=True|False (ignored, allowed for compatibility reasons)]
[, useCache=True|False (ignored, allowed for compatibility reasons)]
[, creationDate=<date_time>]
[, latest=True|False]
)

Properties
Object **obsid** [INPUT, OPTIONAL, default=no default value]
String **path** [INPUT, OPTIONAL, default=no default value]
String **urn** [INPUT, OPTIONAL, default=no default value]
String **tag** [INPUT, OPTIONAL, default=no default value]
String **instrument** [INPUT, OPTIONAL, default=no default value]
Integer **od** [INPUT, OPTIONAL, default=no default value]
String **poolName** [INPUT, OPTIONAL, default=no default value]
String **poolLocation** [INPUT, OPTIONAL, default=no default value]
Boolean **useHsa** [INPUT, OPTIONAL, default=False]
Boolean **save** [INPUT, OPTIONAL, default=False]
Boolean **console** [INPUT, OPTIONAL, default=True]
Boolean **allVersions** [INPUT, OPTIONAL, default=False]
Integer **version** [INPUT, OPTIONAL, default=no default value]
String **spgVersion** [INPUT, OPTIONAL, default=no default value]
Boolean **verbose** [INPUT, OPTIONAL, default=no default value]
Boolean **useCache** [INPUT, OPTIONAL, default=no default value]
Object **creationDate** [INPUT, OPTIONAL, default=no default value]
Boolean **latest** [INPUT, OPTIONAL, default=True]

Miscellaneous

Parameters:

- Mandatory: **one or more** of the following (see incompatibilities):
  - `obsid` (IN, optional): The observation you want to open; Long or String
  - `path` (IN, optional): Observation directory or XML descriptor file (it must be located inside the directory that contains the products); String

    It is incompatible with:

    - `useHsa=True, poolLocation.path` allows `on_demand=True|False`

  - `urn` (IN, optional): The urn you want to open; String
  - `tag` (IN, optional): The tag you want to open; String
- `instrument` (IN, optional): The instrument used to create the observation. Possible values: `"HI-FI", "PACS", "SPIRE"`. For Parallel observations this must be specified.
- `od` (IN, optional): Operational Day; Integer
- `poolName` (IN optional): pool where the observation is located; String

  It is incompatible with:

  - `useHsa=True`

  If `poolLocation` is not specified, PoolManager is used to find `poolName`. If `poolLocation` is specified, a `poolName` directory is expected inside `poolLocation`. 'MyHSA' (case insensitive) can be used to refer to local MyHSA pool.
- `poolLocation` (IN optional): location of the pool where the observation is located; String

  It is incompatible with:

  - `useHsa=True`
  - `path`

  It is expected that `poolName` specifies the name of a directory inside `poolLocation`. This name is the LocalStore name.
- `useHsa` (IN optional): Use HSA for retrieving the observation; Boolean

  It is incompatible with:

  - `poolName`
  - `poolLocation`
  - `path`
- `console` (IN optional): specifies whether the error handler is the console one; Boolean

  True means that all the messages will be dumped to the console, so no GUI will be used to show errors or to ask for a user input. False means that a GUI will be used and the user can choose an option when an error occurs.
- `allVersions` (IN optional): shows all track versions. Default value: 'false'; Boolean

  If 'allVersions' is 'True', 'latest' parameter is ignored.

  If 'version' is specified, 'allVersions' is ignored.

  If 'spgVersion' is specified, 'allVersions' is ignored.
- `version` (IN optional): product track version (resets to zero when downloaded to a local pool, tracks only user changes from that moment on); Integer
- `spgVersion` (IN optional): SPG Version that created this product, format is like '11.1.0'; String
- `verbose` (IN optional): ignored; Boolean
- `useCache` (IN optional): ignored; Boolean

- `creationDate` (IN optional): product creation date; Object

  Can have the following formats:

  - {@link FineTime} object
  - long
  - {@link TimeScale#TAI} string object
  - {@link TimeScale#UTC} string object
  - "dd-MMM-yyy HH:mm:ss" string object
  - "dd-MM-yyy HH:mm:ss" string object

- `latest` (IN optional): specifies that the latest (highest URN number) version by pool must be retrieved without asking the user. Default value: 'true'; Boolean

  If 'version' is specified or 'allVersions' is 'true', 'latest' is parameter is ignored.

Rules on parameters:

1. If `path` (for an XML file) is provided, `useHsa,` and `poolLocation` cannot be used.
   - If `poolName` is given, `path` should be interpreted as the path to the pool if it is given.
   - If `poolName` is empty, `path` should be assumed to be the path to an XML file for loading an observation from an HSA download directory structure.

   Products are ingested into MyHsa or the required pool based on the product URN.

   URNs handling

   If the product URN is an HSA urn (pool identifier is 'hsa'), the product is imported into MyHsa. If the URN specifies a different pool (pool identifier is not 'hsa'), the import mechanism will use PoolManager to locate the suitable pool: `Urn urn = ... PoolManager.getPool(urn.getPoolId());` See URNs clashes.

2. `save` parameter is associated to HSA (an HSA product has been selected). It specifies whether the entire observation is saved into MyHSA or not.
   - `save=False` No products are saved locally.
   - `save=True` Import the entire observation in background.

3. If `path` is not provided:
   a. By default, all local pools (LocalStores + MyHSA) are used in order to search the required observation.
   b. if `useHsa=True`, `poolName` and `poolLocation` cannot be used. If the selected product is an HSA product, based on `save` parameter, the entire observation will be ingested (save=True) into MyHSA or products are loaded in memory only (save=False).
   c. If `poolName` or `poolLocation` are used, `useHsa=False` cannot be provided. `save` parameter cannot be used.

4. When searching for an observation, if more than one product is found, a user input is required.

5. When importing an observation URN clashes can appear: see URNs clashes below.

6. Rules on `obsid, od, instrument, urn, tag` parameters

a. `urn` : if an urn is specified, the rest of parameters are ignored.

b. `tag` : if a tag is specified and `obsid` is not present, `instrument` and `od` (operational day) are ignored. If `obsid` is present, `tag` is used as another filter.

c. `obsid, od` (operational day), `instrument`: a query is constructed with this parameters when they are available.

7. URNs clashes

   When importing an observation than contains non HSA URNs, URN clashes can be found. An URN clash means that the destination pool contains the same identifier (URN) than the URN being imported. This is an error and a user input is required.

8. If a parameter incompatibility is found, an exception is raised.

9. When specifying spgVersion, only the number is required e.g) '11.1.0'

API details

Properties

Object obsid [INPUT, OPTIONAL, default=no default value]

The observation you want to open (number or string).

String path [INPUT, OPTIONAL, default=no default value]

Observation directory or XML descriptor file containing the observation.

String urn [INPUT, OPTIONAL, default=no default value]

The urn you want to open.

String tag [INPUT, OPTIONAL, default=no default value]

The tag you want to open.

String instrument [INPUT, OPTIONAL, default=no default value]

The instrument used to create the observation. Possible values: `"HIFI", "PACS", "SPIRE"`. For parallel observations this must be specified.

Integer od [INPUT, OPTIONAL, default=no default value]

Operational Day.

String poolName [INPUT, OPTIONAL, default=no default value]

pool where the observation is located.

String poolLocation [INPUT, OPTIONAL, default=no default value]

location of the pool where the observation is located.

Boolean useHsa [INPUT, OPTIONAL, default=False]

Use HSA for retrieving the observation.

Boolean save [INPUT, OPTIONAL, default=False]

Specifies whether the entire observation is saved into MyHSA.

Boolean console [INPUT, OPTIONAL, default=True]

Specifies whether the error handler is the console one.

Boolean allVersions [INPUT, OPTIONAL, default=False]

Specifies whether the queries will search for all observation versions or not.

Integer version [INPUT, OPTIONAL, default=no default value]

Product version (track version). Resets to zero when downloaded to a local pool and tracks user changes from that moment on.

String spgVersion [INPUT, OPTIONAL, default=no default value]

SPG Version that created this product. Format is like '11.1.0'.

Boolean verbose [INPUT, OPTIONAL, default=no default value]

Ignored. Allowed for compatibility reasons.

Boolean useCache [INPUT, OPTIONAL, default=no default value]

Ignored. Allowed for compatibility reasons.

Object creationDate [INPUT, OPTIONAL, default=no default value]

Product creation date. Can have the following formats:

- herschel.share.fltdyn.time.FineTime object
- long
- herschel.share.fltdyn.time.TimeScale.TAI string object
- herschel.share.fltdyn.time.TimeScale.UTC string object
- "dd-MMM-yyy HH:mm:ss" string object
- "dd-MM-yyy HH:mm:ss" string object

Boolean latest [INPUT, OPTIONAL, default=True]

Specifies that the latest (highest URN number) version by pool must be retrieved without asking the user. If 'version' is provided, 'latest' value is ignored.

See also

- Developers Manual: herschel.ia.toolbox.util.GetObsTask

1.178. HAMMING

Full Name:	herschel.ia.numeric.toolbox.xform.Hamming
Alias:	HAMMING
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.xform import Hamming
Category	Mathematics/Signal processing

Description

Hamming function for multiplying with input arrays.

This function performs an element-by-element multiplication of a input array with a Hamming function having the same number of elements. The Hamming function is defined by the following equation:

```
Hamming[i] = 0.54 + 0.46 * cos(2π( i-ξ ) / N)
where:
Hamming[i] is the value of the Hamming function at array index i, where 0≤i≤(N-1)
N is the length of the array for which the Hamming function is calculated.
ξ is the phase shift, defined as ξ = N/2.0
```

Input:

The input array should be a `Double1d` or `Float1d` and have length N≥3.

Output:

The output array is a floating point array of length N that contains the element-by-element multiplication of the input array with the Hamming function.

Symmetry:

This equation produces a Hamming function with a unique point at index `i=0`. For even length arrays, the maximum amplitude is located at `N/2.0`. For odd length arrays, there are two amplitude maxima located at `ceiling(N/2.0)` and `floor(N/2.0)`.

Syntax:

To apply an element-by-element multiplication of the array `x` by the Hamming function, use any of the following options while coding in Jython:

1. `p = HAMMING(x) #Area normalized`
2. `p = HAMMING.AREA(x) #Area normalized, alternative syntax`
3. `q = HAMMING.AMPLITUDE(x) #Amplitude normalized`
4. `r = HAMMING.ENERGY(x) #Energy normalized`

Example

Example 1: Apply Hamming function to an input array x.

```
x = Double1d(100,1.0)
p = HAMMING(x) #Area normalized
p2 = HAMMING.AREA(x) #Area normalized, alternative syntax
```

Example 1: Apply Hamming function to an input array x.

```
q = HAMMING.AMPLITUDE(x) #Amplitude normalized
r = HAMMING.ENERGY(x) #Energy normalized
```

API Summary

Jython Syntax

```
<y>=HAMMING.[<normalization> = AREA|AMPLITUDE|ENERGY] (<x>)
```

Properties

Double1d or Float1d **x** [INPUT, MANDATORY, default=No default value]

Double1d or Float1d **y** [OUTPUT, OPTIONAL, default=No default value]

AMPLITUDE|ENERGY|AREA|PROCEDURE **normalization** [INPUT, OPTIONAL, default=No default value]

Limitations

This function only operates on `Double1d` and `Float1d` arrays. It does not work for `Complex1d` arrays.

API details

Properties

Double1d or Float1d x [INPUT, MANDATORY, default=No default value]

The input array to be multiplied by the Hamming function.

Double1d or Float1d y [OUTPUT, OPTIONAL, default=No default value]

Outputs an array containing the element-by-element multiplication of the input array with the Hamming function.

AMPLITUDE|ENERGY|AREA|PROCEDURE normalization [INPUT, OPTIONAL, default=No default value]

The type of normalization of the Hamming function.

See also

- HAMMING
- Developers Manual: herschel.ia.numeric.toolbox.xform.Hamming

History

- `2007-11-23 - ZW`: First version.

1.179. HANNING

Full Name:	herschel.ia.numeric.toolbox.xform.Hanning
Alias:	HANNING
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.xform import Hanning
Category	Mathematics/Signal processing

Description

Hanning function for multiplying with input arrays.

This function performs an element-by-element multiplication of a input array with a Hanning function having the same number of elements. The Hanning function is defined by the following equation:

```
Hanning[i] = 0.50 + 0.50 * cos(2π( i-ξ ) / N)
where:
Hanning[i] is the value of the Hanning function at array index i, where 0≤i≤(N-1)
N is the length of the array for which the Hanning function is calculated.
ξ is the phase shift, defined as ξ = N/2.0
```

Input:

The input array should be a `Double1d` or `Float1d` and have length N≥3.

Output:

The output array is a floating point array of length N that contains the element-by-element multiplication of the input array with the Hanning function.

Symmetry:

This equation produces a Hanning function with a unique point at index `i=0`. For even length arrays, the maximum amplitude is located at `N/2.0`. For odd length arrays, there are two amplitude maxima located at `ceiling(N/2.0)` and `floor(N/2.0)`.

Syntax:

To apply an element-by-element multiplication of the array `x` by the Hanning function, use any of the following options while coding in Jython:

1. `p = HANNING(x) #Area normalized`
2. `p = HANNING.AREA(x) #Area normalized, alternative syntax`
3. `q = HANNING.AMPLITUDE(x) #Amplitude normalized`
4. `r = HANNING.ENERGY(x) #Energy normalized`

Example

Example 1: Apply Hanning function to an input array.

```
x = Double1d(100,1.0)
p = HANNING(x) #Area normalized
p2 = HANNING.AREA(x) #Area normalized, alternative syntax
```

Example 1: Apply Hanning function to an input array.

```
q = HANNING.AMPLITUDE(x) #Amplitude normalized
r = HANNING.ENERGY(x) #Energy normalized
```

API Summary

Jython Syntax

```
<y>=HANNING.[<normalization> = AREA|AMPLITUDE|ENERGY] (<x>)
```

Properties

Double1d or Float1d **x** [INPUT, MANDATORY, default=No default value]

Double1d or Float1d **y** [OUTPUT, OPTIONAL, default=No default value]

AMPLITUDE|ENERGY|AREA|PROCEDURE **normalization** [INPUT, OPTIONAL, default=No default value]

Limitations

This function only operates on `Double1d` and `Float1d` arrays. It does not work for `Complex1d` arrays.

API details

Properties

Double1d or Float1d x [INPUT, MANDATORY, default=No default value]

The input array to be multiplied by the Hanning function.

Double1d or Float1d y [OUTPUT, OPTIONAL, default=No default value]

Outputs an array containing the element-by-element multiplication of the input array with the Hanning function.

AMPLITUDE|ENERGY|AREA|PROCEDURE normalization [INPUT, OPTIONAL, default=No default value]

The type of normalization of the Hanning function.

See also

- HANNING
- Developers Manual: herschel.ia.numeric.toolbox.xform.Hanning

History

- `2007-11-23 - ZW`: First version.

1.180. HarmonicDynamicModel

Full Name:	herschel.ia.numeric.toolbox.fit.sample.HarmonicDynamicModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit.sample import HarmonicDynamicModel
Category	Mathematics/Fitting

Description

Harmonic oscillator Model of adaptable Order.

$$f(x:p) = \sum_j (p_k \cos(2 \pi j x) + p_{k+1} \sin(2 \pi j x))$$

$$j = 1, N; k = 0, 2N, 2$$

All parameters are initialized at 1.0. It is a linear model.

See example

Example

Example 1: HarmonicDynamicModel

```
harm = HarmonicDynamicModel( 3 )     # period = 1
print harm.getNumberOfParameters()        # 6
harm = HarmonicModel( 4, 2.7 )        # period = 2.7
```

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.sample.HarmonicDynamicModel

1.181. HarmonicModel

Full Name:	herschel.ia.numeric.toolbox.fit.HarmonicModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import HarmonicModel
Category	Mathematics/Fitting

Description

Harmonic oscillator Model.

f(x:p) = SUM_j (p_k * cos(2*pi*j*x) + p_k+1 * sin(2*pi*j*x)) j = 1, N; k = 0, 2N-2, 2

The number of parameters is 2 * order. All parameters are initialized at 1.0. It is a linear model.

See example

Example

Example 1: HarmonicModel

```
harm = HarmonicModel( 3 )            # period = 1
print harm.getNumberOfParameters()        # 6
harm = HarmonicModel( 4, 2.7 )        # period = 2.7
```

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.HarmonicModel

1.182. HduHeaders

Full Name:	herschel.ia.io.fits.HduHeaders
Alias:	HduHeaders
Type:	Java Class -
Import:	from herschel.ia.io.fits import HduHeaders
Category	Input-output

Description

A class for working with FITS HDUs. HDU data is loaded only when requested.

Restrictions:

- Only HCSS FITS files (version 4) can be modified and saved.
- Avoid working with several datasets at the same time if you are planning to modify them.
- Only FITS files can be modified (fits from an InputStream cannot be updated).
- If the HDU is a compositeDataset, the children are returned when asking for the HDU dataset.
- If you modify a CompositeDataset, only its Metadata can be updated. If you want to add or remove a CompositeDataset child, you must work with it directly (add/remove CompositeDataset children directly).
- If you remove a CompositeDataset, its children are removed also.
- When modifying/updating/removing HDUs, a temporary file is created.

Examples

Example 1: default usage:

```
fa = FitsArchive();
product = Product()
ds = ArrayDataset(Int1d([1,2,3,4]))
product.sets['ds1']=ds
fa.save(path,product)
#fa.reader = FitsArchive.STANDARD_READER #for reading non HCSS FITS files.
fh = fa.getFitsHduHeaders(path)
#get last dataset
ds = fh.headers[fh.numHeaders-1].dataset
print ds
#or
h = fh.headers[fh.numHeaders-1]
ds = h.dataset
print ds
```

Example 2: modify a metadata/fits header:

```
fa = FitsArchive();
product = Product()
ds = ArrayDataset(Int1d([1,2,3,4]))
product.sets['ds1']=ds
fa.save(path,product)
fh = fa.getFitsHduHeaders(path)
#get last dataset
h = fh.headers[fh.numHeaders-1]
ds = h.dataset
```

Example 2: modify a metadata/fits header:

```
print ds
ds.description = 'new description'
h.updateDataset(ds)
#or
fh.updateDataset(fh.numHeaders-1, ds)
```

Example 3: add a new dataset/HDU:

```
fa = FitsArchive();
product = Product()
ds = ArrayDataset(Int1d([1,2,3,4]))
product.sets['ds1']=ds
fa.save(path,product)
fh = fa.getFitsHduHeaders(path)
ds = ArrayDataset(Int1d([1,2,3]))
fh.addDataset('new_dataset_id',ds)
```

Example 4: remove a HDU:

```
fa = FitsArchive();
product = Product()
ds = ArrayDataset(Int1d([1,2,3,4]))
product.sets['ds1']=ds
fa.save(path,product)
fh = fa.getFitsHduHeaders(path)
#remove last header
fh.removeHeader(fh.numHeaders-1)
```

See also

- Developers Manual: herschel.ia.io.fits.HduHeaders

1.183. help

Full Name:	herschel.ia.toolbox.util.HelpTask
Alias:	help
Type:	Java Task -
Import:	from herschel.ia.toolbox.util import HelpTask
Category	Session utilities

Description

Display help page associated with an object.

Users can access the help system by calling help from the command line. Help opens a navigator window (if help is not already opened) displaying the specific help topic for the specified item or the main entry of documentation if no entry is specified or specific help cannot found.

The current implementation displays the specific help only for the URMs (User Reference Manuals), Plot and Image.

Examples

Example 1: Main page of documentation

```
help()
```

Example 2: help on a task (help)

```
help(help)
```

Example 3: help on plot (1)

```
p = PlotXY
help(p)
```

Example 4: help on plot (2)

```
help("plotxy")
```

Example 5: help on display (1)

```
d = Display()
help(d)
```

Example 6: help on display (2)

```
help("display")
```

Example 7: Python help on modules

```
help('modules', askPython=True)
```

API Summary

Jython Syntax
`help([item, askPython=False])`

Properties
`Object item [INPUT, OPTIONAL, default=null]`
`Boolean askPython [INPUT, OPTIONAL, default=False]`

Limitations

- The current implementation displays the specific help only for classes in the URMs, Plot and Image.
- There is very little support if you pass a string (plot, display), pass objects instead.
- While you can ask for Python's help, interactive help is not supported (help(askPython=True), without an item).

Miscellaneous

No miscellaneous

API details

Properties

`Object item [INPUT, OPTIONAL, default=null]`

The item to look for in the help. Currently supported: explicit documentation ids (Strings), classes in the URM, Plot and Image (see examples). If no value is passed, then the main page of the documentation is shown. If askJython is True then item is mandatory.

`Boolean askPython [INPUT, OPTIONAL, default=False]`

If true it will ask Python for help instead of HCSS.

See also

- Developers Manual: herschel.ia.toolbox.util.HelpTask

History

- `2004-07-13 - NdC`: first release.
- `2013-09-23 - JDS`: enable access to Python's help (askPython)

1.184. Histogram

Full Name:	herschel.ia.numeric.toolbox.basic.Histogram
Alias:	Histogram
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Histogram
Category	Mathematics/Histograms

Description

Returns histogram column sizes for an array, using a user-specified bin size.

There are two steps involved in using this tool. First you create an object of type `Histogram` by specifying the bin size of the histogram you want to create. Then you pass your data array to this object, getting the array of histogram column sizes as output. You can use this array to create a plot of your histogram, together with the **BinCentres** function (see the link in the *See also* section). Look at the examples and the *Jython Syntax* section below to learn more.

Examples

Example 1: Apply Histogram to a Double1d

```
from herschel.ia.numeric.toolbox.basic import Histogram
hist=Histogram (3)
x=Double1d( [1,2,3,4] )
print hist(x)  # [3,1]
```

Example 2: Plot histogram over bin centres

```
from herschel.ia.numeric.toolbox.basic import Histogram
from herschel.ia.numeric.toolbox.basic import BinCentres
d = Double1d(200000,10)
d[99000:110000] = 15
d[99900:101000] = 20.0
rnd = RandomGauss()
for ix in range(200000):
   d[ix] += rnd.calc(1.0)
p1 = PlotXY (d)
binsize = STDDEV (d) * 2.35
print binsize
hist = Histogram(binsize)
bins = BinCentres(binsize)
p2 = PlotXY(bins(d),hist(d))
# Or you could try:  p = PlotXY (Histogram(binsize), BinCentres (binSize))
p2.style.chartType = Style.HISTOGRAM
```

API Summary

Jython Syntax

```
histogram=Histogram(&lt;binsize&gt;)
&lt;h&gt;=histogram(&lt;x&gt;)
```

Properties

```
Number binsize [INPUT, MANDATORY, default=no default value]
Array x [INPUT, MANDATORY, default=no default value]
```

Properties
Double1d h [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Number binsize [INPUT, MANDATORY, default=no default value]
The size of the histogram bins.

Array x [INPUT, MANDATORY, default=no default value]
The data of which to compute the histogram column sizes.

Double1d h [OUTPUT, MANDATORY, default=no default value]
The histogram column sizes.

See also

- BinCentres
- Developers Manual: herschel.ia.numeric.toolbox.basic.Histogram

1.185. historyExtract

Full Name:	herschel.ia.toolbox.history.HistoryExtractTask
Alias:	historyExtract
Type:	Java Task -
Import:	from herschel.ia.toolbox.history import HistoryExtractTask
Category	History

Description

This task extracts a specific parameter value from a task in the history of a product.

You have to provide the task name and the parameter from which you want to get the value. If the task has been performed more than once then the value from the latest result will be returned by default. In such cases you can get the value from one of the other performed tasks by providing the optional parameter 'taskNo'.

A TaskException will be thrown in case the task has not been performed on the product, or when the parameter does not exist within the task.

API Summary

Jython Syntax
`value = historyExtract(product,'taskName','parName',taskNo=-1)`

Properties
`Product product [INPUT, MANDATORY, default=no default value]`
`String task [INPUT, MANDATORY, default=no default value]`
`String parameter [INPUT, MANDATORY, default=no default value]`
`String value [OUTPUT, MANDATORY, default=no default value]`

API details

Properties

`Product product [INPUT, MANDATORY, default=no default value]`

The product from which the parameter will be extracted.

`String task [INPUT, MANDATORY, default=no default value]`

Task which contains the parameter

`String parameter [INPUT, MANDATORY, default=no default value]`

Parameter to extract

`String value [OUTPUT, MANDATORY, default=no default value]`

The value of the extracted parameter

See also

- Developers Manual: herschel.ia.toolbox.history.HistoryExtractTask

1.186. HttpClientFactory

Full Name:	herschel.ia.pal.pool.http.HttpClientFactory
Type:	Java Class -
Import:	from herschel.ia.pal.pool.http import HttpClientFactory
Category	Data access

Description

Factory class to create HttpClientPool instances.

Example

Example 1: Create a HttpClientPool

```
params = java.util.HashMap()
params.put("id", "foo")
params.put("url", "http://localhost:8080/hcss/pal")
storage=ProductStorage();
from herschel.ia.pal.pool.http import HttpClientFactory
storage.register(HttpClientFactory().createPool("http", params))
```

See also

- Developers Manual: herschel.ia.pal.pool.http.HttpClientFactory

1.187. imageAbs

Full Name:	herschel.ia.toolbox.image.ImageAbsTask
Alias:	imageAbs
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import ImageAbsTask
Category	Images/Analysis

Description

This is a task that takes the absolute value of the intensity values of an image.

This is a task that takes the absolute value of the intensity values of an image.

Example

Example 1: This is an example of how to use the imageAbs :

```
absolute = imageAbs(image = myImage)
```

API Summary

Properties
Image **image** [INPUT, MANDATORY, default=None]
Image **absolute** [OUTPUT, MANDATORY, default=None]

API details

Properties

Image image [INPUT, MANDATORY, default=None]

This is the input image.

Image absolute [OUTPUT, MANDATORY, default=None]

The output image, being the image with the absolute values of the input image.

See also

- Developers Manual: herschel.ia.toolbox.image.ImageAbsTask

1.188. imageAdd

Full Name:	herschel.ia.toolbox.image.ImageAddTask
Alias:	imageAdd
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import ImageAddTask
Category	Images/Analysis

Description

This task is able to add two images or a scalar to an image, using pixel positions or Wcs values.

You can add two images pixel-by-pixel (ref = 0) or based on their Wcs (ref = 1), or to add a scalar to all intensity values in an image. If two images are to be added based on their Wcs, they are regridded onto the spatial grid of the smallest image. If two images are added, the unit of the resulting sum will only be set only if both input images have the same unit. Their unit will then be used as the unit of the resulting sum. If a scalar was used for the addition, the unit of the resulting sum will be the unit of the input image.

Examples

Example 1: This is an example of how to add two images, pixel-by-pixel :

```
sum = imageAdd(image1 = myImage1, image2 = myImage2, ref = 0)
```

Example 2: This is an example of how to add two images, Wcs-based :

```
sum = imageAdd(image1 = myImage1, image2 = myImage2, ref = 1)
```

Example 3: This is an example of how to add a scalar to an image :

```
sum = imageAdd(image1 = myImage, scalar = 5.0)
```

API Summary

Properties
Image **image1** [INPUT, MANDATORY, default=None]
Image **image2** [INPUT, OPTIONAL, default=None]
Integer **ref** [INPUT, OPTIONAL, default=None]
Number **scalar** [INPUT, OPTIONAL, default=None]
Image **sum** [OUTPUT, OPTIONAL, default=None]

API details

Properties

Image image1 [INPUT, MANDATORY, default=None]

This is the first image term/addend.

Image image2 [INPUT, OPTIONAL, default=None]

This is the second image term/addend.

Integer ref [INPUT, OPTIONAL, default=None]

This is the reference frame for the calculation of the sum. Possible values are 0 for pixel-by-pixel based addition, and 1 for Wcs based addition.

Number scalar [INPUT, OPTIONAL, default=None]

This is the scalar term/addend.

Image sum [OUTPUT, OPTIONAL, default=None]

This is the resulting sum.

See also

- Developers Manual: herschel.ia.toolbox.image.ImageAddTask

1.189. ImageAxis

Full Name:	herschel.ia.gui.image.ImageAxis
Type:	Java Class -
Import:	from herschel.ia.gui.image import ImageAxis
Category	Images/Display

Description

Axes for image display.

This class can change the look of the axes of an image display. A grid can also be enabled.

API Summary

Methods
disable Disables the axis.
enable Enables the axis.
setLabel (String newLabel) Set the label of the axis.
String **getLabel** Returns the label.
Position **getOrientation** Returns the orientation of the axis.
setOrientation (Position orientation) Sets the orientation of the axis.
setAxisStroke (int stroke) Sets the axis stroke.
int **getAxisStroke** Returns the stroke (line width) of the axis.
setTicksColor (Color color) Sets the color of the ticks of the axis.
Color **getTicksColor** Returns the color of the ticks of the axis.
setAxisColor (Color color) Sets the color of the axis.
Color **getAxisColor** Returns the color of the axis.
setLabelFont (Font labelFont) Sets the font for the label.
setLabelFont (int newSize) Sets the size of the font for the label.
Font **getLabelFont**

Methods
Returns the font for the label.
Color **getLabelFontColor** Returns the color of the label.
setLabelFontColor (Color labelFontcolor) Sets the color for the label.
showColorTable (boolean showColorTable) Shows (true) or hides (false) the color table for this axis.
getColorTable Returns whether or not to show the color table for this axis.
setWorldCoordinates (boolean wcs) Shows the world coordinates (True) or the pixel coordinates (False).
getWorldCoordinates Returns whether of not to show the world Coordinates.
setTickLabelFont (int newSize) Sets the size of the font for the ticks.
setTickLabelFont (Font tickLabelFont) Sets the font of the ticks.
Font **getTickLabelFont** Returns the font of the ticks.
Color **getTickLabelFontColor** Returns the color of the font of the ticks.
setTickLabelFontColor (Color color) Sets the color of the ticks font.
setInnerTickLength (int length) Sets the length of the inner ticks.
int **getInnerTickLength** Returns the inner tick length.
setOuterTickLength (int length) Sets the length of the outer ticks.
int **getOuterTickLength** Returns the outer tick length.
setMainTicks (int ticks) Sets the number of main ticks.
int **getMainTicks** Returns the number of main (labeled) ticks.
setMinorTicks (int ticks) Sets the number of minor ticks.
int **getMinorTicks** Returns the number of minor (non-labeled) ticks, between two main ticks.
setTickDistance (int pixels) Sets the tick distance.

Methods
int **getTickDistance** Returns the number of pixels between two ticks on the axis.
showGridLines (boolean gl) Enables or disables the grid.
boolean **getShowGridLines** Returns the status of the grid.
setGridColor (Color gridColor) Sets the color of the grid for this axis.
Color **getGridColor** Returns the color of the grid.
setDecimalDegrees (boolean b) Shows the world coordinates in decimal degrees (True) or not.
getDecimalDegrees Returns whether of not to show the world Coordinates in decimal degrees.
getAutomaticTicks Returns the values of the automatically calculated ticks.

API Details

Methods

disable
Disables the axis.

enable
Enables the axis.

setLabel (String newLabel)
Set the label of the axis. The label of the axis is set. If no label is needed, an empty string should be given as label **Argument** String **newLabel** [INPUT, MANDATORY, default=no default value] A String value that will be the new Label of the axis.

String **getLabel**
Returns the label. **Return** **String** The label of the axis.

Position **getOrientation**
Returns the orientation of the axis. **Return**

***Position* getOrientation**

Position

The orientation (as a Position)

setOrientation (Position orientation)

Sets the orientation of the axis.

Argument

`Position` **`orientation`** `[INPUT, MANDATORY, default=no default value]`

The orientation of the Axis (AxisConstants.TOP, AxisConstants.BOTTOM, AxisConstants.LEFT, AxisConstants.RIGHT)

setAxisStroke (int stroke)

Sets the axis stroke.

Sets the stroke (line width) of the axis (in pixels)

Argument

`int` **`stroke`** `[INPUT, MANDATORY, default=no default value]`

The linewidth in pixels.

***int* getAxisStroke**

Returns the stroke (line width) of the axis.

Return

int

The stroke of the axis.

setTicksColor (Color color)

Sets the color of the ticks of the axis.

Argument

`Color` **`color`** `[INPUT, MANDATORY, default=no default value]`

The `Color` of the ticks of this axis.

***Color* getTicksColor**

Returns the color of the ticks of the axis.

Return

Color

Color The color of the ticks of the axis.

setAxisColor (Color color)

Sets the color of the axis.

Argument

`Color` **`color`** `[INPUT, MANDATORY, default=no default value]`

The `Color` of this axis.

***Color* getAxisColor**

Returns the color of the axis.

Color getAxisColor

Return

Color

Color The color of the axis.

setLabelFont (Font labelFont)

Sets the font for the label.

Argument

`Font` **labelFont** `[INPUT, MANDATORY, default=no default value]`

The new `Font`.

setLabelFont (int newSize)

Sets the size of the font for the label.

Argument

`int` **newSize** `[INPUT, MANDATORY, default=no default value]`

The new size of the label font.

Font getLabelFont

Returns the font for the label.

Return

Font

The font for the label.

Color getLabelFontColor

Returns the color of the label.

Return

Color

the color of the label.

setLabelFontColor (Color labelFontcolor)

Sets the color for the label.

Argument

`Color` **labelFontcolor** `[INPUT, MANDATORY, default=no default value]`

The `Color` of the axis labels.

showColorTable (boolean showColorTable)

Shows (true) or hides (false) the color table for this axis.

Argument

`boolean` **showColorTable** `[INPUT, MANDATORY, default=no default value]`

Show(true) or hide(false) the colortable.

getColorTable

Returns whether or not to show the color table for this axis.

setWorldCoordinates (boolean wcs)

Shows the world coordinates (True) or the pixel coordinates (False).

Argument

`boolean` **`wcs`** `[INPUT, MANDATORY, default=no default value]`

True if the world coordinates should be shown.

getWorldCoordinates

Returns whether of not to show the world Coordinates.

setTickLabelFont (int newSize)

Sets the size of the font for the ticks.

Argument

`int` **`newSize`** `[INPUT, MANDATORY, default=no default value]`

The new size of the ticklabel font.

setTickLabelFont (Font tickLabelFont)

Sets the font of the ticks.

Argument

`Font` **`tickLabelFont`** `[INPUT, MANDATORY, default=no default value]`

A `Font` value.

Font **getTickLabelFont**

Returns the font of the ticks.

Return

Font

The font of the ticks.

Color **getTickLabelFontColor**

Returns the color of the font of the ticks.

Return

Color

The color of the ticks font.

setTickLabelFontColor (Color color)

Sets the color of the ticks font.

Argument

`Color` **`color`** `[INPUT, MANDATORY, default=no default value]`

A `Color` value.

setInnerTickLength (int length)

Sets the length of the inner ticks.

Sets the distance that the ticks enter in the image.

Argument

`int` **`length`** `[INPUT, MANDATORY, default=no default value]`

setInnerTickLength (int length)

An `int` value.

***int* getInnerTickLength**

Returns the inner tick length.

Returns the distance that the ticks enter in the image.

Return

int

The distance that the ticks enter in the image.

setOuterTickLength (int length)

Sets the length of the outer ticks.

Sets the distance that the ticks go out of the image.

Argument

`int` **`length`** `[INPUT, MANDATORY, default=no default value]`

An `int` value.

***int* getOuterTickLength**

Returns the outer tick length.

Returns the distance that the ticks go out of the image.

Return

int

The distance that the ticks go out of the image.

setMainTicks (int ticks)

Sets the number of main ticks.

Main ticks are labeled and are slightly longer than the other (minor) ticks. If the number of ticks is less than two, automatic selection of the number of ticks is enabled.

Argument

`int` **`ticks`** `[INPUT, MANDATORY, default=no default value]`

The number of main ticks to use on the axis.

***int* getMainTicks**

Returns the number of main (labeled) ticks.

Return

int

The number of main ticks.

setMinorTicks (int ticks)

Sets the number of minor ticks.

Sets the number of minor ticks between 2 main ticks. Minor ticks are not labeled. Negative values will take the standard value of 4 minor ticks.

Argument

setMinorTicks (int ticks)

`int` **`ticks`** `[INPUT, MANDATORY, default=no default value]`

The number of minor ticks between 2 main ticks.

int **getMinorTicks**

Returns the number of minor (non-labeled) ticks, between two main ticks.

Return

int

The number of minor ticks between two main ticks.

setTickDistance (int pixels)

Sets the tick distance.

Sets the distance (in pixels) between two (minor) ticks. Negative (or 0) values will enable automatic selection of the distance.

Argument

`int` **`pixels`** `[INPUT, MANDATORY, default=no default value]`

The number of pixels between two ticks.

int **getTickDistance**

Returns the number of pixels between two ticks on the axis.

A negative value means the automatic selection of the tick distance is enabled.

Return

int

int The distance (in pixels) between two ticks.

showGridLines (boolean gl)

Enables or disables the grid.

Shows or hides the grid for this axis. The grid can only be displayed if the magnification is at least 4!

Argument

`boolean` **`gl`** `[INPUT, MANDATORY, default=no default value]`

A `boolean` value.

boolean **getShowGridLines**

Returns the status of the grid.

Return

boolean

True if the grid is shown

setGridColor (Color gridColor)

Sets the color of the grid for this axis.

Argument

`Color` **`gridColor`** `[INPUT, MANDATORY, default=no default value]`

The color for the grid.

`Color` `getGridColor`

Returns the color of the grid.

Return

Color

The color of the grid.

`setDecimalDegrees (boolean b)`

Shows the world coordinates in decimal degrees (True) or not.

Argument

`boolean` **`b`** `[INPUT, MANDATORY, default=no default value]`

True if the world coordinates should be shown in decimal degrees.

`getDecimalDegrees`

Returns whether of not to show the world Coordinates in decimal degrees.

`getAutomaticTicks`

Returns the values of the automatically calculated ticks.

See also

- Developers Manual: herschel.ia.gui.image.ImageAxis

1.190. imageCeil

Full Name:	herschel.ia.toolbox.image.ImageCeilTask
Alias:	imageCeil
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import ImageCeilTask
Category	Images/Analysis

Description

This is a task that ceils the intensity values of an image.

This is a task that ceils the intensity values of an image.

Example

Example 1: This is an example of how to use the imageCeil :

```
ceiled = imageCeil(image = myImage)
```

API Summary

Properties
`Image image [INPUT, MANDATORY, default=None]`
`Image ceiled [OUTPUT, MANDATORY, default=None]`

API details

Properties

`Image image [INPUT, MANDATORY, default=None]`

This is the input image.

`Image ceiled [OUTPUT, MANDATORY, default=None]`

This is the output ceiled image.

See also

- Developers Manual: herschel.ia.toolbox.image.ImageCeilTask

1.191. ImageContourExplorer

Full Name:	herschel.ia.gui.image.ImageContourExplorer
Type:	Java Class -
Import:	from herschel.ia.gui.image import ImageContourExplorer

Description

An explorer for ImageContours.

API Summary

Constructor
ImageContourExplorer The construction of a new ImageContourExplorer.

Methods
String **getDescription** Returns the description for this ImageContourExplorer .
ImageContour **getObject** Returns the object for this ImageContourExplorer.
Class **getVariableType** Returns the expected variable type for this ImageContourExplorer.

API Details

Constructor

ImageContourExplorer
The construction of a new ImageContourExplorer.

Methods

String **getDescription**
Returns the description for this ImageContourExplorer . **Return** **String** The description for this ImageContourExplorer.

ImageContour **getObject**
Returns the object for this ImageContourExplorer. **Return** **ImageContour** The object for this ImageContourExplorer.

Class **getVariableType**
Returns the expected variable type for this ImageContourExplorer. **Return** **Class** The expected variable type for this ImageContourExplorer.

See also

- Developers Manual: herschel.ia.gui.image.ImageContourExplorer

1.192. ImageContour

Full Name:	herschel.ia.dataset.image.ImageContour
Type:	Java Class -
Import:	from herschel.ia.dataset.image import ImageContour

Description

A class to deal with ImageContours.

API Summary

Constructors
ImageContour This constructor creates a new ImageContour.
ImageContour (Image image) This constructor creates a new ImageContour, associated with the given image.
ImageContour (Image image, int nbOfContourValues) This constructor creates a new ImageContour, associated with the given image
ImageContour (Image image, int nbOfContourLevels, double[] extremeContourValues, String distribution) This constructor creates a new ImageContour.

Methods
addContourLevel (ContourLevel contourLevel, double contourValue) Adds the given ContourLevel for the given contour value to this ImageContour.
addContourLevel (ContourLevel contourLevel, String key) Adds the given ContourLevel to this ImageContour with the given key.
setWcs (Wcs wcs) Sets the Wcs for this ImageContour.
Wcs **getWcs** Returns the Wcs for this ImageContour.
boolean **hasWcs** Checks whether a Wcs is attached to this ImageContour.
boolean **hasValidWcs** Checks whether a valid Wcs is attached to this ImageContour.

API Details

Constructors

ImageContour
This constructor creates a new ImageContour.

ImageContour (Image image)

This constructor creates a new ImageContour, associated with the given image.

Argument

`Image` **`image`** `[INPUT, MANDATORY, default=no default value]`

The image with which the new ImageContour should be associated, as Image

ImageContour (Image image, int nbOfContourValues)

This constructor creates a new ImageContour, associated with the given image

and with the given number of contour levels.

Arguments

`Image` **`image`** `[INPUT, MANDATORY, default=no default value]`

The image with which the new ImageContour should be associated, as Image

`int` **`nbOfContourValues`** `[INPUT, MANDATORY, default=no default value]`

The number of contour values the new ImageContour should have, as int

Error

`IllegalArgumentException`

If the given number of contour values for the new ImageContour is not positive, an IllegalArgumentException is thrown.

ImageContour (Image image, int nbOfContourLevels, double[] extremeContourValues, String distribution)

This constructor creates a new ImageContour.

This constructor creates a new ImageContour, associated with the given image and with the given number of contour levels, the given extreme contour values and the given distribution of the contour levels.

Arguments

`Image` **`image`** `[INPUT, MANDATORY, default=no default value]`

The image with which the new ImageContour should be associated, as Image

`int` **`nbOfContourLevels`** `[INPUT, MANDATORY, default=no default value]`

The number of ContourLevels the new ImageContour should have, as int

`double[]` **`extremeContourValues`** `[INPUT, MANDATORY, default=no default value]`

The extreme contour values the new ImageContour should have, as double[]

`String` **`distribution`** `[INPUT, MANDATORY, default=no default value]`

The distribution the ContourLevels of the new ImageContour should have, as String

Error

`IllegalArgumentException`

If the given number of contour levels for the new ImageContour is not positive, an IllegalArgumentException is thrown.

If the given minimum contour values for the new ImageContour is not strictly smaller than the maximum contour value forthe new ImageContour.

If the given distribution of the contour levels for the new ImageContour does not equal lin, log or ln.

Methods

addContourLevel (ContourLevel contourLevel, double contourValue)

Adds the given ContourLevel for the given contour value to this ImageContour.

Arguments

`ContourLevel` **`contourLevel`** `[INPUT, MANDATORY, default=no default value]`

The ContourLevel to add to this ImageContour, as ContourLevel

`double` **`contourValue`** `[INPUT, MANDATORY, default=no default value]`

The contour value for which you wish to add the given ContourLevel to this ImageContour, as double

addContourLevel (ContourLevel contourLevel, String key)

Adds the given ContourLevel to this ImageContour with the given key.

Arguments

`ContourLevel` **`contourLevel`** `[INPUT, MANDATORY, default=no default value]`

The ContourLevel to add to this ImageContour, as ContourLevel

`String` **`key`** `[INPUT, MANDATORY, default=no default value]`

The key for the given ContourLevel in this ImageContour, as String

setWcs (Wcs wcs)

Sets the Wcs for this ImageContour.

Argument

`Wcs` **`wcs`** `[INPUT, MANDATORY, default=no default value]`

The Wcs to set as the Wcs for this ImageContour, as Wcs

Wcs **getWcs**

Returns the Wcs for this ImageContour.

Return

Wcs

Returns the Wcs for this ImageContour.

boolean **hasWcs**

Checks whether a Wcs is attached to this ImageContour.

Returns true if a Wcs is attached to this ImageContour; false otherwise.

Return

boolean

Returns true if a Wcs is attached to this ImageContour; false otherwise.

boolean **hasValidWcs**

Checks whether a valid Wcs is attached to this ImageContour.

Returns true if a valid Wcs is atached to this ImageContour; false otherwise.

Return

boolean

***boolean* hasValidWcs**
Returns true if a valid Wcs is attached to this ImageContour; false otherwise.

See also

- Developers Manual: herschel.ia.dataset.image.ImageContour

1.193. imageContourSaver

Full Name:	herschel.ia.toolbox.image.ImageContourSaverTask
Alias:	imageContourSaver
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import ImageContourSaverTask
Category	Images/Analysis

Description

This is a task to save image contours as a ds9 region file.

This is a task to save image contours as a ds9 region file. You must specify whether you want to save the pixel or sky coordinates and which filename you want to use for that. The file will get the .con extension. Note that you can only store the sky coordinates of the image contour, if the image for which the contours have been generated, had a valid Wcs attached to it. To explain the structure of the saved file, we explain the structure of the image contour product. An image contour product can hold multiple contour levels, and each of these holds the contours of one contour value. One of these contours is basically a list of position that make up the contour. In the saved file, we allocate one line per position, and contours are separated by a blank line. There is no distiction between contours belonging to different contour levels, but they are treated the same way. The reason that 1 is added to the pixel coordinates, is to make is compatible with other programmes like ds9, where counting starts at 1 (instead at 0, like in java).

Examples

Example 1: This is how you can save (the sky coordinates of) an image contour as "contours.con" :

```
imageContourSaver(contours = myContours, filename = "contours", sky = True)
```

Example 2: This is how you can save (the pixel coordinates of) an image contour as "contours.con" :

```
imageContourSaver(contours = myContours, filename = "contours", sky = False)
```

API Summary

Properties
ImageContour **contours** [INPUT, MANDATORY, default=None]
boolean **sky** [INPUT, OPTIONAL, default=true]
String **filename** [INPUT, MANDATORY, default=None]

API details

Properties

ImageContour contours [INPUT, MANDATORY, default=None]

This is the input image contour. This is the output of one of the contour tasks (ContourTask, AutomaticContourTask, ManualContourTask).

boolean sky [INPUT, OPTIONAL, default=true]

This indicates whether to save the sky coordinates (if true) or the pixel coordinates (when false).

String filename [INPUT, MANDATORY, default=None]

This is the filename to use to save the input image contour. The file will get the .con extension.

See also

- Developers Manual: herschel.ia.toolbox.image.ImageContourSaverTask

1.194. imageConvolution

Full Name:	herschel.ia.toolbox.image.ImageConvolutionTask
Alias:	imageConvolution
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import ImageConvolutionTask
Category	Images/Analysis

Description

This is a task to convolve an image with a given convolution kernel of the same dimensions.

This is a task to convolve an image with a given convolution kernel of the same dimensions. This convolution kernel is typically a PSF-image (point-spread-function). The resulting convolved image has the same dimensions, Wcs, coverage, flag,... as the input image. The task uses the convolution theorem. A Fast Fourier Transform (FFT) is applied to the image and the convolution kernel. Then these are multiplied and the inverse FFT is applied. The result is shifted back to the centre and the real part (of the inverse FFT) is used as image data for the output (convolved) image of the task. In case NaNs occur in the image or in the convolution kernel, we replace them by zeroes. The flux conservation contstraint is no longer existing.

Example

Example 1: This is how you can convolve an image with a PSF :

```
convolution = imageConvolution(image = myImage, kernel = myPsf)
```

API Summary

Properties
Image **image** [INPUT, MANDATORY, default=None]
Image **kernel** [INPUT, MANDATORY, default=None]
Image **convolution** [OUTPUT, MANDATORY, default=None]

API details

Properties

Image image [INPUT, MANDATORY, default=None]

The input image.

Image kernel [INPUT, MANDATORY, default=None]

The convolution kernel.

Image convolution [OUTPUT, MANDATORY, default=None]

The resulting convolved image.

See also

- Developers Manual: herschel.ia.toolbox.image.ImageConvolutionTask

1.195. imageDivide

Full Name:	herschel.ia.toolbox.image.ImageDivideTask
Alias:	imageDivide
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import ImageDivideTask
Category	Images/Analysis

Description

This is a task to divide two images pixel-by-pixel (ref = 0) or based on their Wcs (ref = 1), or to divide all intensity values in an image by a scalar.

If two images are to be divided based on their Wcs, they are regridded onto the spatial grid of the smallest image. If two images are divided, the unit of the resulting division will be the division of the units of the input images. If a scalar was used for the division, the unit of the resulting quotient will be the unit of the input image.

Examples

Example 1: This is an example of how to divide two images, pixel-by-pixel :

```
quotient = imageDivide(image1 = myImage1, image2 = myImage2, ref = 0)
```

Example 2: This is an example of how to divide two images, Wcs-based :

```
quotient = imageDivide(image1 = myImage1, image2 = myImage2, ref = 1)
```

Example 3: This is an example of how to divide an image by a scalar :

```
quotient = imageDivide(image1 = myImage, scalar = 5.0)
```

API Summary

Properties
Image **image1** [INPUT, MANDATORY, default=None]
Image **image2** [INPUT, OPTIONAL, default=None]
Integer **ref** [INPUT, OPTIONAL, default=None]
Number **scalar** [INPUT, OPTIONAL, default=None]
Image **quotient** [OUTPUT, MANDATORY, default=None]

API details

Properties

Image image1 [INPUT, MANDATORY, default=None]

This is the image dividend.

Image image2 [INPUT, OPTIONAL, default=None]

This is the image divisor.

Integer ref [INPUT, OPTIONAL, default=None]

This is the reference frame for the calculation of the quotient. Possible values are 0 for pixel-by-pixel based division, and 1 for Wcs based division.

Number scalar [INPUT, OPTIONAL, default=None]

This is the scalar divisor.

Image quotient [OUTPUT, MANDATORY, default=None]

This is the resulting quotient.

See also

- Developers Manual: herschel.ia.toolbox.image.ImageDivideTask

1.196. imageExp10

Full Name:	herschel.ia.toolbox.image.ImageExp10Task
Alias:	imageExp10
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import ImageExp10Task
Category	Images/Analysis

Description

This is a task that allows to change the intensity values of an image according to a exp10 scaling.

This is a task that allows to change the intensity values of an image according to a exp10 scaling.

Example

Example 1: This is an example of how to use the imageExp10 :

```
exp10 = imageExp10(image = myImage)
```

API Summary

Properties
Image.class **image** [INPUT, MANDATORY, default=None]
Image.class **exp10** [OUTPUT, MANDATORY, default=None]

API details

Properties

Image.class image [INPUT, MANDATORY, default=None]

This is the input image.

Image.class exp10 [OUTPUT, MANDATORY, default=None]

This is the output exp10 scaled image.

See also

- Developers Manual: herschel.ia.toolbox.image.ImageExp10Task

1.197. imageExp

Full Name:	herschel.ia.toolbox.image.ImageExpTask
Alias:	imageExp
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import ImageExpTask
Category	Images/Analysis

Description

This is a task that allows to change the intensity values of an image according to a exp scaling.

This is a task that allows to change the intensity values of an image according to a exp scaling.

Example

Example 1: This is an example of how to use the imageExp :

```
exp = imageExp(image = myImage)
```

API Summary

Properties
Image.class **image** [INPUT, MANDATORY, default=None]
Image.class **exp** [OUTPUT, MANDATORY, default=None]

API details

Properties

Image.class image [INPUT, MANDATORY, default=None]

This is the input image.

Image.class exp [OUTPUT, MANDATORY, default=None]

This is the output exp scaled image.

See also

- Developers Manual: herschel.ia.toolbox.image.ImageExpTask

1.198. imageExpN

Full Name:	herschel.ia.toolbox.image.ImageExpNTask
Alias:	imageExpN
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import ImageExpNTask
Category	Images/Analysis

Description

This is a task that allows to change the intensity values of an image according to a expN scaling.

This is a task that allows to change the intensity values of an image according to a expN scaling.

Example

Example 1: This is an example of how to use the imageExpN :

```
expN = imageExpN(image = myImage, n = 2.0)
```

API Summary

Properties
Image.class **image** [INPUT, MANDATORY, default=None]
Double.class **n** [INPUT, MANDATORY, default=2.0]
Image.class **expN** [OUTPUT, MANDATORY, default=None]

API details

Properties

Image.class image [INPUT, MANDATORY, default=None]

This is the input image.

Double.class n [INPUT, MANDATORY, default=2.0]

This is the base of the exponential.

Image.class expN [OUTPUT, MANDATORY, default=None]

This is the output expN scaled image.

See also

- Developers Manual: herschel.ia.toolbox.image.ImageExpNTask

1.199. imageFloor

Full Name:	herschel.ia.toolbox.image.ImageFloorTask
Alias:	imageFloor
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import ImageFloorTask
Category	Images/Analysis

Description

This is a task that floors the intensity values of an image.

This is a task that floors the intensity values of an image.

Example

Example 1: This is an example of how to use the imageFloor :

```
floored = imageFloor(image = myImage)
```

API Summary

Properties
`Image image [INPUT, MANDATORY, default=None]`
`Image floored [OUTPUT, MANDATORY, default=None]`

API details

Properties

`Image image [INPUT, MANDATORY, default=None]`

This is the input image.

`Image floored [OUTPUT, MANDATORY, default=None]`

This is the output floored image.

See also

- Developers Manual: herschel.ia.toolbox.image.ImageFloorTask

1.200. imageHistogram

Full Name:	herschel.ia.toolbox.image.ImageHistogramTask
Alias:	imageHistogram
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import ImageHistogramTask
Category	Images/Analysis

Description

This is a task to make a histogram of an image as a whole.

This is a task to make a histogram of an image as a whole. You must specify the number of bins for the histogram. By default the cut levels of the image will be used to construct the histogram, but it is also possible for you to specify the low and high cut level for the histogram.

Example

Example 1: This is an example of how to use the imageHistogram :

```
histogram = imageHistogram(image = myImage, bins = 200, lowCut = 100.0, highCut
 = 150.0)
```

API Summary

Properties
Image **image** [INPUT, MANDATORY, default=None]
Double **lowCut** [INPUT, OPTIONAL, default=NaN]
Double **highCut** [INPUT, OPTIONAL, default=NaN]
Integer **bins** [INPUT, OPTIONAL, default=2000]
ImageHistogramProduct **histogram** [OUTPUT, MANDATORY, default=None]

API details

Properties

Image image [INPUT, MANDATORY, default=None]

This is the input image.

Double lowCut [INPUT, OPTIONAL, default=NaN]

This is the low cut level you want to use for the histogram. If this is not specified, the image cut levels will be used.

Double highCut [INPUT, OPTIONAL, default=NaN]

This is the high cut level you want to use for the histogram. If this is not specified, the image cut levels will be used.

Integer bins [INPUT, OPTIONAL, default=2000]

This is the number of bins in the histogram.

ImageHistogramProduct histogram [OUTPUT, MANDATORY, default=None]
This is the resulting histogram.

See also

- Developers Manual: herschel.ia.toolbox.image.ImageHistogramTask

1.201. ImageHistogramProduct

Full Name:	herschel.ia.dataset.image.ImageHistogramProduct
Type:	Java Class -
Import:	from herschel.ia.dataset.image import ImageHistogramProduct
Category	Images/Display

Description

A class to deal with the results of an image histogram.

API Summary

Constructor
`ImageHistogramProduct` The constructor of a new ImageHistogramProduct.

Methods
`setCutLevels (double lowCut, double highCut)` Sets the cut levels for this ImageHistogramProduct to the given cut levels.
`setNbOfBins (int bins)` Sets the number of bins for this ImageHistogramProduct to the given number of bins.
`setHistogram (Double1d values, Double1d frequencies)` Sets the histogram for this ImageHistogramProduct to a histogram with the given values and frequencies.
`setUnit (Unit unit)` Sets the unit for this ImageHistogramProduct to the given unit.
`double` **`getLowCut`** Returns the minimum cut level for this ImageHistogramProduct.
`double` **`getHighCut`** Returns the maximum cut level for this ImageHistogramProduct.
`int` **`getNbOfBins`** Returns the number of bins for this ImageHistogramProduct.
`TableDataset` **`getHistogram`** Returns the histogram for this ImageHistogramProduct.
`Double1d` **`getValues`** Returns the values for the histogram for this ImageHistogramProduct.
`Double1d` **`getFrequencies`** Returns the frequencies for the histogram for this ImageHistogramProduct.
`String` **`getUnit`** Returns the unit for this ImageHistogramProduct.

API Details

Constructor

ImageHistogramProduct

The constructor of a new ImageHistogramProduct.

Methods

setCutLevels (double lowCut, double highCut)

Sets the cut levels for this ImageHistogramProduct to the given cut levels.

Arguments

`double` **lowCut** `[INPUT, MANDATORY, default=no default value]`

The minimum cut level, as double

`double` **highCut** `[INPUT, MANDATORY, default=no default value]`

The maximum cut level, as double

setNbOfBins (int bins)

Sets the number of bins for this ImageHistogramProduct to the given number of bins.

Argument

`int` **bins** `[INPUT, MANDATORY, default=no default value]`

The number of bins, as int

setHistogram (Double1d values, Double1d frequencies)

Sets the histogram for this ImageHistogramProduct to a histogram with the given values and frequencies.

Arguments

`Double1d` **values** `[INPUT, MANDATORY, default=no default value]`

The values, as Double1d

`Double1d` **frequencies** `[INPUT, MANDATORY, default=no default value]`

The frequencies, as Double1d

setUnit (Unit unit)

Sets the unit for this ImageHistogramProduct to the given unit.

Argument

`Unit` **unit** `[INPUT, MANDATORY, default=no default value]`

The unit to set as the unit for this ImageHistogramProduct, as Unit

double **getLowCut**

Returns the minimum cut level for this ImageHistogramProduct.

Return

double

Returns the minimum cut level for this ImageHistogramProduct.

double **getHighCut**

Returns the maximum cut level for this ImageHistogramProduct.

***double* getHighCut**

Return

double

Returns the maximum cut level for this ImageHistogramProduct.

***int* getNbOfBins**

Returns the number of bins for this ImageHistogramProduct.

Return

int

Returns the number of bins for this ImageHistogramProduct.

***TableDataset* getHistogram**

Returns the histogram for this ImageHistogramProduct.

Return

TableDataset

Returns the histogram for this ImageHistogramProduct.

***Double1d* getValues**

Returns the values for the histogram for this ImageHistogramProduct.

Return

Double1d

Returns the values for the histogram for this

***Double1d* getFrequencies**

Returns the frequencies for the histogram for this ImageHistogramProduct.

Return

Double1d

Returns the frequencies for the histogram for this ImageHistogramProduct.

***String* getUnit**

Returns the unit for this ImageHistogramProduct.

Return

String

Returns the unit for this ImageHistogramProduct.

See also

- Developers Manual: herschel.ia.dataset.image.ImageHistogramProduct

1.202. imageLog10

Full Name:	herschel.ia.toolbox.image.ImageLog10Task
Alias:	imageLog10
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import ImageLog10Task
Category	Images/Analysis

Description

This is a task that allows to change the intensity values of an image according to a log10 scaling.

This is a task that allows to change the intensity values of an image according to a log10 scaling.

Example

Example 1: This is an example of how to use the imageLog10 :

```
log10 = imageLogN(image = myImage)
```

API Summary

Properties
Image.class **image** [INPUT, MANDATORY, default=None]
Image.class **log10** [OUTPUT, MANDATORY, default=None]

API details

Properties

Image.class image [INPUT, MANDATORY, default=None]

This is the input image.

Image.class log10 [OUTPUT, MANDATORY, default=None]

This is the output log10 scaled image.

See also

- Developers Manual: herschel.ia.toolbox.image.ImageLog10Task

1.203. imageLog

Full Name:	herschel.ia.toolbox.image.ImageLogTask
Alias:	imageLog
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import ImageLogTask
Category	Images/Analysis

Description

This is a task that allows to change the intensity values of an image according to a log scaling.

This is a task that allows to change the intensity values of an image according to a log scaling.

Example

Example 1: This is an example of how to use the imageLog :

```
log = imageLogN(image = myImage)
```

API Summary

Properties
Image.class **image** [INPUT, MANDATORY, default=None]
Image.class **log** [OUTPUT, MANDATORY, default=None]

API details

Properties

Image.class image [INPUT, MANDATORY, default=None]

This is the input image.

Image.class log [OUTPUT, MANDATORY, default=None]

This is the output log scaled image.

See also

- Developers Manual: herschel.ia.toolbox.image.ImageLogTask

1.204. imageLogN

Full Name:	herschel.ia.toolbox.image.ImageLogNTask
Alias:	imageLogN
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import ImageLogNTask
Category	Images/Analysis

Description

This is a task that allows to change the intensity values of an image according to a logN scaling.

This is a task that allows to change the intensity values of an image according to a logN scaling.

Example

Example 1: This is an example of how to use the imageLogN :

```
logN = imageLogN(image = myImage, n = 2.0)
```

API Summary

Properties
Image.class **image** [INPUT, MANDATORY, default=None]
Double.class **n** [INPUT, MANDATORY, default=2.0]
Image.class **logN** [OUTPUT, MANDATORY, default=None]

API details

Properties

Image.class image [INPUT, MANDATORY, default=None]

This is the input image.

Double.class n [INPUT, MANDATORY, default=2.0]

This is the base of the logarithm.

Image.class logN [OUTPUT, MANDATORY, default=None]

This is the output logN scaled image.

See also

- Developers Manual: herschel.ia.toolbox.image.ImageLogNTask

1.205. imageModulo

Full Name:	herschel.ia.toolbox.image.ImageModuloTask
Alias:	imageModulo
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import ImageModuloTask
Category	Images/Analysis

Description

This is a task that allows to take the modulo of an image, either with another image or with a scalar.

Examples

Example 1: This is an example of how to take the remainder of division of two images, pixel-to-pixel :

```
remainder = imageModulo(image1 = myImage1, image2 = myImage2, ref = 0)
```

Example 2: This is an example of how to take the remainder of division of two images, Wcs :

```
remainder = imageModulo(image1 = myImage1, image2 = myImage2, ref = 1)
```

Example 3: This is an example of how to take the remainder of division of an image by a scalar :

```
remainder = imageModulo(image1 = myImage, scalar = 5.0)
```

API Summary

Properties
Image **image1** [INPUT, MANDATORY, default=None]
Image **image2** [INPUT, OPTIONAL, default=None]
Integer **ref** [INPUT, OPTIONAL, default=None]
Number **scalar** [INPUT, OPTIONAL, default=None]
Image **remainder** [OUTPUT, MANDATORY, default=None]

API details

Properties

Image image1 [INPUT, MANDATORY, default=None]

This is the image dividend.

Image image2 [INPUT, OPTIONAL, default=None]

This is the image divisor.

Integer ref [INPUT, OPTIONAL, default=None]

This is the reference frame for the calculation of the remainder. Possible values are 0 for pixel-to-pixel based division, and 1 for Wcs based division.

Number scalar [INPUT, OPTIONAL, default=None]
This is the scalar divisor.

Image remainder [OUTPUT, MANDATORY, default=None]
This is the resulting remainder after division.

See also

- Developers Manual: herschel.ia.toolbox.image.ImageModuloTask

1.206. imageMultiply

Full Name:	herschel.ia.toolbox.image.ImageMultiplyTask
Alias:	imageMultiply
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import ImageMultiplyTask
Category	Images/Analysis

Description

This is a task to multiply two images or an image and a scalar.

Available operation modes are pixel-by-pixel (ref = 0) or based on their Wcs (ref = 1), or to multiply all intensity values in an image with a scalar. If two images are to be multiplied based on their Wcs, they are regridded onto the spatial grid of the smallest image. If two images are multiplied, the unit of the resulting product will be the product of the units of the input images. If a scalar was used for the multiplication, the unit of the resulting product will be the unit of the input image.

Examples

Example 1: This is an example of how you can multiply two images, pixel-by-pixel :

```
product = imageMultiply(image1 = myImage1, image2 = myImage2, ref = 0)
```

Example 2: This is an example of how you can multiply two images, Wcs based :

```
product = imageMultiply(image1 = myImage1, image2 = myImage2, ref = 1)
```

Example 3: This is a example of of to multiply an image with a scalar

```
result = imageMultiply(image1 = myImage, scalar = 5.0)
```

API Summary

Properties
Image **image1** [INPUT, MANDATORY, default=None]
Image **image2** [INPUT, OPTIONAL, default=None]
Integer **ref** [INPUT, OPTIONAL, default=None]
Number **scalar** [INPUT, OPTIONAL, default=None]
Image **product** [OUTPUT, MANDATORY, default=None]

API details

Properties

Image image1 [INPUT, MANDATORY, default=None]

This is the image multiplier.

Image image2 [INPUT, OPTIONAL, default=None]

This is image multiplicand.

Integer ref [INPUT, OPTIONAL, default=None]

This is the reference frame for the calculation of the product. Possible values are 0 for pixel-by-pixel based multiplication, and 1 for Wcs based multiplication.

Number scalar [INPUT, OPTIONAL, default=None]

This is the scalar multiplicand.

Image product [OUTPUT, MANDATORY, default=None]

This is the resulting product.

See also

- Developers Manual: herschel.ia.toolbox.image.ImageMultiplyTask

1.207. imagePower

Full Name:	herschel.ia.toolbox.image.ImagePowerTask
Alias:	imagePower
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import ImagePowerTask
Category	Images/Analysis

Description

This is a task that changes the intensity values of an image according to a power scaling.

This is a task that changes the intensity values of an image according to a power scaling.

Example

Example 1: This is an example of how to use the imagePower :

```
powered = imagePower(image = myImage, n = 2.0)
```

API Summary

Properties
Image **image** [INPUT, MANDATORY, default=None]
Double **n** [INPUT, OPTIONAL, default=2.0]
Image **powered** [OUTPUT, MANDATORY, default=None]

API details

Properties

Image image [INPUT, MANDATORY, default=None]

This is the input image.

Double n [INPUT, OPTIONAL, default=2.0]

This is the exponent.

Image powered [OUTPUT, MANDATORY, default=None]

This is the output power scaled image.

See also

- Developers Manual: herschel.ia.toolbox.image.ImagePowerTask

1.208. imageRound

Full Name:	herschel.ia.toolbox.image.ImageRoundTask
Alias:	imageRound
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import ImageRoundTask
Category	Images/Analysis

Description

This is a task that rounds the intensity values of an image.

This is a task that rounds the intensity values of an image.

Example

Example 1: This is an example of how to use the imageRoubd :

```
rounded = imageRound(image = myImage)
```

API Summary

Properties
`Image image [INPUT, MANDATORY, default=None]`
`Image rounded [OUTPUT, MANDATORY, default=None]`

API details

Properties

`Image image [INPUT, MANDATORY, default=None]`

This is the input image.

`Image rounded [OUTPUT, MANDATORY, default=None]`

This is the output rounded image.

See also

- Developers Manual: herschel.ia.toolbox.image.ImageRoundTask

1.209. imageSaver

Full Name:	herschel.ia.toolbox.image.ImageSaverTask
Alias:	imageSaver
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import ImageSaverTask
Categories	Images Input-output

Description

This is a task which creates a grey colors JPEG file from an image.

This is a task which creates a grey colors JPEG file from an image, not taking into account annotations or cut levels.

Example

Example 1: This is how you can save an image :

```
from herschel.ia.toolbox.image import ImageSaverTask
ImageSaverTask()(image = myImage, filename = "test.jpg")
```

API Summary

Jython Syntax

```
ImageSaverTask()(image, filename)
```

Properties

```
Image image [INPUT, MANDATORY, default=None]
String filename [INPUT, MANDATORY, default=None]
```

API details

Properties

`Image image [INPUT, MANDATORY, default=None]`

This is the input image

`String filename [INPUT, MANDATORY, default=None]`

This is the filename to use to store the image.

See also

- Developers Manual: herschel.ia.toolbox.image.ImageSaverTask

1.210. imageSqrt

Full Name:	herschel.ia.toolbox.image.ImageSqrtTask
Alias:	imageSqrt
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import ImageSqrtTask
Category	Images/Analysis

Description

This is a task that changes the intensity values of an image according to a sqrt scaling.

This is a task that changes the intensity values of an image according to a sqrt scaling.

Example

Example 1: This is an example of how to use the imageSqrt :

```
sqrt = imageSqrt(image = myImage)
```

API Summary

Properties
`Image image [INPUT, MANDATORY, default=None]`
`Image sqrt [OUTPUT, MANDATORY, default=None]`

API details

Properties

`Image image [INPUT, MANDATORY, default=None]`

This is the input image.

`Image sqrt [OUTPUT, MANDATORY, default=None]`

This is the output sqrt scaled image.

See also

- Developers Manual: herschel.ia.toolbox.image.ImageSqrtTask

1.211. imageSquare

Full Name:	herschel.ia.toolbox.image.ImageSquareTask
Alias:	imageSquare
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import ImageSquareTask
Category	Images/Analysis

Description

This is a task that changes the intensity values of an image according to a square scaling.

This is a task that changes the intensity values of an image according to a square scaling.

Example

Example 1: This is a example of how to use the imageSquare :

```
square = imageSquare(image = myImage)
```

API Summary

Properties
Image **image** [INPUT, MANDATORY, default=No default value]
Image **square** [OUTPUT, MANDATORY, default=No default value]

API details

Properties

Image image [INPUT, MANDATORY, default=No default value]

This is the input image.

Image square [OUTPUT, MANDATORY, default=No default value]

This is the output square scaled image.

See also

- Developers Manual: herschel.ia.toolbox.image.ImageSquareTask

1.212. imageSubtract

Full Name:	herschel.ia.toolbox.image.ImageSubtractTask
Alias:	imageSubtract
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import ImageSubtractTask
Category	Images/Analysis

Description

This is a task to subtract two images pixel-by-pixel (ref = 0) or based on their Wcs (ref = 1), or to subtract a scalar from all intensity values in an image.

This is a task to subtract two images pixel-by-pixel (ref = 0) or based on their Wcs (ref = 1), or to subtract a scalar from all intensity values in an image. If two images are to be subtracted based on their Wcs, they are regridded onto the spatial grid of the smallest image. If two images are subtracted, the unit of the resulting difference will only be set only if both input images have the same unit. Their unit will then be used as the unit of the resulting sum. If a scalar was used for the subtraction, the unit of the resulting sum will be the unit of the input image.

Examples

Example 1: This is an example of how to subtract two images, pixel-by-pixel :

```
difference = imageSubtract(image1 = myImage1, image2 = myImage2, ref = 0)
```

Example 2: This is an example of how to subtract two images, Wcs based :

```
difference = imageSubtract(image1 = myImage1, image2 = myImage2, ref = 1)
```

Example 3: This is an example of how to subtract a scalar from an image :

```
difference = imageSubtract(image1 = myImage, scalar = 5.0)
```

API Summary

Properties
Image **image1** [INPUT, MANDATORY, default=None]
Image **image2** [INPUT, OPTIONAL, default=None]
Integer **ref** [INPUT, OPTIONAL, default=None]
Number **scalar** [INPUT, OPTIONAL, default=None]
Image **difference** [OUTPUT, OPTIONAL, default=None]

API details

Properties

Image image1 [INPUT, MANDATORY, default=None]

This is the image minuend.

Image image2 [INPUT, OPTIONAL, default=None]

This is the image subtrahend.

Integer ref [INPUT, OPTIONAL, default=None]

This is the reference frame for the calculation of the difference. Possible values are 0 for pixel-by-pixel based subtraction, and 1 for Wcs based subtraction.

Number scalar [INPUT, OPTIONAL, default=None]

This is scalar subtrahend.

Image difference [OUTPUT, OPTIONAL, default=None]

This is the resulting difference.

See also

- Developers Manual: herschel.ia.toolbox.image.ImageSubtractTask

1.213. importCube

Full Name:	herschel.ia.toolbox.cube.ImportCubeTask
Alias:	importCube
Type:	Java Task -
Import:	from herschel.ia.toolbox.cube import ImportCubeTask
Categories	Data cubes Input-output

Description

The ImportCube task for Cubes.

A task which imports a Cube from a fits file

API Summary

Jython Syntax
`importCube(cube, filename)`

Properties
`SimpleCube simplecube [INOUT, MANDATORY, default=No default value]`
`String filename [INPUT, MANDATORY, default=no default value]`

API details

Properties

`SimpleCube simplecube [INOUT, MANDATORY, default=No default value]`

The input Cube to fill

`String filename [INPUT, MANDATORY, default=no default value]`

The file to import

See also

- Developers Manual: herschel.ia.toolbox.cube.ImportCubeTask

1.214. importImage

Full Name:	herschel.ia.toolbox.image.ImportImageTask
Alias:	importImage
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import ImportImageTask
Categories	Images Input-output

Description

This is a task to import an image from a file.

This is a task to import an image from a file (bmp, fpx, gif, jpg, png, pnm, tiff or fits format).

Example

Example 1: This is how you can import a file into an image

```
path = "testImage/filename.fits"
myImage = importImage(filename = path)
```

API Summary

Properties
`Image image [INOUT, OPTIONAL, default=None]`
`string filename [INPUT, MANDATORY, default=None]`

API details

Properties

`Image image [INOUT, OPTIONAL, default=None]`

This is the input image, in which the file must be loaded.

`string filename [INPUT, MANDATORY, default=None]`

This is the file to import.

See also

- Developers Manual: herschel.ia.toolbox.image.ImportImageTask

1.215. importRgbImage

Full Name:	herschel.ia.toolbox.image.ImportRgbImageTask
Alias:	importRgbImage
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import ImportRgbImageTask
Categories	Images Input-output

Description

This is a task to import an RGB image from a file.

This is a task to import an RGB image from a file (bmp, fpx, gif, jpg, png, pnm, tiff or fits format).

Example

Example 1: This is how you can import a file into an image

```
path = "testRgbImage/filename.fits"
myImage = importRgbImage(filename = path)
```

API Summary

Properties
`Image image [INOUT, OPTIONAL, default=None]`
`string filename [INPUT, MANDATORY, default=None]`

API details

Properties

`Image image [INOUT, OPTIONAL, default=None]`

This is the input RGB image, in which the file must be loaded.

`string filename [INPUT, MANDATORY, default=None]`

This is the file to import.

See also

- Developers Manual: herschel.ia.toolbox.image.ImportRgbImageTask

1.216. importSpectralCube

Full Name:	herschel.ia.toolbox.cube.ImportSpectralCubeTask
Alias:	importSpectralCube
Type:	Java Task -
Import:	from herschel.ia.toolbox.cube import ImportSpectralCubeTask
Categories	Data cubes Input-output

Description

The ImportSpectralCube task for SpectralSimpleCubes.

A task which imports a SpectralSimpleCube from a fits file

API Summary

Jython Syntax
`importSpectralCube(spectralcube, filename)`

Properties
`SpectralSimpleCube` **`spectralcube`** `[INOUT, MANDATORY, default=No default value]`
`String` **`filename`** `[INPUT, MANDATORY, default=no default value]`

API details

Properties

`SpectralSimpleCube spectralcube [INOUT, MANDATORY, default=No default value]`
The input SpectralCube to fill

`String filename [INPUT, MANDATORY, default=no default value]`
The file to import

See also

- Developers Manual: herschel.ia.toolbox.cube.ImportSpectralCubeTask

1.217. importSpectrumFromAscii

Full Name:	herschel.ia.toolbox.spectrum.ImportSpectrumFromAsciiTask
Alias:	importSpectrumFromAscii
Type:	Java Task -
Import:	from herschel.ia.toolbox.spectrum import ImportSpectrumFromAsciiTask
Category	Spectra/Analysis

Description

Task for importing spectrum data from an ASCII file into a HCSS spectrum data structure.

If more than one spectrum is found in the data a {@link Spectrum2d} is returned, otherwise a {@link Spectrum1d}. The task also allows to populate the meta data: Setting the `meta` parameter to `true` implies that the header lines starting with "#" are parsed as meta data. The following forms are valid: # hasSegments = true # equinox = 2000.0 # odNumber = 461 # nyquistSampling = {description="Map is Nyquist sampled", boolean=false} # v_frame = {description="Velocity of the source w.r.t. frame", unit="km/s", double=34.5} Possible value types are `long, double, string, boolean`.

By setting `wavename`, `fluxname`, `weightname` and `flagname` the 'wave', 'flux', 'weight' and 'flag' columns are identified. The names columns found in the ASCII file are then assumed to start with these names to be identified as wave-, flux-, weight- or flag-data. The ending of the column names can be used to encode segments or different (point) spectra (see {@link PointSpectrum}). This additional information is separated by underscores ("_"), the segment indices are identified with a trailing "segm". Examples:

- flux_segm2_3: flux column associated with segment "2" and point spectrum id 3.
- wave_2: wave column associated with point spectrum id 2 (no or just a single segment).
- flux: just a single flux

Note that the naming of the columns needs to be consistent. Furthermore, note that the name of the wave column in the {@link Spectrum1d} or {@link Spectrum2d} is the wavename found in the meta data (once `meta` is set to True) or 'wave'.

In case the ASCII file contains a single spectrum (possibly consisting of several segments) a {@link Spectrum1d} - otherwise a {@link Spectrum2d} is generated.

Example

Example 1: import the spectra found in the given ascii file

```
global spectra # defined elsewhere
# import CSV file as exported by exportSpectrumToAscii
spectra = importSpectrumFromAscii(file="spectra-1.csv")
# import CSV file with two columns of which the first 4 rows are as follows:
#   x,y
#   Double,Double
#   GHz,W m-2 Hz-1 sr-1
#   Some frequency, And some flux
#
spectra = importSpectrumFromAscii(file="spectra-2.csv", meta=False,
 wavename="x", fluxname="y")
```

API Summary

Properties
`String` **`file`** `[INPUT, MANDATORY, default=no default value.]`

Properties
SpectrumContainer **ds** [OUTPUT, OPTIONAL, default=no default value.]
String **fluxname** [INPUT, OPTIONAL, default="flux".]
String **wavename** [INPUT, OPTIONAL, default=wavename from meta data (if available) or "wave".]
String **flagname** [INPUT, OPTIONAL, default="flag".]
String **weightname** [INPUT, OPTIONAL, default="weight".]
Boolean **meta** [INPUT, OPTIONAL, default=True.]
String **parserDelim** [INPUT, OPTIONAL, default=comma separator.]

API details

Properties

String file [INPUT, MANDATORY, default=no default value.]

The location and name of the file to import.

SpectrumContainer ds [OUTPUT, OPTIONAL, default=no default value.]

The imported spectrum data.

String fluxname [INPUT, OPTIONAL, default="flux".]

The name of the column to be interpreted as "flux" columns.

String wavename [INPUT, OPTIONAL, default=wavename from meta data (if available) or "wave".]

The name of the column to be interpreted as "wave" columns.

String flagname [INPUT, OPTIONAL, default="flag".]

The name of the column to be interpreted as "flag" columns.

String weightname [INPUT, OPTIONAL, default="weight".]

The name of the column to be interpreted as "weight" columns.

Boolean meta [INPUT, OPTIONAL, default=True.]

Specifies the comments in the header should be parsed for meta data found in the input data are also included as header information in the output file.

String parserDelim [INPUT, OPTIONAL, default=comma separator.]

Specifies the field separator. If it is one character, a csvParser is selected. If it is an expression, a RegExpParser is selected.

See also

- Developers Manual: herschel.ia.toolbox.spectrum.ImportSpectrumFromAsciiTask

History

- `2011-08-08 - melchior`: renamed from ImportSpectrumFromASCII

1.218. Int1d

Full Name:	herschel.ia.numeric.Int1d
Type:	Java Class -
Import:	from herschel.ia.numeric import Int1d
Category	Arrays and datasets

Description

A rectangular numeric int array of rank 1.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Int1d

1.219. Int2d

Full Name:	herschel.ia.numeric.Int2d
Type:	Java Class -
Import:	from herschel.ia.numeric import Int2d
Category	Arrays and datasets

Description

A rectangular numeric int array of rank 2.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Int2d

1.220. Int3d

Full Name:	herschel.ia.numeric.Int3d
Type:	Java Class -
Import:	from herschel.ia.numeric import Int3d
Category	Arrays and datasets

Description

A rectangular numeric int array of rank 3.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Int3d

1.221. Int4d

Full Name:	herschel.ia.numeric.Int4d
Type:	Java Class -
Import:	from herschel.ia.numeric import Int4d
Category	Arrays and datasets

Description

A rectangular numeric int array of rank 4.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Int4d

1.222. Int5d

Full Name:	herschel.ia.numeric.Int5d
Type:	Java Class -
Import:	from herschel.ia.numeric import Int5d
Category	Arrays and datasets

Description

A rectangular numeric int array of rank 5.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Int5d

1.223. integrateSpectralMap

Full Name:	herschel.ia.toolbox.cube.IntegrateSpectralMapTask
Alias:	integrateSpectralMap
Type:	Java Task -
Import:	from herschel.ia.toolbox.cube import IntegrateSpectralMapTask
Category	Data cubes/Analysis

Description

Integrates a spectral cube over one or several ranges in order to produce a series of integrated maps.

The maps are stored in a SimpleCube. Each layer contains one map. Additionally, the output product contains a LayerInfo table with the start and end of each range.

This task can most easily be run from the Spectrum Explorer. You can then select each range using the range selection tool of the SE. The startOfRange and endOfRange parameter fields are then automatically filled by the SE.

API Summary

Properties
Cube **cube** [INPUT, MANDATORY, default=no default value]
Double1d **startArray** [INPUT, MANDATORY, default=default None]
Double1d **endArray** [INPUT, MANDATORY, default=default None]
SimpleCube **integratedMap** [OUTPUT, MANDATORY, default=default None]

API details

Properties

Cube cube [INPUT, MANDATORY, default=no default value]

The Cube containing the spectra to extract

Double1d startArray [INPUT, MANDATORY, default=default None]

The Array of spectral values for the start of each integration the dimension of this array is the number of integration to execute

Double1d endArray [INPUT, MANDATORY, default=default None]

The Array of spectral values for the end of each integration the dimension of this array is the number of integration to execute

SimpleCube integratedMap [OUTPUT, MANDATORY, default=default None]

The resulting cube

See also

- Developers Manual: herschel.ia.toolbox.cube.IntegrateSpectralMapTask

1.224. Integrator

Full Name:	herschel.ia.numeric.toolbox.integr.Integrator
Alias:	Integrator
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.integr import Integrator
Category	Mathematics/Integration

Description

Interface for all integrators.

Available integrators are: - GaussHermiteIntegrator (closed interval) - GaussianQuad4Integrator (open interval) - GaussianQuad5Integrator (open interval) - GaussJacobiIntegrator (closed interval) - GaussLaguerreIntegrator (closed interval) - GaussLegendreIntegrator (open interval) - RectangularIntegrator (open interval) - TrapezoidalIntegrator (open interval) - SimpsonIntegrator (open interval) - RombergIntegrator (open interval)

Examples

Example 1: Integration using the Romberg's method

```
from herschel.ia.numeric.toolbox import RealFunction
class MyFunction(RealFunction):
    def calc(self,x):
        return x*x
f = MyFunction()
i = RombergIntegrator(-3, 3)
print i.integrate(f)   # 18.0
```

Example 2: Pass parameters to 'RealFunction' for the integrator

```
from herschel.ia.numeric.toolbox import RealFunction
class e(RealFunction):
    def __init__(self, factor):
        self.factor = factor
    def calc(self,x):
        return x**self.factor
f = e(1.1)
integrator=SimpsonIntegrator(0, 2)
print integrator.integrate(f)
# 2.04147326508
```

API Summary

Jython Syntax

<r>=SomeOpenIntervalIntegrator(<a>,<b>).integrate(<f>)

<r>=SomeShutIntervalIntegrator().integrate(<f>)

Properties

double **a** [INPUT, MANDATORY, default=no default value]

double **b** [INPUT, MANDATORY, default=no default value]

RealFunction **f** [INPUT, MANDATORY, default=no default value]

Properties
double **r** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

double a [INPUT, MANDATORY, default=no default value]

Lower limit of integration, only mandatory for open interval integrators.

double b [INPUT, MANDATORY, default=no default value]

Upper limit of integration, only mandatory for open interval integrators. It must be greater or equal than the lower limit.

RealFunction f [INPUT, MANDATORY, default=no default value]

The function must implement the calc method; see example below.

double r [OUTPUT, MANDATORY, default=no default value]

Returns the integral of the given function between the specified limits; that is, the area behind the function within these limits.

See also

- Developers Manual: herschel.ia.numeric.toolbox.integr.Integrator

1.225. IntensityCalculator

Full Name:	herschel.ia.toolbox.image.IntensityCalculator
Type:	Java Class -
Import:	from herschel.ia.toolbox.image import IntensityCalculator
Category	Images/Analysis

Description

This is a class to apply sky estimtion algorithms.

This is a class to estimate the sky intensity through five different sky estimation algorithms : average, median, mean-median, synthetic mode and daophot.

See also

- Developers Manual: herschel.ia.toolbox.image.IntensityCalculator

1.226. IntTabulated

Full Name:	herschel.ia.numeric.toolbox.integr.IntTabulated
Alias:	IntTabulated
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.integr import IntTabulated
Category	Mathematics/Integration

Description

Integrate tabular data similar to IDL's INT_TABULATED.

Example

Example 1: Simple example.

```
x = Double1d.range(100) + 1.0
y = 1.0/x
result = IntTabulated(x)(y)
print result
print LOG(100.0) # the analytic result
# Example 2:
x = Double1d([0.0, .12, .22, .32, .36, .40, .44, .54, .64, .70, .80])
y = Double1d([0.200000, 1.30973, 1.30524, 1.74339, 2.07490, 2.45600, 2.84299,
 3.50730, 3.18194, 2.36302, 0.231964])
result = IntTabulated(x)(y)
print result # exact answer is 1.6405
```

API Summary

Jython Syntax

```
result = intTabulated(x)(y)
```

Properties

an 1d array **x** [INPUT, MANDATORY, default=No default value]

an 1d array **y** [INPUT, MANDATORY, default=No default value]

API details

Properties

an 1d array x [INPUT, MANDATORY, default=No default value]

the input vector of independent variable

an 1d array y [INPUT, MANDATORY, default=No default value]

the input vector of the dependent variable

See also

- http://www.exelisvis.com/docs/INT_TABULATED.html
- Developers Manual: herschel.ia.numeric.toolbox.integr.IntTabulated

History

- `2005-01-01 - IV`: Version 1.0: Coded in DP/jython
- `2012-01-07 - CB`: Version 1.1: Augmented and converted to a Jython Task
- `2012-01-08 - JSS`: Converted to Java

1.227. inverseDFT2d

Full Name:	herschel.ia.toolbox.image.InverseDFT2dTask
Alias:	inverseDFT2d
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import InverseDFT2dTask
Category	Mathematics/Signal processing

Description

This is a task to calculate the inverse Discrete Fourier Transform and the power spectrum of an image.

This is a task to calculate the inverse Discrete Fourier Transform and the power spectrum of an image.

Example

Example 1: How to calculate the inverse Discrete Fourier Transform and the power spectrum of an image :

```
transform = inverseDFT2d(image = myImage)
spectrum = inversDFT2d.spectrum
```

API Summary

Properties
Image **image** [INPUT, MANDATORY, default=None]
Complex2d **transform** [OUTPUT, MANDATORY, default=None]
Double2d **spectrum** [OUTPUT, MANDATORY, default=None]

API details

Properties

Image image [INPUT, MANDATORY, default=None]

This is the input image.

Complex2d transform [OUTPUT, MANDATORY, default=None]

This is the inverse Discrete Fourier Transform.

Double2d spectrum [OUTPUT, MANDATORY, default=None]

This is the power spectrum.

See also

- Developers Manual: herschel.ia.toolbox.image.InverseDFT2dTask

1.228. inverseFFT2d

Full Name:	herschel.ia.toolbox.image.InverseFFT2dTask
Alias:	inverseFFT2d
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import InverseFFT2dTask
Category	Mathematics/Signal processing

Description

This is a task to calculate the inverse Fast Fourier Transform and the power spectrum of an image.

This is a task to calculate the inverse Fast Fourier Transform and the power spectrum of an image.

Example

Example 1: How to calculate the inverse Fast Fourier Transform and the power spectrum of an image :

```
transform = inverseFFT2d(image = myImage)
spectrum = inversFFT2d.spectrum
```

API Summary

Properties
`Image image [INPUT, MANDATORY, default=None]`
`Complex2d transform [OUTPUT, MANDATORY, default=None]`
`Double2d spectrum [OUTPUT, MANDATORY, default=None]`

API details

Properties

`Image image [INPUT, MANDATORY, default=None]`

This is the input image.

`Complex2d transform [OUTPUT, MANDATORY, default=None]`

This is the inverse Fast Fourier Transform.

`Double2d spectrum [OUTPUT, MANDATORY, default=None]`

This is the power spectrum.

See also

- Developers Manual: herschel.ia.toolbox.image.InverseFFT2dTask

1.229. INVERSE

Full Name:	herschel.ia.numeric.toolbox.matrix.MatrixInverse
Alias:	INVERSE
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.matrix import MatrixInverse
Category	Mathematics/Matrices

Description

Returns the inverse of a square matrix.

Example

Example 1: Apply INVERSE to a Float2d matrix

```
x=Float2d([ [1,2],[3,4] ])
print INVERSE(x) # [ [-2.0,1.0], [1.5,-0.5] ]
```

API Summary

Jython Syntax

```
&lt;y&gt;=INVERSE(&lt;x&gt;)
```

Properties

```
any square matrix x [INPUT, MANDATORY, default=no default value]
double y [INPUT, OPTIONAL, default=false]
```

Miscellaneous

Does not work for complex matrices.

API details

Properties

any square matrix x [INPUT, MANDATORY, default=no default value]

Any square matrix

double y [INPUT, OPTIONAL, default=false]

Returns a double

See also

- Developers Manual: herschel.ia.numeric.toolbox.matrix.MatrixInverse

1.230. IS_ANY_NAN

Full Name:	herschel.ia.numeric.toolbox.basic.IsAnyNaN
Alias:	IS_ANY_NAN
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import IsAnyNaN
Category	Arrays and datasets/Element selection

Description

Checks if the given array contains a NaN value.

Example

Example 1: Apply IS_ANY_NAN on a Double1d

```
x = Double1d([1,2,3])
print IS_ANY_NAN(x)              # False (0)
x = Double1d([1,Double.NaN,3])
print IS_ANY_NAN(x)              # True (1)
```

API Summary

Jython Syntax

```
&lt;y&gt;=IS_ANY_NAN(&lt;x&gt;)
```

Properties

a float or double array **x** [INPUT, MANDATORY, default=no default value]

a boolean value **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

a float or double array x [INPUT, MANDATORY, default=no default value]

The input array

a boolean value y [OUTPUT, MANDATORY, default=no default value]

True if the given array contains a NaN value, False otherwise.

See also

- Developers Manual: herschel.ia.numeric.toolbox.basic.IsAnyNaN

1.231. IS_FINITE

Full Name:	herschel.ia.numeric.toolbox.basic.IsFinite
Alias:	IS_FINITE
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import IsFinite
Category	Arrays and datasets/Element selection

Description

Checks whether a number or the elements of an array are finite.

If the input is an array, the output is a boolean array of the same size, with true values if the corresponding array values are finite, false otherwise.

Example

Example 1: Applying IS_FINITE to a Double1d

```
x = Double1d([1, Double.NaN, Double.NEGATIVE_INFINITY,
 Double.POSITIVE_INFINITY])
print IS_FINITE(x) # [true,false,false,false]
```

API Summary

Jython Syntax

```
&lt;y&gt; = IS_FINITE(&lt;x&gt;)
```

Properties

Number or array **x** [INPUT, MANDATORY, default=no default value]

Boolean number or array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Number or array x [INPUT, MANDATORY, default=no default value]

The number or array to check.

Boolean number or array y [OUTPUT, MANDATORY, default=no default value]

True if the corresponding number or array element is finite, false otherwise.

See also

- IS_NAN
- IS_INFINITE
- Developers Manual: herschel.ia.numeric.toolbox.basic.IsFinite

1.232. IS_INFINITE

Full Name:	herschel.ia.numeric.toolbox.basic.IsInfinite
Alias:	IS_INFINITE
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import IsInfinite
Category	Arrays and datasets/Element selection

Description

Checks whether a number or the elements of an array are infinite.

If the input is an array, the output is a boolean array of the same size, with true values if the corresponding array values are infinite, false otherwise.

Example

Example 1: Applying IS_INFINITE to a Double1d

```
x = Double1d([1, Double.NaN, Double.NEGATIVE_INFINITY,
 Double.POSITIVE_INFINITY])
print IS_INFINITE(x) # [false,false,true,true]
```

API Summary

Jython Syntax

```
&lt;y&gt; = IS_INFINITE(&lt;x&gt;)
```

Properties

Number or array **x** [INPUT, MANDATORY, default=no default value]

Boolean number or array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Number or array x [INPUT, MANDATORY, default=no default value]

The number or array to check.

Boolean number or array y [OUTPUT, MANDATORY, default=no default value]

True if the corresponding number or array element is infinite, false otherwise.

See also

- IS_NAN
- IS_FINITE
- Developers Manual: herschel.ia.numeric.toolbox.basic.IsInfinite

1.233. IS_NAN

Full Name:	herschel.ia.numeric.toolbox.basic.IsNaN
Alias:	IS_NAN
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import IsNaN
Category	Arrays and datasets/Element selection

Description

Checks whether a number or the elements of an array are NaN (Not a Number).

If the input is an array, the output is a boolean array of the same size, with true values if the corresponding array values are NaN, false otherwise.

Example

Example 1: Applying IS_NAN to a Double1d

```
x = Double1d([1, Double.NaN, Double.NEGATIVE_INFINITY,
 Double.POSITIVE_INFINITY])
print IS_NAN(x) # [false,true,false,false]
```

API Summary

Jython Syntax

```
&lt;y&gt; = IS_NAN(&lt;x&gt;)
```

Properties

Number or array **x** [INPUT, MANDATORY, default=no default value]

Boolean number or array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Number or array x [INPUT, MANDATORY, default=no default value]

The number or array to check.

Boolean number or array y [OUTPUT, MANDATORY, default=no default value]

True if the corresponding number or array element is NaN (Not a Number), false otherwise.

See also

- IS_FINITE
- IS_INFINITE
- Developers Manual: herschel.ia.numeric.toolbox.basic.IsNaN

1.234. JeffreysPrior

Full Name:	herschel.ia.numeric.toolbox.fit.JeffreysPrior
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import JeffreysPrior
Category	Mathematics/Fitting

Description

Jeffreys prior distribution, for scale-like parameters.

Jeffreys prior is a improper prior (i.e. its integral is unbound). Because of that it always needs limits, low and high, such that 0 < low < high < +Inf. `Pr(x) = 1.0 / ( x * norm ) if low < x < high 0.0 otherwise where norm = log( high ) - log( low )` No limits are set by default.

domain2unit: u = (log(d) - log(lo)) / (log(hi) - log(lo));

unit2domain: d = exp(u * (log(hi) - log(lo)) + log(lo));

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.JeffreysPrior

1.235. Kernel2dModel

Full Name:	herschel.ia.numeric.toolbox.fit.kernel.Kernel2dModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit.kernel import Kernel2dModel
Category	Mathematics/Fitting

Description

Two dimensional Kernel Model.

The Kernel2dModel is defined as

$$f(x:p) = p_0 * K(r)$$

where K(r) is a selectable kernel function and r is the distance to the center.

$r = sqrt(u^2 + v^2)$.

There are 3 options for u and v:

Table 1.2. Kernel table.

option	name	u	v	nr params
1	**CIRCULAR**	**(x - p_1) / p_3**	**(y - p_2) / p_3**	**4**
2	**ELLIPTIC**	**(x - p_1) / p_3**	**(y - p_2) / p_4**	**5**
3	**ROTATED**	**((x - p_1)*cos(p_5) - (y - p_2)*sin(p_5)) / p_3**	**((x - p_1)*sin(p_5) + (y - p_2)*cos(p_5)) / p_4**	**6**

The "center" parameters (1&2) and the "angle" parameter (5) are initialized as 0. The rotational angle is measured counterclockwise from the x-axis. The "width" parameters (3&4) are initialized as 1.0, except for the ROTATED case; then they need to be different (2.0 and 0.5 resp.). Otherwise the model parameter "angle" is degenerate. The "amplitude" parameter is set to 1.0.

Several kernel functions, K(x) are defined in {@link Kernel}.

Beware: These models are unaware of anything outside their range.

Example

Example 1: Kernel2dModel

```
model = Kernel2dModel( )
model.setKernel( Kernel.LORENTZ )                          # circular Lorentz
 model.
model = Kernel2dModel( Kernel2dModel.ROTATED )
model.setKernel( Kernel.GAUSS )                            # equivalent to
 Gauss2DRotModel.
```

See also

- Fitter
- NestedSampler
- Developers Manual: herschel.ia.numeric.toolbox.fit.kernel.Kernel2dModel

1.236. Kernel

Full Name:	herschel.ia.numeric.toolbox.fit.kernel.Kernel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit.kernel import Kernel
Category	Mathematics/Fitting

Description

A Kernel is a non-negative real-valued integrable function.

It is satisfying the following two requirements:

- The integral over [-Inf,+Inf] exists (< Inf).
- It is an even function: K(x) = K(-x).

A kernel is called bound when it is 0 everywhere except when |x| < 1.0.

Several kernel functions, K(x) are defined in this package:

Table 1.3. Kernel table.

type	name	definition	amplitude	bound
1	UNIFORM	1	0.5	yes
2	TRIANGLE	1 - |x|	1	yes
3	PARABOLIC	1-x^2	3/4	yes
4	BIWEIGHT	(1-x^2)^2	15/16	yes
5	TRIWEIGHT	(1-x^2)^3	35/32	yes
6	TRICUBE	(1-|x|^3)^3	70/81	yes
7	COSINE	cos(0.5*PI*x)	PI/4	yes
8	GAUSS	exp(0.5*x*x)	PI/2	no
9	LORENTZ	1 / (1 + x*x)	PI	no
10	SINC	sin(x) / x	PI	no

See also

- KernelModel
- Kernel2dModel
- Developers Manual: herschel.ia.numeric.toolbox.fit.kernel.Kernel

1.237. KernelModel

Full Name:	herschel.ia.numeric.toolbox.fit.kernel.KernelModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit.kernel import KernelModel
Category	Mathematics/Fitting

Description

Kernel Model, a Model build around an Kernel.

The KernelModel is defined as

f(x:p) = p_0 * K((x - p_1) / p_2)

where K(u) is a selectable kernel function on the rescaled input u; u = (x - p_1) / p_2.

p_0 is the amplitude, p_1 is the center and p_2 is the range.

The parameters are initialized at {amp,0,1}. the amplitude is such that the function integrates to 1.0. They are listed in the table.

Several kernel functions predefined. See

Example

Example 1: KernelModel

```
model = KernelModel( )
model.setKernel( Kernel.BIWEIGHT )
```

See also

- Kernel
- Fitter
- NestedSampler
- Developers Manual: herschel.ia.numeric.toolbox.fit.kernel.KernelModel

1.238. KernelModel

Full Name:	herschel.ia.numeric.toolbox.fit.KernelModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import KernelModel
Category	Mathematics/Fitting

Description

Kernel Model, a Model build around an Kernel.

The KernelModel is defined as

f(x:p) = p_0 * K((x - p_1) / p_2)

where K(u) is a selectable kernel function on the rescaled input u; u = (x - p_1) / p_2.

p_0 is the amplitude, p_1 is the center and p_2 is the range.

The parameters are initialized at {amp,0,1}. the amplitude is such that the function integrates to 1.0. They are listed in the table.

Several kernel functions predefined. See

Example

Example 1: KernelModel

```
model = KernelModel( )
model.setKernel( Kernel.BIWEIGHT )
```

See also

- Kernel
- Fitter
- NestedSampler
- Developers Manual: herschel.ia.numeric.toolbox.fit.KernelModel

1.239. KURTOSIS

Full Name:	herschel.ia.numeric.toolbox.basic.Kurtosis
Alias:	KURTOSIS
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Kurtosis
Category	Mathematics/Statistics

Description

Returns the kurtosis excess of an array.

Kurtosis excess measures the degree of peakedness of a distribution, and is defined as the fourth central moment divided by the standard deviation raised to the fourth power, all minus three. A normal distribution has a kurtosis of three and a kurtosis excess of zero.

For more information on kurtosis and kurtosis excess see this website.

Returns NaN (Not a Number) if any of the array elements is NaN.

Example

Example 1: Applying KURTOSIS to a Float1d

```
x = Float1d([1,3,2,3,4])
print KURTOSIS(x) # -1.7484023668639055
x = Float1d([1,Double.NaN,2,3,4])
print KURTOSIS(x) # NaN
# To remove NaN values (the original array is not modified):
print KURTOSIS(NAN_FILTER(x)) # -2.0775 (KURTOSIS is applied to [1,2,3,4])
```

API Summary

Jython Syntax

```
&lt;y&gt; = KURTOSIS(&lt;x&gt;)
```

Properties

```
Array x [INPUT, MANDATORY, default=no default value]
Double y [OUTPUT, MANDATORY, default=no default value]
```

API details

Properties

Array x [INPUT, MANDATORY, default=no default value]

The array of which to compute the kurtosis excess. Complex arrays are not allowed.

Double y [OUTPUT, MANDATORY, default=no default value]

The kurtosis excess of the input array.

See also

- MEAN
- MEDIAN
- SKEWNESS
- STDDEV
- VARIANCE
- Developers Manual: herschel.ia.numeric.toolbox.basic.Kurtosis

1.240. LaplaceErrorDistribution

Full Name:	herschel.ia.numeric.toolbox.fit.sample.LaplaceErrorDistribution
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit.sample import LaplaceErrorDistribution
Category	Mathematics/Fitting

Description

LaplaceErrorDistribution.

Laplace distr : f(x) = 1 / (2 s) exp(- |x| / s)

where s is the scale and |x| is the absolute value of the residuals.

Using this error distribution results in median-like solutions.

See example

For a simple way to choose the distribution in the NestedSampler use NestedSampler.setLaplaceDistribution().

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.sample.LaplaceErrorDistribution

1.241. LaplacePrior

Full Name:	herschel.ia.numeric.toolbox.fit.LaplacePrior
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import LaplacePrior
Category	Mathematics/Fitting

Description

Laplace prior distribution.

Pr(x) = exp(- | x - offset | / scale)

By default: scale = 1 and offset = 0. It can also have a limited domain. By default the domain is [-Inf,+Inf].

Equivalent to a double-sided exponential prior

domain2unit: u = (d < 0) ? 0.5 * exp(d / scale) : 1.0 - 0.5 * exp(-d / scale);

unit2domain: d = (u < 0.5) ? log(2 * u) * scale : -log(2 * (1 - u)) * scale;

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.LaplacePrior

1.242. LevenbergMarquardtFitter

Full Name:	herschel.ia.numeric.toolbox.fit.LevenbergMarquardtFitter
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import LevenbergMarquardtFitter
Category	Mathematics/Fitting

Description

Non-linear fitter using the Levenberg-Marquardt method.

Implementation of the Levenberg-Marquardt algorithm to fit the parameters of a non-linear model. It is a gradient fitter which uses partial derivatives to find the downhill gradient. Consequently it ends in the first minimum it finds.

An introduction to the use of fit package can be found here.

At least once have a look at the background on the organization and structure of the package.

The fit/demo directory contains worked examples on almost all aspects of of the package.

Example

Example 1: LevenbergMarquardtFitter (for non-linear models)

```
# assume x and y are Double1d data arrays:
x = Double1d.range(100) / 10
y = Double1d.range(100) / 122            # make slope
rg = RandomGauss( seed=12345L )            # Gaussian random number generator
y += rg( Double1d(100) ) * 0.2             # add noise
y[Range(9,12)] += Double1d([5,10,7])         # make some peak
# define a model: GaussModel + background polynomial
gauss = GaussModel( )                              # Gaussian
gauss += PolynomialModel( 1 )                      # add linear background
gauss.setParameters( Double1d([1,1,0.1,0,0]) )    # initial parameter guess
print gauss.getNumberOfParameters()                  # 5 (= 3 for Gauss + 2 for
 line)
gauss.keepFixed( Int1d([2]), Double1d([0.1]))    # keep width fixed at 0.1
lmfit = LevenbergMarquardtFitter( x, gauss )
param = lmfit.fit( y )
print param.length()                               # 4 (= 5 - 1 fixed)
stdev = lmfit.getStandardDeviation()               # stdevs on the parameters
chisq = lmfit.getChiSquared()
scale = lmfit.getScale()                           # noise scale
yfit  = lmfit.getResult()                          # fitted values
yband = lmfit.monteCarloError()                      # 1 sigma confidence region
# for diagnostics (or just for fun)
lmfit = LevenbergMarquardtFitter( x, gauss )
lmfit.setVerbose( 10 )                              # report every 10th iteration
plotter = IterationPlotter()                       # from herschel.ia.toolbox.fit
lmfit.setPlotter( plotter, 20 )                      # make a plot every 20th
 iteration
param = lmfit.fit( y )
```

Limitations

1. LMF is **not** guaranteed to find the global minimum.

2. The calculation of the evidence is an Gaussian approximation which is only exact for linear models with a fixed scale.

Miscellaneous

In case of problems look at the trouble shooting section.

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.LevenbergMarquardtFitter

1.243. LinearInterpolator

Full Name:	herschel.ia.numeric.toolbox.interp.LinearInterpolator
Alias:	LinearInterpolator
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.interp import LinearInterpolator
Category	Mathematics/Interpolation

Description

This class linearly interpolates unknown values based on the two closest known knots (x,y data).

Given a set of knots (x/y data), this function computes values at arbitrary positions by linearly interpolating the y value based on the knot lower and higher than the input x position. This can only be applied to numeric arrays of rank 1.

Examples

Example 1: Create and apply a LinearInterpolator on a Double1d

```
# create some knots
x=Double1d.range(10) # [0.0,1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0]
f=LinearInterpolator(x,SQUARE(x))
u=Double1d([1,1.5,2,2.5,3,3.5])
print f(u) # [1.0,2.5,4.0,6.5,9.0,12.5]
```

Example 2: Create and apply a LinearInterpolator on a Double1d and interpolation method TruncatedLinear

```
x = Double1d.range(5)
y = x
u=Double1d([-1.0,1.1,5.5])
f=LinearInterpolator(x,y,"TruncatedLinear")
print f(u) #[0.0,1.1,4.0]
```

API Summary

Jython Syntax

```
&lt;f&gt;=LinearInterpolator(&lt;x&gt;,&lt;y&gt;[,allowExtrapolation])
```

Properties

Double1d **x** [INPUT, MANDATORY, default=no default value]
Double1d **y** [INPUT, OPTIONAL, default=false]
boolean **allowExtrapolation** [INPUT, OPTIONAL, default=false]

API details

Properties

Double1d x [INPUT, MANDATORY, default=no default value]

The knots are Double1d

Double1d y [INPUT, OPTIONAL, default=false]
The knots are Double1d

boolean allowExtrapolation [INPUT, OPTIONAL, default=false]
Extrapolation is not allowed by default. Boolean.TRUE allows extrapolation.

See also

- CubicSplineInterpolator
- NearestNeighborInterpolator
- Developers Manual: herschel.ia.numeric.toolbox.interp.LinearInterpolator

1.244. ListContext

Full Name:	herschel.ia.pal.ListContext
Type:	Java Class -
Import:	from herschel.ia.pal import ListContext
Category	Data access

Description

Groups products (or other Contexts that in turn group products) in a list-like structure.

Products grouped in a ListContext can be subsequently retrieved by an index (with index=0 corresponding to the first product in the list), through the 'refs' method. Note: users may be confused as to why there is a need to have to go though a 'refs' method to add or access products from a ListContext. This is due to a technical limitation in the design which will be addressed in due course.

Examples

Example 1: Adding a product to a ListContext

```
product = Product()
ref = storage.save(product) # the ref is a ProductRef object
listcontext = ListContext()
listcontext.refs.add(ref)
```

Example 2: Getting the first product from a ListContext

```
ref_first = listcontext.refs.get(0)
product = ref_first.product
```

Example 3: Saving a ListContext to ProductStorage (same way as any other product)

```
ref_context = storage.save(listcontext)
```

API Summary

Method
List **getRefs** get the list of ProductRefs stored. From this list, you can put

API Details

Method

List **getRefs**

get the list of ProductRefs stored. From this list, you can put

products into the ListContext, or retrieve products by index.

Return

List

A list of product refs.

List **getRefs**

Examples

Putting a product into the the ListContext

```
ref = storage.save(product) # the ref is a ProductRef object
listcontext.refs.add(ref)
```

Getting the first product from the ListContext

```
ref_first = listcontext.refs.get(0)
```

See also

- Developers Manual: herschel.ia.pal.ListContext

1.245. localStoreCopier

Full Name:	herschel.ia.toolbox.util.LocalStoreCopierTask
Alias:	localStoreCopier
Type:	Java Task -
Import:	from herschel.ia.toolbox.util import LocalStoreCopierTask
Category	Data access

Description

Creates a copy of a LocalStore (pool).

Creates an optionally compressed copy of a Local Store. This copy can be distributed to other HIPE installations and imported easily.

Example

Example 1: Copying a local store into a zip file

```
localStoreCopier(poolName = "mib", filename = "mycopy.zip", compression =
 "ZIP")
```

API Summary

Jython Syntax

```
localStoreCopier(&lt;poolName&gt; [, &lt;warn&gt;=True] ,
&lt;filename&gt; [, &lt;compression&gt;="ZIP"])
```

Properties

String **poolName** [INPUT, MANDATORY, default=no default value]
Boolean **warn** [INPUT, OPTIONAL, default=True]
String **filename** [INPUT, MANDATORY, default=no default value]
String **compression** [INPUT, OPTIONAL, default="ZIP"]

API details

Properties

String poolName [INPUT, MANDATORY, default=no default value]

LocalStore id of the pool to copy.

Boolean warn [INPUT, OPTIONAL, default=True]

If true, warn if overwriting

String filename [INPUT, MANDATORY, default=no default value]

Path of the copy

String compression [INPUT, OPTIONAL, default="ZIP"]

The type of compression in the output file.

String compression [INPUT, OPTIONAL, default="ZIP"]
Valid values ares: • **"NONE"** : (default) do not compress the copy • **"ZIP"** : make a zip compressed copy

See also

- Developers Manual: herschel.ia.toolbox.util.LocalStoreCopierTask

History

- 2010-04-28 - JDS: first release
- 2013-03-06 - JDS: renaming parameters and warn default to True

1.246. LOG10

Full Name:	herschel.ia.numeric.toolbox.basic.Log10
Alias:	LOG10
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Log10
Category	Mathematics/General functions

Description

Returns the base 10 logarithm of a number or array.

If the input is an array, the output is an array of the same size, with the computed logarithms instead of the original elements.

Example

Example 1: Applying LOG10 to a Double1d

```
x = Double1d([1,10,100])
print LOG10(x) # [0.0,1.0,2.0]
```

API Summary

Jython Syntax

```
&lt;y&gt; = LOG10(&lt;x&gt;)
```

Properties

Number or array **x** [INPUT, MANDATORY, default=no default value]

Number or array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Number or array x [INPUT, MANDATORY, default=no default value]

The value or values of which to compute the logarithm. Complex numbers are not allowed.

Number or array y [OUTPUT, MANDATORY, default=no default value]

The logarithm value or values, or NaN if the input is negative.

See also

- EXP10
- LOG
- LogN
- Developers Manual: herschel.ia.numeric.toolbox.basic.Log10

1.247. LogFactorial

Full Name:	herschel.ia.numeric.toolbox.random.LogFactorial
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.random import LogFactorial
Category	Mathematics/General functions

Description

To get a log factorial of a value.

Example

Example 1: To get a log factorial of k=10

```
f = LogFactorial.get(10)
```

API Summary

Method
double **get** (int k) get

API Details

Method

***double* get (int k)**

get

Return the (natural) log(k!).

Argument

`int k [INPUT, MANDATORY, default=no default value]`

The number the factorial is wanted for.

Return

double

The (natural) log(k!)

See also

- Developers Manual: herschel.ia.numeric.toolbox.random.LogFactorial

1.248. LOG

Full Name:	herschel.ia.numeric.toolbox.basic.Log
Alias:	LOG
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Log
Category	Mathematics/General functions

Description

Returns the natural logarithm of a number or array.

If the input is an array, the output is an array of the same size, with the computed logarithms instead of the original elements.

Example

Example 1: Applying LOG to a Double1d

```
x = Double1d([3,5])
print LOG(x) #  [1.0986122886681098,1.6094379124341003]
```

API Summary

Jython Syntax
<y> = LOG(<x>)

Properties
Number or array **x** [INPUT, MANDATORY, default=no default value]
Number or array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Number or array x [INPUT, MANDATORY, default=no default value]

The value or values of which to compute the logarithm. Complex numbers are not allowed.

Number or array y [OUTPUT, MANDATORY, default=no default value]

The logarithm value or values, or NaN if the input is negative.

See also

- EXP
- LOG10
- LogN
- Developers Manual: herschel.ia.numeric.toolbox.basic.Log

1.249. LogN

Full Name:	herschel.ia.numeric.toolbox.basic.LogN
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import LogN
Category	Mathematics/General functions

Description

Returns the base *n* logarithm of a number or array.

If the input is an array, the output is an array of the same size, with the computed logarithms instead of the original elements.

Example

Example 1: Applying LogN to an Int1d

```
x = 2**Int1d.range(6) # [1,2,4,8,16,32]
print LogN(2)(x)      # [0.0,1.0,2.0,3.0,4.0,5.0]
```

API Summary

Jython Syntax
<y> = LogN(<base>)(<x>)

Properties
Number or array **x** [INPUT, MANDATORY, default=no default value]
Number **base** [INPUT, MANDATORY, default=no default value]
Number or array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Number or array x [INPUT, MANDATORY, default=no default value]

The value or values of which to compute the logarithm. Complex numbers are not allowed.

Number base [INPUT, MANDATORY, default=no default value]

The base of the logarithm.

Number or array y [OUTPUT, MANDATORY, default=no default value]

The logarithm value or values, or NaN if the input is negative.

See also

- ExpN
- LOG

- LOG10
- Developers Manual: herschel.ia.numeric.toolbox.basic.LogN

1.250. Long1d

Full Name:	herschel.ia.numeric.Long1d
Type:	Java Class -
Import:	from herschel.ia.numeric import Long1d
Category	Arrays and datasets

Description

A rectangular numeric long array of rank 1.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Long1d

1.251. Long2d

Full Name:	herschel.ia.numeric.Long2d
Type:	Java Class -
Import:	from herschel.ia.numeric import Long2d
Category	Arrays and datasets

Description

A rectangular numeric long array of rank 2.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Long2d

1.252. Long3d

Full Name:	herschel.ia.numeric.Long3d
Type:	Java Class -
Import:	from herschel.ia.numeric import Long3d
Category	Arrays and datasets

Description

A rectangular numeric long array of rank 3.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Long3d

1.253. Long4d

Full Name:	herschel.ia.numeric.Long4d
Type:	Java Class -
Import:	from herschel.ia.numeric import Long4d
Category	Arrays and datasets

Description

A rectangular numeric long array of rank 4.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Long4d

1.254. Long5d

Full Name:	herschel.ia.numeric.Long5d
Type:	Java Class -
Import:	from herschel.ia.numeric import Long5d
Category	Arrays and datasets

Description

A rectangular numeric long array of rank 5.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Long5d

1.255. LorentzModel

Full Name:	herschel.ia.numeric.toolbox.fit.LorentzModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import LorentzModel
Category	Mathematics/Fitting

Description

Lorentzian Model.

$$f(x:p) = p_0 * (p_2^2 / ((x - p_1)^2 + p_2^2)$$

p_0 = amplitude p_1 = x-shift p_2 = gamma (width)

The parameters are initialized at {1/PI, 0.0, 1.0} where the integral over the function equals 1. Parameter 2 (gamma) is always kept positive (>=0).

This model is also known as Cauchy or Cauchy-Lorentz.

See example (change GaussModel into LorentzModel)

Example

Example 1: LorentzModel

```
lorentz = LorentzModel()
lorentz.setParameters( Double1d( [5, 4, 1] ) )
print lorentz( Double1d.range( 41 ) / 5 )    # from [0,8]
# ... fitter etc. see LevenbergMarquardtFitter
```

Miscellaneous

There are other possible definitions of this model, where the integral equals 1.0. Definitions that integrate to 1.0 are more fit as a distribution function. See sample/CauchyErrorDistribution.

We choose our definition for 2 reasons.

1. to be in line with the definitions of the GaussModel, SincModel, VoigtModel, all KernelModels etc. In all of them the amplitude parameter, p_0, equals the maximum of the function. I.e. p_0 is indeed the amplitude.

2. to have maximally independent parameters, meaning that if you change one parameter, only that aspect changes. In the present definition this is the case. In the alternative definition if you change p_2, not only the width of the function changes, but also the amplitude.

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.LorentzModel

1.256. manualContour

Full Name:	herschel.ia.toolbox.image.ManualContourTask
Alias:	manualContour
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import ManualContourTask
Category	Images/Display

Description

This is a task that generates the contours for a given list of contour values.

This is a task that generates the contours for a given list of contour values. The output product is an ImageContour product, which holds the generated contours. This product can be dragged over an image explorer for visual inspection. The user can also specify in this task which colors should be used for visualization.

Example

Example 1: This is an example of how you can use manualContour :

```
from java.awt.Color import GREEN
from java.awt.Color import RED
contours = manualContour(image = myImage, values = Double1d([3.7,4.2]), colors
 = [GREEN, RED])
contours = manualContour(image = myImage, values = Double1d([3.7,4.2]))
```

API Summary

Properties
Image **image** [INPUT, MANDATORY, default=None]
Double1d **values** [INPUT, MANDATORY, default=None]
Color[] **colors** [INPUT, OPTIONAL, default=None (BLUE is used then)]
ImageContour **contours** [OUTPUT, MANDATORY, default=None]

API details

Properties

Image image [INPUT, MANDATORY, default=None]

This is the input image.

Double1d values [INPUT, MANDATORY, default=None]

These are the contour values.

Color[] colors [INPUT, OPTIONAL, default=None (BLUE is used then)]

These are the colors to visualize the contours.

ImageContour contours [OUTPUT, MANDATORY, default=None]
These are the resulting contours

See also

- Developers Manual: herschel.ia.toolbox.image.ManualContourTask

1.257. MapContext

Full Name:	herschel.ia.pal.MapContext
Type:	Java Class -
Import:	from herschel.ia.pal import MapContext
Category	Data access

Description

Groups products (or other Contexts that in turn group products) in a map-like structure.

Products grouped in a MapContext can be subsequently retrieved by key, through the 'refs' method. Note: users may be confused as to why there is a need to have to go though a 'refs' method to add or access products from a MapContext. This is due to a technical limitation in the design which will be addressed in due course.

Examples

Example 1: Adding a product to a MapContext

```
product = Product()
ref = storage.save(product) # the ref is a ProductRef object
mapcontext = MapContext()
mapcontext.refs.put("john", ref)
```

Example 2: Getting a product from a MapContext

```
ref_john = mapcontext.refs.get("john")
product = ref_john.product
```

Example 3: Saving a MapContext to ProductStorage (same way as any other product)

```
ref_context = storage.save(mapcontext)
```

API Summary

Method
Map **getRefs** get the 'map' of ProductRefs stored. From this 'map', you can put

API Details

Method

Map **getRefs**

get the 'map' of ProductRefs stored. From this 'map', you can put

products into the MapContext, or retrieve products by key.

Return

Map

A map of product refs.

Map getRefs

Examples

Putting a product into the the MapContext

```
ref = storage.save(product) # the ref is a ProductRef object
mapcontext.refs.put("john", ref)
```

Getting the product from the MapContext with key "john"

```
ref_john = mapcontext.refs.get("john")
```

See also

- Developers Manual: herschel.ia.pal.MapContext

1.258. MATRIXMULTIPLY

Full Name:	herschel.ia.numeric.toolbox.matrix.MatrixMultiply
Alias:	MATRIXMULTIPLY
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.matrix import MatrixMultiply
Category	Mathematics/Matrices

Description

Performs matrix multiplication of numeric or logical matrices.

Examples

Example 1: Different syntax forms

```
# spelled out:
c=MatrixMultiply(b)(a)
c=a.apply(MatrixMultiply(b))
#
# reusing an instance of a matrix-multiplier:
f=MatrixMultiply(b)
c=f(a)
c=a.apply(f)
```

Example 2: Matrix multiplication examples

```
A = Int2d([[1,2,3], [2,3,4]])
B = Int2d([[1,2], [2,3], [3,4]])
X = Int1d([1,2])
Y = Int1d([1,2,3])
#
print A.apply(MatrixMultiply(B))
[[14, 20], [20, 29]]
print X.apply(MatrixMultiply(A))
[5, 8, 11]
print A.apply(MatrixMultiply(Y))
[14, 20]
```

API Summary

Jython Syntax

```
&lt;y&gt;=MatrixMultiply(&lt;b&gt;)(&lt;a&gt;)
```

Properties

Array1dData or Array2dData **a** [INPUT, MANDATORY, default=no default value]

Array1dData or Array2dData **b** [INPUT, MANDATORY, default=no default value]

Miscellaneous

Complex arrays are not supported yet.

API details

Properties

Array1dData or Array2dData a [INPUT, MANDATORY, default=no default value]

Input must be a rank one or rank two array of any type, but String or Complex.

Array1dData or Array2dData b [INPUT, MANDATORY, default=no default value]

Input must be an array of the same type as 'a'. If 'a' has rank one, 'b' must have rank two. If 'b' has rank one, 'a' must have rank two.

See also

- Developers Manual: herschel.ia.numeric.toolbox.matrix.MatrixMultiply

1.259. MATRIXSOLVE

Full Name:	herschel.ia.numeric.toolbox.matrix.MatrixSolve
Alias:	MATRIXSOLVE
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.matrix import MatrixSolve
Category	Mathematics/Matrices

Description

Solves systems of linear equations of type A x = b.

A system of equations is a collection of equations that you deal with all together at once.

Example

Example 1: Apply MatrixSolve

```
# Solve
#  | 1  2 | |x1|   | 8|
#  |      | |  | = |  |
#  | 3  4 | |x2|   |18|
#
A=Float2d([ [1,2],[3,4] ])
b=Double1d([8.0,18.0])
print MatrixSolve(b)(A) # [2.0,3.0]
print A.apply(MatrixSolve(b))
```

API Summary

Jython Syntax

```
&lt;x&gt;=MatrixSolve(&lt;b&gt;)(&lt;A&gt;)
```

Properties

any matrix **A** [INPUT, MANDATORY, default=no default value]

Double1d **b** [INPUT, MANDATORY, default=no default value]

Miscellaneous

Does not work for complex matrices.

API details

Properties

any matrix A [INPUT, MANDATORY, default=no default value]

Any matrix

Double1d b [INPUT, MANDATORY, default=no default value]

Input must be a Double1d array.

See also

- Developers Manual: herschel.ia.numeric.toolbox.matrix.MatrixSolve

1.260. MAX

Full Name:	herschel.ia.numeric.toolbox.basic.Max
Alias:	MAX
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Max
Category	Arrays and datasets/Reduction

Description

Returns the value of the largest element in the input array.

For multidimensional arrays, this function can return the largest array values along a given dimension. For example, you can obtain the largest values of each column, or each row, of a two-dimensional array. See also the example below. You cannot use this function with arrays of complex numbers.

Example

Example 1: Applying MAX to an Int2d

```
x = Int2d( [ [1,2], [-1,3] ])
print MAX(x)    # 3
print MAX(x, 0) # [1,3] Max of [1,-1] and [2,3]
print MAX(x, 1) # [2,3] Max of [1,-1] and [2,3]
```

API Summary

Jython Syntax

```
&lt;y&gt; = MAX(&lt;x&gt; [,&lt;dim&gt;])
```

Properties

Array **x** [INPUT, MANDATORY, default=no default value]

Integer **dim** [INPUT, OPTIONAL, default=no default value]

Number or array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Array x [INPUT, MANDATORY, default=no default value]

The array of which to find the largest value.

Integer dim [INPUT, OPTIONAL, default=no default value]

The dimension along which to compute the calculation.

Number or array y [OUTPUT, MANDATORY, default=no default value]

The value of the largest element of the input array, or the largest values along a dimension if the dim parameter is specified.

See also

- MIN
- Developers Manual: herschel.ia.numeric.toolbox.basic.Max

1.261. MEAN

Full Name:	herschel.ia.numeric.toolbox.basic.Mean
Alias:	MEAN
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Mean
Category	Mathematics/Statistics

Description

Returns the mean of an array.

Returns NaN (Not a Number) if any of the array elements is NaN.

Example

Example 1: Applying MEAN to a Float1d

```
x = Float1d([1,3,2,3,4])
print MEAN(x) # 2.6
x = Float1d([1,Double.NaN,2,3,4])
print MEAN(x) # NaN
# To remove NaN values (the original array is not modified):
print MEAN(NAN_FILTER(x)) # 2.5 (MEAN is applied to [1,2,3,4])
```

API Summary

Jython Syntax
<y> = MEAN(<x>)

Properties
Array **x** [INPUT, MANDATORY, default=no default value]
Double **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Array x [INPUT, MANDATORY, default=no default value]

The array of which to compute the mean. Complex arrays are not allowed.

Double y [OUTPUT, MANDATORY, default=no default value]

The mean of the input array.

See also

- MEDIAN
- STDDEV
- VARIANCE

- SKEWNESS
- KURTOSIS
- Developers Manual: herschel.ia.numeric.toolbox.basic.Mean

1.262. meanSmoothing

Full Name:	herschel.ia.toolbox.image.MeanSmoothingTask
Alias:	meanSmoothing
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import MeanSmoothingTask
Category	Images/Analysis

Description

This is a task to smooth/filter an image by applying a mean filter.

This is a task to smooth/filter an image using the average (mean) filter, for which the user must specify the width. The average filter computes the sum of all pixel values in the filter window and then divides the sum by the number of pixels in the filter window.

Example

Example 1: This is an example of how to use the meanSmoothing :

```
smoothed = meanSmoothing(image = myImage, width = 3)
```

API Summary

Properties
Image **image** [INPUT, MANDATORY, default=None]
Integer **width** [INPUT, MANDATORY, default=3]
Image **smoothed** [OUTPUT, OPTIONAL, default=None]

API details

Properties

Image image [INPUT, MANDATORY, default=None]

This is the input image.

Integer width [INPUT, MANDATORY, default=3]

This is the width of the filter window in pixels. This must be an odd, strictly positive integer.

Image smoothed [OUTPUT, OPTIONAL, default=None]

This is the resultign smoothed/filtered image.

See also

- Developers Manual: herschel.ia.toolbox.image.MeanSmoothingTask

1.263. MedianAbsoluteDeviation

Full Name:	herschel.ia.numeric.toolbox.basic.MedianAbsoluteDeviation
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import MedianAbsoluteDeviation
Category	Mathematics/Statistics

Description

Returns the median absolute deviation of an array.

The median absolute deviation is one of several ways to estimate the error of the median.

For an array **data** the algorithm calculates the new array | data[i] - median(data) | for all values, where | indicates the absolute value. The median absolute deviation is the median of this new array.

Example

Example 1: Apply MedianAbsoluteDeviation to an Int1d array

```
data = Int1d.range(9)
median = MEDIAN(data)
dev = data.apply( MedianAbsoluteDeviation(median) )
```

API Summary

Jython Syntax

```
median = MEDIAN(&lt;x&gt;)
&lt;y&gt; = &lt;x&gt;.apply( MedianAbsoluteDeviation(median) )
```

Properties

Array **x** [INPUT, MANDATORY, default=no default value]
Double **median** [INPUT, MANDATORY, default=no default value]
Double **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Array x [INPUT, MANDATORY, default=no default value]

The array of which to compute the median absolute deviation.

Double median [INPUT, MANDATORY, default=no default value]

The median of the input array.

Double y [OUTPUT, MANDATORY, default=no default value]

The median absolute deviation of the input array with respect to the provided median.

See also

- Developers Manual: herschel.ia.numeric.toolbox.basic.MedianAbsoluteDeviation

1.264. MEDIAN

Full Name:	herschel.ia.numeric.toolbox.basic.Median
Alias:	MEDIAN
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Median
Category	Mathematics/Statistics

Description

Returns the median of an array.

Returns NaN (Not a Number) if any of the array elements is NaN.

Example

Example 1: Apply MEDIAN to a Float1d

```
x = Float1d([1,3,2,3,4])
print MEDIAN(x) # 3.0
x = Float1d([1,Double.NaN,2,3,4])
print MEDIAN(x) # NaN
# To remove NaN values (the original array is not modified):
print MEDIAN(NAN_FILTER(x)) # 2.5 (MEDIAN is applied to [1,2,3,4])
```

API Summary

Jython Syntax

```
&lt;y&gt; = MEDIAN(&lt;x&gt;)
```

Properties

```
Array x [INPUT, MANDATORY, default=no default value]
Double y [OUTPUT, MANDATORY, default=no default value]
```

API details

Properties

`Array x [INPUT, MANDATORY, default=no default value]`

The array of which to compute the median. Complex arrays are not allowed.

`Double y [OUTPUT, MANDATORY, default=no default value]`

The median of the input array.

See also

- MEAN
- STDDEV
- VARIANCE

- SKEWNESS
- KURTOSIS
- Developers Manual: herschel.ia.numeric.toolbox.basic.Median

1.265. medianSmoothing

Full Name:	herschel.ia.toolbox.image.MedianSmoothingTask
Alias:	medianSmoothing
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import MedianSmoothingTask
Category	Images/Analysis

Description

This is a task to smooth/filter an image by applying a median filter.

This is a task to smooth/filter an image using the median filter, for which the user must specify the width. The median filter computes the median of all pixel values in the filter window.

Example

Example 1: This is an example of how to use the medianSmoothing :

```
smoothed = medianSmoothing(image = myImage, width = 3)
```

API Summary

Properties
Image **image** [INPUT, MANDATORY, default=None]
Integer **width** [INPUT, OPTIONAL, default=3]
Image **smoothed** [OUTPUT, MANDATORY, default=None]

API details

Properties

Image image [INPUT, MANDATORY, default=None]

This is the input image.

Integer width [INPUT, OPTIONAL, default=3]

This is the width of the filter window in pixels. This must be an odd, strictly positive integer.

Image smoothed [OUTPUT, MANDATORY, default=None]

This is the resulting smoothed/filtered image.

See also

- Developers Manual: herschel.ia.toolbox.image.MedianSmoothingTask

1.266. MetaQuery

Full Name:	herschel.ia.pal.query.MetaQuery
Type:	Java Class -
Import:	from herschel.ia.pal.query import MetaQuery
Category	Data access

Description

Meta data query formulates a query on the meta data of a Product.

Typically this type of query is slower than an Attribute Query, but faster than a full query on the Product Access Layer.

Example

Example 1: Example of a query on meta data

```
q = MetaQuery(SimpleSpectrum, "p", "p.meta['creator'].value == 'Me'")
```

See also

- Developers Manual: herschel.ia.pal.query.MetaQuery

1.267. MIN

Full Name:	herschel.ia.numeric.toolbox.basic.Min
Alias:	MIN
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Min
Category	Arrays and datasets/Reduction

Description

Returns the value of the smallest element in the input array.

For multidimensional arrays, this function can return the smallest array values along a given dimension. For example, you can obtain the smallest values of each column, or each row, of a two-dimensional array. See also the example below. You cannot use this function with arrays of complex numbers.

Example

Example 1: Applying MIN to an Int2d.

```
x = Int2d( [ [1,2], [-1,3] ])
print MIN(x)    # -1
print MIN(x, 0) # [-1, 2]
print MIN(x, 1) # [ 1,-1]
```

API Summary

Jython Syntax

```
&lt;y&gt; = MIN(&lt;x&gt; [,&lt;dim&gt;])
```

Properties

Array **x** [INPUT, MANDATORY, default=no default value]

Integer **dim** [INPUT, OPTIONAL, default=no default value]

Number or array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Array x [INPUT, MANDATORY, default=no default value]

The array of which to find the smallest value.

Integer dim [INPUT, OPTIONAL, default=no default value]

The dimension along which to compute the calculation.

Number or array y [OUTPUT, MANDATORY, default=no default value]

The value of the smallest element of the input array, or the smallest values along a dimension if the dim parameter is specified.

See also

- MAX
- Developers Manual: herschel.ia.numeric.toolbox.basic.Min

1.268. MinpackFunc

Full Name:	herschel.ia.numeric.toolbox.fit.minpack.MinpackFunc
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit.minpack import MinpackFunc

Description

MinpackFunc

Extend this class to implement your function.

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.minpack.MinpackFunc

1.269. MinpackPro

Full Name:	herschel.ia.numeric.toolbox.fit.minpack.MinpackPro
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit.minpack import MinpackPro

Description

MinpackPro-1 Least Squares Fitting Library

Original public domain version by B. Garbow, K. Hillstrom, J. More (Argonne National Laboratory, Minpack project, March 1980) Tranlation to C Language by S. Moshier (moshier.net) Translated to Java by J. Jacobson (jdj@ipac.caltech.edu) Enhancements and packaging by C. Markwardt (comparable to IDL fitting routine MPFIT see http://cow.physics.wisc.edu/~craigm/idl/idl.html) Note on parameter values: From C> Markwardt's web page "Finally, and this can't be stressed enough, it is crucial to estimate the starting parameters as best you can. Initializing the parameters to all zeroes is actually the worst thing you can do, since then the problem becomes scale-less, and MPFIT has a much harder time deciding what to do. CHANGES: 20100920 Added multiple model handling. (jdj@ipac.caltech.edu) 20110103 Tune hard-coded conf. values to improve chi-square 20110503 Documentation. Note that enhancments to the S. Mosher's c-language version have been borrowed from C. Markwardt. 20110916 Java docs, jtags and re-factorings.

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.minpack.MinpackPro

1.270. MODE

Full Name:	herschel.ia.numeric.toolbox.stat.Mode
Alias:	MODE
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.stat import Mode
Category	Mathematics/Statistics

Description

Yields mode(s), or the most common element(s), in an array of rank 1 to 5.

Example

Example 1: Apply MODE on Double2d, Int3d

```
x=Double2d([ [1,2,2,2,7,8] ])
print MODE(x) #[2.0]
x=Int3d([ [ [1,2],[3,3] ], [ [5,6],[5,8] ], [ [9,10],[11,12] ] ])
print MODE(x) # [3,5]
```

API Summary

Jython Syntax

```
&lt;y&gt;=MODE(&lt;x&gt;)
```

Properties

any array type **x** [INPUT, MANDATORY, default=no default value]

an array of any type **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

any array type x [INPUT, MANDATORY, default=no default value]

The input must be an array of rank 1 to 5

an array of any type y [OUTPUT, MANDATORY, default=no default value]

The result is an array of rank 1 .

See also

- MEAN
- Developers Manual: herschel.ia.numeric.toolbox.stat.Mode

1.271. modelFit

Full Name:	herschel.ia.toolbox.fit.ModelFitTask
Alias:	modelFit
Type:	Java Task -
Import:	from herschel.ia.toolbox.fit import ModelFitTask
Category	Mathematics/Fitting

Description

Task shell around the package herschel.ia.numeric.toolbox.fit.

Both a command line and a GUI interface are provided. With the GUI interface there is more of the Fitter package accessable than on this tasks command line. However not everything possible within the fitter package is accessable via this task, GUI or otherwise. Most text fields in the GUI accept either numerics or HIPE variables or HIPE expressions.

Example

Example 1: ModelFitTask

```
# usage on command line:
tt = Double1d.range(11) - 5
data = Double1d(11)
data[5] = 1.0
data[6] = 4.0
data[7] = 2.0
# one-liner using default GaussModel and LevenbergMarquardtFitter
p1 = modelFit( x=tt, y=data, modelname="GaussModel", fittername =
 "LevenbergMarquardtFitter" )
print p1
# more extended
ft = modelFit
ft.x = tt
ft.y = data
ft.modelname = "LorentzModel"
ft.fittername = "AmoebaFitter"
p2 = ft()            # performs the execute method
print p2            # parameters of the fit
print ft.stdev          # standard deviations
print ft.fittername          # name of the fitter
# Etcetera, etc.
# use the GUI.
ft = modelFit
ft.gui = 1          # start the GUI
# from the GUI select the data, model, fitter, properties etc and run.
# Etcetera as before.
```

API Summary

Jython Syntax
`see example below`

Properties
`NumericData x [INPUT, OPTIONAL, default=null]`
`DoubleArray y [INPUT, MANDATORY, default=no]`
`Boolean indexview [INPUT, OPTIONAL, default=true]`

Properties
DoubleArray **weights** [INPUT, OPTIONAL, default=null]
AbstractModel **model** [INOUT, OPTIONAL, default=null]
String **modelname** [INPUT, OPTIONAL, default=no]
Array1dData **modelarg** [INPUT, OPTIONAL, default=null]
Fitter **fitter** [INOUT, OPTIONAL, default=null]
String **fittername** [INPUT, OPTIONAL, default=no]
Double1d **result** [OUTPUT, OPTIONAL, default=n/a]
Double1d **parameters** [OUTPUT, OPTIONAL, default=n/a]
Double **chisq** [OUTPUT, OPTIONAL, default=n/a]
Double1d **prior** [INPUT, OPTIONAL, default=null]
Double **scale** [OUTPUT, OPTIONAL, default=n/a]
IterationPlotable **plotter** [INPUT, OPTIONAL, default=null]
Integer **plotfreq** [INPUT, OPTIONAL, default=10]
Integer **printfreq** [INPUT, OPTIONAL, default=0]
Boolean **auto** [INPUT, OPTIONAL, default=true]
Double **fixed** [INPUT, OPTIONAL, default=1.0]
Double **mixed** [INPUT, OPTIONAL, default=0.0]
Double **tolerance** [INPUT, OPTIONAL, default=0.01]
Integer **iterations** [INPUT, OPTIONAL, default=10000]
Double **temperature** [INPUT, OPTIONAL, default=0.0]
Double **cooling** [INPUT, OPTIONAL, default=0.95]
Integer **tempsteps** [INPUT, OPTIONAL, default=100]
Double1d **initialpars** [INPUT, OPTIONAL, default=from model]
Double1d **highlimits** [INPUT, OPTIONAL, default=null]
Double1d **lowlimits** [INPUT, OPTIONAL, default=null]
Int1d **keepfixed** [INPUT, OPTIONAL, default=null]
Double1d **fixedvalues** [INPUT, OPTIONAL, default=null]
Boolean **gui** [INPUT, OPTIONAL, default=false]

Limitations

Not everything which is possible using the package directly is accessable via this task.

Miscellaneous

In case of problems look at the trouble shooting section.

API details

Properties

NumericData x [INPUT, OPTIONAL, default=null]
The independent variable(s) of the fit problem. If not provided, it will use indices ranging from 0 to len(y). If y is more-dimensional the proper 2-d indices are provided.

DoubleArray y [INPUT, MANDATORY, default=no]

The data to be fitted. It can be more-dimensional. E.g. a 2-dimensional map.

Boolean indexview [INPUT, OPTIONAL, default=true]

Applicable parameters appear in the same order as in the y DoubleArray This parameter is only active when y has 2 dimensions or more.

DoubleArray weights [INPUT, OPTIONAL, default=null]

The weights to be used in the fit. Can be more-dimensional. When provided it should be the same size as parameter y.

AbstractModel model [INOUT, OPTIONAL, default=null]

The model to be fitted. One of "model" or "modelname" is MANDATORY

String modelname [INPUT, OPTIONAL, default=no]

The name of the model to be fitted. One of "model" or "modelname" is MANDATORY

Array1dData modelarg [INPUT, OPTIONAL, default=null]

Arguments, if any, needed in the Constructor of the model.

Fitter fitter [INOUT, OPTIONAL, default=null]

The fitter to be used. One of "fitter" or "fittername" is MANDATORY

String fittername [INPUT, OPTIONAL, default=no]

The name of the fitter to be used. One of "fitter" or "fittername" is MANDATORY

Double1d result [OUTPUT, OPTIONAL, default=n/a]

The parameters of the fit. Also the default result of the task.

Double1d parameters [OUTPUT, OPTIONAL, default=n/a]

The parameters of the fit. (same as result)

Double chisq [OUTPUT, OPTIONAL, default=n/a]

Chi-squared of the fit.

Double1d prior [INPUT, OPTIONAL, default=null]

Prior ranges for the parameters, needed when the evidence is requested. When noise scaling is set to auto=true (default), the prior needs 1 extra value as prior for the noise scale.

Double scale [OUTPUT, OPTIONAL, default=n/a]

Noise scale of the data.

IterationPlotable plotter [INPUT, OPTIONAL, default=null]

make a plot of the fit at each "plotfreq" iteration. A plotter is provided at herschel.ia.tool-box.fit.IterationPlotter

Integer plotfreq [INPUT, OPTIONAL, default=10]

to be used in conjunction with plotter.

Integer printfreq [INPUT, OPTIONAL, default=0]

Report about the present parameter settings every printfreq-th iteration.

Boolean auto [INPUT, OPTIONAL, default=true]

Select automatic noise scaling. The noise level in the data is not exactly known.

Double fixed [INPUT, OPTIONAL, default=1.0]

Fixed noise scale. The noise level is known.

Double mixed [INPUT, OPTIONAL, default=0.0]

Automatic noise scaling with a minimum. The noise level is not exactly known, but a minimum level is known.

Double tolerance [INPUT, OPTIONAL, default=0.01]

Stopping criterion for iterative fitting.

Integer iterations [INPUT, OPTIONAL, default=10000]

Maximum number of iterations in iterative fitters.

Double temperature [INPUT, OPTIONAL, default=0.0]

Starting temperature in annealing amoeba fitting.

Double cooling [INPUT, OPTIONAL, default=0.95]

Cooling step in annealing amoeba fitting.

Integer tempsteps [INPUT, OPTIONAL, default=100]

Number of exploratory steps at each temperature.

Double1d initialpars [INPUT, OPTIONAL, default=from model]

Initial parameters in iterative fitters.

Double1d highlimits [INPUT, OPTIONAL, default=null]

Upper limits of the parameters (only in AmoebaFitter)

Double1d lowlimits [INPUT, OPTIONAL, default=null]

Lower limits of the parameters (only in AmoebaFitter)

Int1d keepfixed [INPUT, OPTIONAL, default=null]

Keep these parameters fixed. (Parameter numbering starts at 0)

Double1d fixedvalues [INPUT, OPTIONAL, default=null]

Values at which the parameters should be kept.

Boolean gui [INPUT, OPTIONAL, default=false]

Allows to handle this task using a gui.

See also

- paper

- Developers Manual: herschel.ia.toolbox.fit.ModelFitTask

History

- `2006-10-15 - DK`: Creation

1.272. MonteCarloError

Full Name:	herschel.ia.numeric.toolbox.fit.MonteCarloError
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import MonteCarloError
Category	Mathematics/Fitting

Description

Fitter for linear models.

Example

Example 1: Fitter (for models which are linear in its parameters.)

```
# assume x and y are Double1d data arrays:
x = Double1d.range(100)
y = Double1d( Int1d.range(100).divide( 4 ) )         # digitization noise
poly = PolynomialModel( 1 )            # line
fitter = Fitter( x, poly )
param = fitter.fit( y )
stdev = fitter.getStandardDeviation()    # stdevs on the parameters
chisq = fitter.getChiSquared()
scale = fitter.getScale()            # noise scale
yfit  = fitter.getResult()            # fitted values
yfit  = poly( x )                # fitted values (same as previous)
yband = fitter.monteCarloError()     # 1 sigma confidence region
```

API Summary

Constructor
MonteCarloError (NumericData input, AbstractModel model)
Create a new Error, providing inputs and model.

Method
Double1d **getError** (NumericData input)
Calculates 1σ-confidence regions on the model given some inputs.

Limitations

1. The Fitter does **not** work with limits.

2. The calculation of the evidence is an Gaussian approximation which is only exact for linear models with a fixed scale.

Miscellaneous

In case of problems look at the trouble shooting section.

API Details

Constructor

MonteCarloError (NumericData input, AbstractModel model)
Create a new Error, providing inputs and model.

`MonteCarloError (NumericData input, AbstractModel model)`

A Error class is defined by its model and the input vector (the independent variable). When a fit to another model and/or another input vector is needed a new object should be created.

Arguments

`NumericData` **`input`** `[INPUT, MANDATORY, default=no default value]`

- Array of independent input values

`AbstractModel` **`model`** `[INPUT, MANDATORY, default=no default value]`

- The model function to be fitted

Method

`Double1d` **`getError (NumericData input)`**

Calculates 1σ-confidence regions on the model given some inputs.

From the full covariance matrix (= inverse of the Hessian) random samples are drawn, which are added to the parameters. With this new set of parameters the model is calculated. This procedure is done by default, 25 times. The standard deviation of the models is returned as the error bar.

Argument

`NumericData` **`input`** `[INPUT, MANDATORY, default=no default value ]`

Data over which to calculate the error bars (optional).

Return

Double1d

the confidence vector (same length as input)

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.MonteCarloError

1.273. MoreRandom

Full Name:	herschel.ia.numeric.toolbox.random.MoreRandom
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.random import MoreRandom
Category	Mathematics/Random numbers

Description

More Random utilities for use with the rest of classes of this package.

Example

Example 1: Example usages of a random function

```
f = MoreRandom()
f = MoreRandom(seed=6725)
```

API Summary

Constructors
MoreRandom
MoreRandom
MoreRandom (long seed)
MoreRandom

Methods
Int1d **permute** (int bound)
permute
double **nextUniform** (double lolim, double hilim)
nextUniform
double **nextJeffreys** (double lolim, double hilim)
nextJeffreys
double **nextInverseSquared** (double lolim, double hilim)
nextInverseSquared
double **nextGaussian** (double offset, double width)
nextGaussian
double **nextLorentz**
nextLorentz
double **nextLorentz** (double offset, double width)
nextLorentz
double **nextExponential**
nextExponential
double **nextExponential** (double scale)
nextExponential

Methods
double **nextGamma** (double c) nextGamma
int **nextPoisson** (double c) nextPoisson

API Details

Constructors

MoreRandom
MoreRandom Default constructor -- Construct a random number generator.

MoreRandom (long seed)
MoreRandom Constructor -- Construct a random number generator with a seed. **Argument** long **seed** [INPUT, MANDATORY, default=no default value] Seed to start the RNG.

Methods

Int1d **permute (int bound)**
permute Return a random permutation of numbers between [0,bound) Precondition: bound > 0 **Argument** int **bound** [INPUT, MANDATORY, default=no default value] Length of the returned Int1d **Return** **Int1d** A random permutation of numbers between [0,bound)

double **nextUniform (double lolim, double hilim)**
nextUniform Return a random sample from a Uniform distribution between limits. prob(x) = 1 Precondition: lolim < hilim **Arguments** double **lolim** [INPUT, MANDATORY, default=no default value] Low limit double **hilim** [INPUT, MANDATORY, default=no default value] High limit

double nextUniform (double lolim, double hilim)

Return

double

A random sample from a Uniform distribution between limits

double nextJeffreys (double lolim, double hilim)

nextJeffreys

Return a random sample from a Jeffreys prior distribution.

prob(x) = 1 / x for x > 0.

Precondition: lolim < hilim and both limits are positive.

Arguments

double **lolim** [INPUT, MANDATORY, default=no default value]

Low limit

double **hilim** [INPUT, MANDATORY, default=no default value]

High limit

Return

double

A random sample from a Jeffreys prior distribution

double nextInverseSquared (double lolim, double hilim)

nextInverseSquared

Return a sample from a 1 / x^2 distribution.

Precondition: lolim < hilim and both limits are positive.

Arguments

double **lolim** [INPUT, MANDATORY, default=no default value]

Low limit

double **hilim** [INPUT, MANDATORY, default=no default value]

High limit

Return

double

A sample from a 1 / x^2 distribution

double nextGaussian (double offset, double width)

nextGaussian

Return a random sample from a Gaussian with known offset and width.

Arguments

double **offset** [INPUT, MANDATORY, default=no default value]

Input offset

double **width** [INPUT, MANDATORY, default=no default value]

Input width

Return

double nextGaussian (double offset, double width)

double

A random sample from a Gaussian with known offset and width

double nextLorentz

nextLorentz

Return a random sample from a Lorentz-Cauchy distribution.

prob(x) = 1 / (π (x**2 + 1))

From http://www.johndcook.com/cpp_TR1_random.html#cauchy.

See also lighthouse problem.

See herschel.ia.numerix.toolbox.fit.LorentzModel

Return

double

A random sample from a Lorentz-Cauchy distribution

double nextLorentz (double offset, double width)

nextLorentz

Return a random sample from a Lorentz-Cauchy with known offset and width.

Arguments

`double` **`offset`** `[INPUT, MANDATORY, default=no default value]`

Input offset

`double` **`width`** `[INPUT, MANDATORY, default=no default value]`

Input width

Return

double

A random sample from a Lorentz-Cauchy with known offset and width

double nextExponential

nextExponential

Return a random sample from a exponential distribution.

prob(x) = exp(-x)

See herschel.ia.numerix.toolbox.fit.ExpModel

Return

double

A random sample from a exponential distribution

double nextExponential (double scale)

nextExponential

Return a random sample from a exponential with known scale.

***double* nextExponential (double scale)**

Argument

`double` **`scale`** `[INPUT, MANDATORY, default=no default value]`

Input scale

Return

double

A random sample from a exponential with known scale

***double* nextGamma (double c)**

nextGamma

Return a random sample from a gamma distribution.

prob(x) = x^(c-1) exp(x) / Gamma(c) for x in [0,inf)

Notes: x=0 can be returned, especially if c is small, but x=inf cannot. History: JS 24 Jan 1994, 19 Oct 1995, 24 Aug 1996

Argument

`double` **`c`** `[INPUT, MANDATORY, default=no default value]`

Mean value of the distribution.

Return

double

A random sample from a gamma distribution

***int* nextPoisson (double c)**

nextPoisson

Draw random sample from a Poisson distribution

prob(j) = exp(-c) c^j / j! (0 <= c <~ 4000000000)

Notes: Distribution is truncated at 2^32-1, so input parameter c should not approach 2^32. History: JS 15 May 1998, 24 Mar 2001

Argument

`double` **`c`** `[INPUT, MANDATORY, default=no default value]`

Mean value of the distribution.

Return

int

A random sample from a Poisson distribution

See also

- Developers Manual: herschel.ia.numeric.toolbox.random.MoreRandom

1.274. mosaic

Full Name:	herschel.ia.toolbox.image.MosaicTask
Alias:	mosaic
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import MosaicTask
Category	Images/Analysis

Description

This is a task to make mosaics, by

simple co-addition of images. Two blank maps are created : one to represent the total signal and one to represent the total exposure. For each pixel in each image, the pixel value is added into the total signal map and its exposure (or one) in the total exposure map (if not flagged out), taking only the overlap into account. After all pixels in each image have been mapped, the total signal map is divided by the total exposure map to produce the mosaic. When any of the input images have errors associated, the errors will also be weighted by the exposure. After weighting, the square root of the result is taken. Nevertheless, the preferred methods to calculate error/noise maps from coverage (at least in PACS) are the PhotCoverage2NoiseTask and PhotCoverage2NoisePointSourceTask tasks.

Examples

Example 1: This is an example of how you can make a mosaic and let the algorithm construct a spatial grid:

```
from java.util import ArrayList
images = ArrayList()
images.add(myImage1)
images.add(myImage2)
map = mosaic(images = images, oversample = True)
```

Example 2: This is an example of how you can make a mosaic with a given spatial grid:

```
from java.util import ArrayList
images = ArrayList()
images.add(myImage1)
images.add(myImage2)
map = mosaic(images = images, wcs = myWcs)
```

Example 3: To do median stacking of images, you don't need the mosaic task. The following example does pixel-by-pixel co-adding of images with the same dimension.

```
for i in range(len(images)):
  stack[i,:,:] = images[i].image
medianImage = images[0].copy()
medianImage.image = NAN_FILTER(MEDIAN(stack, 0))
```

API Summary

Properties
List<Image> **images** [INPUT, MANDATORY, default=None]
boolean **oversample** [INPUT, OPTIONAL, default=true]
Wcs **wcs** [INPUT, OPTIONAL, default=None]

Properties
Integer **algorithm** [INPUT, OPTIONAL, default=0]
Integer **weighting** [INPUT, OPTIONAL, default=-1]
SimpleImage **mosaic** [OUTPUT, MANDATORY, default=No default value]

API details

Properties

List<Image> images [INPUT, MANDATORY, default=None]

This is a list of images to combine.

boolean oversample [INPUT, OPTIONAL, default=true]

This indicates whether oversampling should be done. If it is set to false, the pixel scaling of the mosaic is the smallest (in absolute value) pixel scaling of all input images. The pixel scaling along the y-axis is always taken position (north up) and the pixel scaling along the x-axis is negative as soon as one of the input images has a negative cdelt1 (east left). If oversampling is enables, the same is done, except that the pixel scaling is divided by three (making the pixels three times smaller).

Wcs wcs [INPUT, OPTIONAL, default=None]

This is an optional spatial grid for the mosaic. This will only be used if it is valid, and has strictly positive dimensions. It is up to the user to make sure all input images fit on this spatial grid.

Integer algorithm [INPUT, OPTIONAL, default=0]

How the fluxes are computed. Possible values are 0 (AVERAGE) and 1 (SUM)

Integer weighting [INPUT, OPTIONAL, default=-1]

How the flux weighting and error calculation are computed. Possible values are -1 (AUTOMATIC), 0 (ERROR) or 1 (COVERAGE). Normally, the correct calculation is done with 'AUTOMATIC'.

SimpleImage mosaic [OUTPUT, MANDATORY, default=No default value]

This is the resulting mosaic.

See also

- Developers Manual: herschel.ia.toolbox.image.MosaicTask

1.275. MpFitter

Full Name:	herschel.ia.numeric.toolbox.fit.minpack.MpFitter
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit.minpack import MpFitter
Category	Mathematics/Fitting

Description

Non-linear fitter using MINPACK minimization.

MINPACK is a robust implementation of the Levenberg-Marquardt algorithm to fit the parameters of a non-linear model. It is a gradient fitter which uses partial derivatives to find the downhill gradient. Consequently it ends in the first minimum it finds. An excellent description of the method employed by MINPACK can be found in http://cran.r-project.org/web/packages/minpack.lm/minpack.lm.pdf, the Timur V. Ezhof et. al. description of the MINPACK method for the R interface or from the sources for the original Fortran version available as a download from the public domain FORTRAN sources of MINPACK bu J.J. More' http://ftp.netlib.org/minpack.

Example

Example 1: MpFitter (for non-linear models)

```
# Note: this is adapted from the LevenbergMarquardtFitter documentation.
# assume x and y are Double1d data arrays:
x = Double1d.range(100) / 10
y = Double1d.range(100) / 122            # make slope
rg = RandomGauss( seed=12345L )            # Gaussian random number generator
y += rg( Double1d(100) ) * 0.2             # add noise
y[Range(9,12)] += Double1d([5,10,7])         # make some peak
# define a model: GaussModel + background polynomial
gauss = GaussModel( )                              # Gaussian
gauss += PolynomialModel( 1 )                      # add linear background
gauss.setParameters( Double1d([1,1,0.1,0,0]) )    # initial parameter guess
print gauss.getNumberOfParameters()                  # 5 (= 3 for Gauss + 2 for
 line)
gauss.keepFixed( Int1d([2]), Double1d([0.1]))    # keep width fixed at 0.1
mpfit = MpFitter( x, gauss )
param = mpfit.fit( y )
print param.length()                               # 4 (= 5 - 1 fixed)
stdev = mpfit.getStandardDeviation()               # stdevs on the parameters
chisq = mpfit.getChiSquared()
scale = mpfit.getScale()                           # noise scale
yfit  = mpfit.getResult()                          # fitted values
yband = mpfit.monteCarloError()                      # 1 sigma confidence region
# for diagnostics (or just for fun)
# This is not yet supported.  TODO: Support this.
mpfit = MpFitter( x, gauss )
mpfit.setVerbose( 10 )                              # report every 10th iteration
plotter = IterationPlotter()                       # from herschel.ia.toolbox.fit
mpfit.setPlotter( plotter, 20 )                      # make a plot every 20th
 iteration
param = mpfit.fit( y )
```

API Summary

Methods
Double1d **fit** (Double1d data)
fit

Methods
String **toString** toString returns the name of the fitter.

Limitations

1. MP is **not** guaranteed to find the global minimum.

2. The calculation of the evidence is an Gaussian approximation which is only exact for linear models with a fixed scale. 3. Finally, and this can't be stressed enough, it is crucial to estimate the starting parameters as best you can. Initializing the parameters to all zeroes is actually the worst thing you can do, since then the problem becomes scale-less, and MPFIT has a much harder time deciding what to do. -C. Markwardt (http://cow.physics.wisc.edu/~craigm/idl/fitqa.html).

Miscellaneous

In case of problems look at the * trouble shooting section.

API Details

Methods

Double1d **fit (Double1d data)**
fit Perform MPFIT. Return the fit parameters as a Double1d. **Argument** `Double1d` **`data`** `[INPUT, MANDATORY, default=no default value ]` - The values to be fitted. **Return** **Double1d** parameters

String **toString**
toString returns the name of the fitter. **Return** **String** name of the fitter

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.minpack.MpFitter

1.276. multiply

Full Name:	herschel.ia.toolbox.spectrum.MultiplySpectrumTask
Alias:	multiply
Type:	Java Task -
Import:	from herschel.ia.toolbox.spectrum import MultiplySpectrumTask
Category	Spectra/Analysis

Description

Task for multiplying the flux data of spectra by a scalar or for pairwise multiplying spectra.

For the scalar mode, 'ds' and 'param' need to be specified - the input spectra and the scalar to multiply; for the pair-wise mode, 'ds1' and 'ds2' need to be set - two spectrum containers. In this pair-wise mode, the first spectrum in the first container is multiplied by the first spectrum in the second container, and so on. In case the size of the two containers is different, the result will contain a number of point spectra equal to the minimum size. Hence, the remaining spectra in the larger container are ignored. The behavior is different in case one of the datasets only contains a single spectrum: Here, the task always refers to single spectrum while iterating over all the spectra in the other container.

In the pairwise mode, the individual spectra are processed on a per frequency or wavelength bin (or whatever the wavescale unit is) basis. Hence, there is no check of whether the frequencies (or wavelengths) assigned to these bins are well aligned across the two spectra. If this is not the case, the spectra should first be resampled to a common wavescale grid.

The input data with the spectra to be processed needs to be an object that implements `SpectrumContainer` (e.g. `Spectrum1d, Spectrum2d, SpectralSimpleCube`). `SpectrumContainer`'s provides a uniform way to access spectrum data organized in quite different structures.

For a successful processing in the pair-wise mode, the data specified with `ds1` and `ds2` should be consistent, i.e. they should have the same wavescale range and spectral sampling. Furthermore, the segments found in all the spectra in the (two) containers should be consistent (same number of segments, same lengths, same segment indices - e.g. check with `ds.getSegmentIndices()`). Otherwise an exception is thrown.

Different selection schemes are available for selecting the point spectra or the segments to be processed. The most simple scheme is to specify lists of point spectrum indices (`selection=[1,3,4,2]`). In this case, the operation (scalar- or pair-operation) is just taken over the point spectra with corresponding indices. Instead of specifying a list of indices (which requires the knowledge of these) other recipes to specify a selection are available. Typical recipes test attribute values to match certain values (e.g. `selection={"bbtype":[6031, 6032]}`) or to be included in a given interval (e.g. `selection={"LoFrequency":(999.98,1000.02)}`). These recipes we refer to as selection models. The different options to specify selections can be combined. Here, an AND logic is adopted. See the parameter descriptions and the examples below for further detail. Alternatively, look in the `SelectSpectrum`-task.

Similarly, you can specify what segments to consider. In the most simple case, you can just specify the indices of segments to be considered (for the scalar operation `segments=[1,3,4]` and for the pair operation `segments1=[1,3,4], segments2=[3,6,5]`). In more advanced situations you can use a `SegmentSelection` object. Note that the pairs of segments to be processed need to be consistent - otherwise the processing will fail.

You can specify whether the orginal datasets should be overwritten with the 'overwrite' flag. By default no data is overwritten. In the pair-wise mode, it is not always possible to overwrite the data - in particular if non-trivial segment selections are specified.

Examples

Example 1: for scalar multiply:

```
global spectra # defined elsewhere
multiply = MultiplySpectrumTask()
spectraOut = multiply(ds=spectra, param=2.1)
# restrict the multiply to a suitable selection of spectra and return the
 processed
# just select by index:
spectraOut = multiply(ds=spectra, selection=[0,1,2,3], param=2.1)
# select by looking up suitable attributes attached to the spectra to be
 selected (here: "bbtype" matched):
spectraOut = multiply(ds=spectra, selection={"bbtype":[6031,6613]}, param=2.1)
# select by looking up suitable attributes attached to the spectra to be
 selected (here: "Chopper" in given ranges):
spectraOut = multiply(ds=spectra, selection={"Chopper":(4.,7.), "bbtype":
[6613]}, param=2.1)
# select by looking up suitable attributes attached to the spectra to be
 selected
# (here: "LoFrequency" in given interval):
spectraOut = multiply(ds=spectra, selection={"LoFrequency":(550.0,570.0)},
 param=2.1)
# the same as above - this time by by modifying the input spectra:
spectra = multiply(ds=spectra, selection={"bbtype":[6031,6613]}, param=2.1,
 overwrite=True)
spectra = multiply(ds=spectra, selection=[0,1,2,3], param=2.1, overwrite=True)
spectra = multiply(ds=spectra, selection={"Chopper":(4.,7.), "bbtype":[6613]},
 param=2.1, overwrite=True)
spectra = multiply(ds=spectra, selection={"LoFrequency":(550.0,570.0)},
 param=2.1, overwrite=True)
```

Example 2: for pairwise multiply:

```
global spectra1, spectra2 # defined elsewhere
multiply = MultiplySpectrumTask()
spectraOut = multiply(ds1=spectra1, ds2=spectra2)
# restrict the multiply to a suitable selection of spectra and return the
 processed
# just select by index:
spectraOut = multiply(ds1=spectra1, ds2=spectra2, selection=[0,1,2,3])
# select by looking up suitable attributes attached to the spectra to be
 selected (here: "bbtype" matched):
spectraOut = multiply(ds1=spectra1, ds2=spectra2, selection={"bbtype":
[6031,6613]})
# select by looking up suitable attributes attached to the spectra to be
 selected (here: "Chopper" in given ranges):
spectraOut = multiply(ds1=spectra1, ds2=spectra2, selection={"Chopper":(4.,7.),
 "bbtype":[6613]})
# select by looking up suitable attributes attached to the spectra to be
 selected
# (here: "LoFrequency" in given interval):
spectraOut = multiply(ds1=spectra1, ds2=spectra2, selection={"LoFrequency":
(550.0,570.0)})
```

API Summary

Properties
SpectrumContainer **ds** [INPUT, OPTIONAL, default=no default value.]
Double **param** [INPUT, OPTIONAL, default=no default value.]
Object **selection** [INPUT, OPTIONAL, default=None.]
PyDictionary|Map<String,Set<Object>> **lookup selection** [INPUT, OPTIONAL, default=no default value.]

Properties
PyList **index selection** [INPUT, OPTIONAL, default=No default value.]
Boolean **overwrite** [INPUT, OPTIONAL, default=False.]
SpectrumContainer **ds1** [INPUT, OPTIONAL, default=no default value.]
SpectrumContainer **ds2** [INPUT, OPTIONAL, default=no default value.]
Object **segments** [INPUT, OPTIONAL, default=no default value.]
Object **segments1** [INPUT, OPTIONAL, default=no default value.]
Object **segments2** [INPUT, OPTIONAL, default=no default value.]
SpectrumContainer **result** [OUTPUT, OPTIONAL, default=no default value]

API details

Properties

SpectrumContainer ds [INPUT, OPTIONAL, default=no default value.]

Input container to be processed by the task in case the task is used as a scalar operation. Examples of `SpectrumContainer` are `Spectrum1d, Spectrum2d, SpectralSimpleCube`.

Double param [INPUT, OPTIONAL, default=no default value.]

The scalar parameter to be considered when the task is used as scalar operation.

Object selection [INPUT, OPTIONAL, default=None.]

Specify what point spectra the operation should be applied to. Different ways to specify these selections are possible:

- Specify a list of indices (in jython) of the point spectra for which the multiply should be applied.
- Use a dictionary (in jython) to specify a selection model (lookup for attributes, filter attributes to be included in ranges or intervals).
- Pass a string representation of a jython list or a jython dictionary. The string will be parsed and the prcessing delegated to the two cases above.
- Pass any java instance that implements the SelectionModel interface (for the advanced user).

See the examples below or the SelectSpectrumTask for how to specify selections.

PyDictionary|Map<String,Set<Object>> lookup_selection [INPUT, OPTIONAL, default=no default value.]

Specify a PyDictionary with keys given by the attribute name to look up and a list of admissible values. Behind the scenes, a DiscreteValueSelectionModel is instantiated. This parameter is actually obsolete but is kept for historical reasons (use `selection`).

PyList index_selection [INPUT, OPTIONAL, default=No default value.]

Specify a PyList with the indices of the point spectra to be considered. This parameter is actually obsolete but is kept for historical reasons (use `selection`).

Boolean overwrite [INPUT, OPTIONAL, default=False.]

Specify whether the input data container can be reused - the values found therein is overwritten.

SpectrumContainer ds1 [INPUT, OPTIONAL, default=no default value.]

First input container for pair-wise operations.

SpectrumContainer ds2 [INPUT, OPTIONAL, default=no default value.]

Second input container for pair-wise operations.

Object segments [INPUT, OPTIONAL, default=no default value.]

Specify what segments the operation should be applied to. There are two options available:

- Either pass an instance of a SegmentSelection which gives the information on what segments for each point spectrum included in the container.
- Specify a PyList of segment indices.

Object segments1 [INPUT, OPTIONAL, default=no default value.]

Specify the segment selection to be associated with 'ds1'.

Object segments2 [INPUT, OPTIONAL, default=no default value.]

Specify the segment selection to be associated with 'ds2'.

SpectrumContainer result [OUTPUT, OPTIONAL, default=no default value]

Result object containing the results of the operation applied.

See also

- SpectrumContainer
- select
- Developers Manual: herschel.ia.toolbox.spectrum.MultiplySpectrumTask

History

- `2011-08-08 - melchior`: renamed from MultiplySpectrum

1.277. NAN_FILTER

Full Name:	herschel.ia.numeric.toolbox.basic.NaNFilter
Alias:	NAN_FILTER
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import NaNFilter
Category	Arrays and datasets/Element selection

Description

Returns an array without any NaN (Not a Number) elements.

This function returns a new array. The original array is left untouched. If the input array is multidimensional and contains NaN values, a one-dimensional array is returned.

This function works for arrays with fewer than 2^{31}-1 NaN elements.

Example

Example 1: Applying NAN_FILTER to a Double1d

```
x = Double1d([1,2,3])
print NAN_FILTER(x)              # [1,2,3] Original array
x = Double1d([1,Double.NaN,3])
print NAN_FILTER(x)              # [1,3]   New array
x = Double2d([[1,Double.NaN,3],[10,11,12]])
print NAN_FILTER(x)              # [1,3,10,11,12]   New array
```

API Summary

Jython Syntax

```
&lt;y&gt; = NAN_FILTER(&lt;x&gt;)
```

Properties

Float or Double array **x** [INPUT, MANDATORY, default=no default value]

Float or Double array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Float or Double array x [INPUT, MANDATORY, default=no default value]

The array from which to filter any NaN elements.

Float or Double array y [OUTPUT, MANDATORY, default=no default value]

A copy of the input array without any NaN elements.

See also

- Developers Manual: herschel.ia.numeric.toolbox.basic.NaNFilter

1.278. NearestNeighborInterpolator

Full Name:	herschel.ia.numeric.toolbox.interp.NearestNeighborInterpolator
Alias:	NearestNeighborInterpolator
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.interp import NearestNeighborInterpolator
Category	Mathematics/Interpolation

Description

This function interpolates values based on a set of known knots (x,y data).

Given a set of knots (x/y data), this function computes values at arbitrary positions by using the y value associated with the knot closest to the given x position. This can only be applied to numeric arrays of rank 1. For positions centered exactly between 2 knots, the y value from the smaller knot will be used.

Example

Example 1: Create and apply a NearestNeighborInterpolator on a Double1d

```
# create some knots
x=Double1d.range(10) # [0.0,1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0]
f=NearestNeighborInterpolator(x,SQUARE(x))
u=Double1d([1,1.1,2,2.6,3,3.9])
print f(u) # [1.0,1.0,4.0,9.0,9.0,16.0]
```

API Summary

Jython Syntax

```
&lt;f&gt;=NearestNeighborInterpolator(&lt;x&gt;,&lt;y&gt;[,al-
lowExtrapolation])
```

Properties

Double1d **x** [INPUT, MANDATORY, default=no default value]

Double1d **y** [INPUT, OPTIONAL, default=false]

boolean **allowExtrapolation** [INPUT, OPTIONAL, default=false]

API details

Properties

Double1d x [INPUT, MANDATORY, default=no default value]

The knots are Double1d

Double1d y [INPUT, OPTIONAL, default=false]

The knots are Double1d

boolean allowExtrapolation [INPUT, OPTIONAL, default=false]

Extrapolation is not allowed by default. Boolean.TRUE allows extrapolation.

See also

- CubicSplineInterpolator
- LinearInterpolator
- Developers Manual: herschel.ia.numeric.toolbox.interp.NearestNeighborInterpolator

1.279. nearestNeighbourProjection

Full Name:	herschel.ia.toolbox.spectrum.projection.NearestNeighbourProjectionTask
Alias:	nearestNeighbourProjection
Type:	Java Task -
Import:	from herschel.ia.toolbox.spectrum.projection import NearestNeighbourProjection-Task
Category	Data cubes

Description

Creates a spectral cube by converting the pixels specified by a Wcs into world coordinates and assigning the nearest spectral data to each pixel.

Spectral Data

Four types of spectral data may be used as input:

1. A Spectrum2d (containing columns of flux, ra, dec, wave, flag, and error data).
2. 3D arrays of corresponding flux, ra, dec, wave, flag, and error data
3. A `SpectralSimpleCube`
4. A list of `SpectralSimpleCube`(s)

Wcs

Data is projected using a Wcs. The shape of the spectral cube output is determined by the Wcs.

Examples

Example 1: How to create a spectral cube from 3d arrays

```
# Given: 3D arrays of flux, ra, and dec input (with variable names flux, ra and
 dec)
# Create a Wcs from a RA, Dec, wave data and units
target = WcsCreator.createWcs(ra, dec, wave, Angle.DEGREES, Angle.DEGREES,
 WaveNumber.RECIPROCAL_CENTIMETERS)
# Execute the task and create spectral cube.
cube = nearestNeighbourProjection(flux = flux, ra = ra, dec = dec, target =
 target)
```

Example 2: How to create a spectral cube from a Spectrum2d

```
cube = nearestNeighbourProjection(spectrum2d = spectrum2d, target = target)
```

Example 3: How to regrid a spectral cube

```
regriddedCube = nearestNeighbourProjection(cube = cube, target = target)
```

Example 4: How to create a spectral cube from a list of spectral cubes

```
cube = nearestNeighbourProjection(cubeList = cubes, target = target)
```

API Summary

Properties
Spectrum2d **spectrum2d** [INPUT, OPTIONAL, default=no default value]
SpectralSimpleCube **cube** [INPUT, OPTIONAL, default=no default value]
SpectralSimpleCube[] **cubeList** [INPUT, OPTIONAL, default=no default value]
Double3d **dec** [INPUT, OPTIONAL, default=no default value]
Double3d **ra** [INPUT, OPTIONAL, default=no default value]
Double3d **error** [INPUT, OPTIONAL, default=no default value]
Flag **flag** [INPUT, OPTIONAL, default=no default value]
Double3d **flux** [INPUT, OPTIONAL, default=no default value]
Wcs **target** [INPUT, MANDATORY, default=no default value]
MetaData **meta** [INPUT, OPTIONAL, default=no default value]
Unit **fluxUnit** [INPUT, OPTIONAL, default=no default value]
SpectralSimpleCube **spectralSimpleCube** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Spectrum2d spectrum2d [INPUT, OPTIONAL, default=no default value]

Primary product containing spectra and units used for making a spectral cube.

SpectralSimpleCube cube [INPUT, OPTIONAL, default=no default value]

SpectralSimpleCube element to be projected.

SpectralSimpleCube[] cubeList [INPUT, OPTIONAL, default=no default value]

List of SpectralSimpleCube elements to be projected.

Double3d dec [INPUT, OPTIONAL, default=no default value]

Declination values corresponding to flux values, having dimensions [Wavenumber,X,Y]. Wavenumber is a spectral dimension. X and Y are spatial dimensions.

Double3d ra [INPUT, OPTIONAL, default=no default value]

Right ascension values corresponding to flux, having dimensions [Wavenumber,X,Y]. Wavenumber is a spectral dimension. X and Y are spatial dimensions.

Double3d error [INPUT, OPTIONAL, default=no default value]

Error on the flux, having dimensions [Wavenumber, X, Y]. Wavenumber is a spectral dimension. X and Y are spatial dimensions.

Flag flag [INPUT, OPTIONAL, default=no default value]

Flag indicating which spectra are invalid, having dimensions [Wavenumber,X,Y]. Wavenumber is a spectral dimension. X and Y are spatial dimensions.

Double3d flux [INPUT, OPTIONAL, default=no default value]

Flux data having dimensions [Wavenumber, X,Y]. Wavenumber is a spectral dimension. X and Y are spatial dimensions.

Wcs target [INPUT, MANDATORY, default=no default value]

The target coordinates as a world coordinate system with three axes (two spatial and one spectral).

MetaData meta [INPUT, OPTIONAL, default=no default value]

Metadata to propagate to the SpectralSimpleCube. Regardless of whether this parameter is provided, the task will create the mandatory metadata parameters (e.g. "creator").

Unit fluxUnit [INPUT, OPTIONAL, default=no default value]

Flux unit to set in the SpecralSimpleCube output.

SpectralSimpleCube spectralSimpleCube [OUTPUT, MANDATORY, default=no default value]

The resulting projected cube.

See also

- Developers Manual: herschel.ia.toolbox.spectrum.projection.NearestNeighbourProjectionTask

History

- `2009-04-22 - CM`: [SPIRE SCR-1474] First version.

1.280. NestedSampler

Full Name:	herschel.ia.numeric.toolbox.fit.sample.NestedSampler
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit.sample import NestedSampler
Category	Mathematics/Fitting

Description

NestedSampler can be used to fit data to a model.

NestedSampler produces Samples from the posterior probability distribution of the parameters, given a AbstractModel and a dataset (x-values, y-values and optionally weights). The Samples are collected in a SampleList.

NestedSampler is constructed according to the ideas of John Skilling and David MacKay (ref TBD).

Internally it contains an ensemble (by default: 100) of trial samples; initially randomly distributed over the space available to the parameters. This randomness is distributed according to the prior distribution of the parameters. In most simple cases it will be uniform.

In an iterative process, the sample with the lowest likelihood is selected and replaced by a copy of one of the others. The parameters of the copy are randomly walked around under the condition that its likelihood stays larger than the selected lowest likelihood. A new independent trial sample is constructed. The original lowest sample is placed, appropriately weighted, into the SampleList for output.

This way the likelihood is climbed until the maximum is found. Along the way the integral of the likelihood function is calculated, which is equal to the evidence for the resulting parameters, given the dataset.

There are 4 different likelihood functions available: Gaussian (Normal), Laplace (Exponential), Poisson and Cauchy (Lorentz). By default the Gaussian distribution is used.

The noise scale can also be optimized, given the parameters and the data, and it is done by default for Gaussian and Laplace distributions. The Cauchy and Poisson distribution do not have a selectable noise scale. The prior for the noise scale is a Jeffreys distribution.

For the random walk of the parameters 4 so-called engines are written. By default they are all switched on.

- ScaleEngine. It walks the scale around according to Jeffreys distribution.
- StepEngine. It walks the parameters around, one by one according to a uniform distribution.
- FrogEngine. It averages the parameters from a number of other Samples and moves the parameters of the present Sample along a line defined by the average and the Sample, by a random factor.
- CrossEngine. It selects a random set from the parameters (and scale) and replaces them by the corresponding values from another Sample.

For Dynamic models 2 extra engines are defined:

- BirthEngine. It tries to increase the number of parameters.
- DeathEngine. It tries to decrease the number of parameters.

See also

- AbstractModel

- Developers Manual: herschel.ia.numeric.toolbox.fit.sample.NestedSampler

1.281. NoiseScale

Full Name:	herschel.ia.numeric.toolbox.fit.NoiseScale
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import NoiseScale
Category	Mathematics/Fitting

Description

NoiseScale contains a measure of the noise.

Information about the scale of the noise is stored in his class. It is either in the form of a fixed number, when the noise scale is known or in the form of a AbstractPrior. with limits. By default this prior is a JeffreysPrior

The full use of priors is reserved for Bayesian calculations as in NestedSampler.

See also

- JeffreysPrior
- NestedSampler
- Developers Manual: herschel.ia.numeric.toolbox.fit.NoiseScale

1.282. Normalize

Full Name:	herschel.ia.numeric.toolbox.basic.Normalize
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Normalize
Category	Arrays and datasets/Manipulation

Description

Normalizes sets of data.

The input data is a two-dimensional vectors where each row is a set of *N-1* independent variables plus one dependent variable: *(x1, x2, x3, ..., xN-1, y)*. The *y* values are normalized.

Four normalization options are available:

1. Normalize the peak *y* value to a user-specified value.
2. Normalize the peak *y* value to be the same as the peak value of a user-specified fiducial data set.
3. Normalize the mean of the *y* values over a range of *x* values to a user-specified value.
4. Normalizes the mean of the *y* values over a range of *x* values to the mean of a user-specified fiducial data set over the same range.

Usage notes:

- All the rows of the data set must have N-1 *x* values and one *y* value.
- The dimentions of the input data set must be same as the dimentions of the fiducial data set.
- The peak or mean value of the input data set cannot be zero.
- The peak or mean value of the input data set cannot have a different sign from the user-supplied peak or mean value.
- When a fiducial data set is used with a range of *x* values, the data set must contain data within that range.
- All dimensions *N* >= 2 are allowed. Data sets can include any integer or floating point data type.

Example

Example 1: from herschel.ia.numeric import *

```
from herschel.ia.numeric.toolbox.basic import Normalize
# Normalize data using the four available options.
# Type 1: Normalize y values to the common peak, a user-specified constant.
# Input data array.
arr = Double2d([(0.,1.,8.,40.),(10.,10.,4.,60.),(2.,0.,4.,120.)])
# User-specified peak value.
peak = 45.
# Normalize.
norm1 = Normalize(peak,Normalize.PEAK_CNST)(arr)
# Check output.
print norm1
# Output: [[0.0,1.0,8.0,15.0],[10.0,10.0,4.0,22.5],[2.0,0.0,4.0,45.0]]
# Type 2: Normalize y values to the max of a user-supplied "fiducial" array.
```

Example 1: from herschel.ia.numeric import *

```
# Input data array.
arr = Int2d([(0,1,8,40),(10,10,4,60),(2,0,4,120)])
# User-supplied fiducial array.
fiducial = Int2d([(1,1,1,10),(2,2,8,30),(4,6,10,20)])
# Normalize.
norm2 = Normalize(fiducial,Normalize.PEAK_FIDUCIAL)(arr)
# Check output.
print norm2
# Output: [[0,1,8,10],[10,10,4,15],[2,0,4,30]]
# Type 3: Normalize y values to have a user-specified constant mean value
# over a x range.
# Input data array.
arr = Double2d([(0.,1.,8.,40.),(1.,10.,4.,60.),(2.,0.,4.,120.)])
# User-specified mean value.
mean = 25.
# User-specified x range array.
xrange = Double2d([(0.,0.,0.),(2.,5.,10.)])
norm3 = Normalize(mean,xrange,Normalize.MEAN_CNST)(arr)
# Check output.
print norm3
# Output: [[0.0,1.0,8.0,12.5],[1.0,10.0,4.0,18.75],[2.0,0.0,4.0,37.5]]
# Type 4: Normalize y values to the mean value of a user-supplied fiducial
# array over a x range.
# Input data array.
arr = Double2d([(0.,1.,8.,40.),(10.,10.,4.,60.),(2.,0.,4.,120.)])
# User-supplied fiducial array.
fiducial = Double2d([(1.,1.,1.,15.),(2.,2.,8.,45),(4.,6.,10.,25.)])
# User-specified x range array.
xrange = Double2d([(0.,0.,0.),(3.,3.,10.)])
# Normalize.
norm4 = Normalize(fiducial,xrange,Normalize.MEAN_FIDUCIAL)(arr)
# Check results.
print norm4
# Output: [[0.0,1.0,8.0,15.0],[10.0,10.0,4.0,22.5],[2.0,0.0,4.0,45.0]]
```

API Summary

Jython Syntax

<result> = Normalize(<a>, [<b>,] <type>)(<input>)

Properties

- Number or 2_D array **a** [INPUT, MANDATORY, default=no default value]
- 2-D array **b** [INPUT, OPTIONAL, default=no default value]
- String **type** [INPUT, MANDATORY, default=no default value]
- 2-D array **input** [INPUT, MANDATORY, default=no default value]
- 2-D array **result** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Number or 2_D array a [INPUT, MANDATORY, default=no default value]

The peak or mean value to which to normalize the data. Alternatively, the fiducial two-dimensional array on which to compute the peak or mean value.

2-D array b [INPUT, OPTIONAL, default=no default value]

The range of *x* values for which the normalized values must have the specified mean value, supplied by parameter <*a*>. This parameter is a two-dimensional vector with two rows, specifying the lower and upper range limit of each *x* variable.

String type [INPUT, MANDATORY, default=no default value]

The type of normalization. Possible values:

1. **Normalize.PEAK_CNST** : normalize to user-supplied peak value.
2. **Normalize.PEAK_FIDUCIAL** : normalize to peak of fiducial array.
3. **Normalize.MEAN_CNST** : normalize to user-supplied mean value over a range of *x* values, specified by parameter <*b*>.
4. **Normalize.MEAN_FIDUCIAL** : normalize to the mean of a fiducial over a range of *x* values, specified by parameter <*b*>.

2-D array input [INPUT, MANDATORY, default=no default value]

Input data set. Two-dimensional vector where each row is a set of *N-1* independent variables plus one dependent variable: *(x1, x2, x3, ..., xN-1, y)*.

2-D array result [OUTPUT, MANDATORY, default=no default value]

The normalized data set. Same as input data set except for the normalized *y* values.

See also

- Developers Manual: herschel.ia.numeric.toolbox.basic.Normalize

1.283. NotPresent

Full Name:	herschel.ia.numeric.toolbox.basic.NotPresent
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import NotPresent
Category	Arrays and datasets/Element selection

Description

Checks whether none of the bits in the specified bitmask are present in the elements of the input array.

For example, if the bitmask is 7 (111 in binary) and one array element is 5 (101 in binary), the function returns `False`, for that element, because two bits in the mask are set in the array value.

Example

Example 1: Apply NotPresent to an Int1d

```
x=Int1d([0,1,2,3])
print NotPresent(3)(x)
#=&gt; [   (0 &amp; 3) == 0,   (1 &amp; 3) == 0,   (2 &amp; 3) == 0,   (3 &amp;
 3) == 0   ] =   [true,false,false,false]
print x.apply(NotPresent(3)) #Another way for using NotPresent
```

API Summary

Jython Syntax

```
&lt;y&gt;=NotPresent(&lt;bitmask&gt;)(&lt;x&gt;)
```

Properties

Integer type array **x** [INPUT, MANDATORY, default=no default value]

Integer **bitmask** [INPUT, MANDATORY, default=no default value]

Bool1d **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Integer type array x [INPUT, MANDATORY, default=no default value]

The input array. It can be of any integer type: ByteNd, ShortNd, IntNd, LongNd.

Integer bitmask [INPUT, MANDATORY, default=no default value]

The bitmask.

Bool1d y [OUTPUT, MANDATORY, default=no default value]

The result of the comparison. Each array element is `True` if none of the bits of the mask are present in the corresponding input array element, `False` otherwise.

See also

- AllPresent

- AnyPresent
- Developers Manual: herschel.ia.numeric.toolbox.basic.NotPresent

1.284. nSigmaClip

Full Name:	herschel.ia.toolbox.image.NSigmaClipTask
Alias:	nSigmaClip
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import NSigmaClipTask
Category	Images/Analysis

Description

This is a task to flag image values for which |value| is greater than n·σ.

In this task, n is an input parameter, and σ is the standard deviation of the values. The resulting flag has the "CLIPPED" flag type.

Example

Example 1: This is an example of how you can use the nSigma :

```
clipped = nSigmaClip(image = myImage, n = 5.0)
```

API Summary

Properties
`Image image [INPUT, MANDATORY, default=None]`
`Float n [INPUT, OPTIONAL, default=3.0]`
`Image clipped [OUTPUT, MANDATORY, default=None]`

API details

Properties

`Image image [INPUT, MANDATORY, default=None]`

This is the input image.

`Float n [INPUT, OPTIONAL, default=3.0]`

This is the clipping level.

`Image clipped [OUTPUT, MANDATORY, default=None]`

This is the resulting clipped image. It is a copy of the input image in which the clipped intensity values are flagged with the "CLIPPED" flag type.

See also

- Developers Manual: herschel.ia.toolbox.image.NSigmaClipTask

1.285. NullModel

Full Name:	herschel.ia.numeric.toolbox.fit.NullModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import NullModel
Category	Mathematics/Fitting

Description

NullModel represents models without parameters.

f(x:p) = f(x)

As such it is irrelevant whether it is linear or not. It has 0 params and returns 0 for its partials.

This NullModel also returns 0 for its result; it states that the data is merely noise.

f(x:p) = 0

The class could however be extended into one which returns something as a result, irrelevant what, as long as it is independent of parameters.

This might all seem quite irrelevant for fitting. And indeed no parameters can be fitted to these models, no standard deviations can be calculated, but it is possible to calculate the evidence for these models and compare them with more complicated models to decide whether there is any evidence for some structure at all.

See example

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.NullModel

1.286. ObservationContext

Full Name:	herschel.ia.obs.ObservationContext
Type:	Java Class -
Import:	from herschel.ia.obs import ObservationContext
Category	Arrays and datasets

Description

An Observation Context is a container of Products applicable to a specific observation.

It provides associations to products which are specific to a single observation (e.g. Telemetry Product, and reduced data products) as well as associations to Products that are applicable to multiple observations (such as the calibration products).

Example

Example 1: Setting up an observation

```
obs = ObservationContext()
auxContext = MapContext()
productContext = MapContext()
print obs.initialized # 0 (false)
obs.id=1L
obs.odNumber=2L
obs.instrument="HIFI"
obs.modelName="0"
obs.startTime=FineTime(1L)
obs.endTime=FineTime(2L)
print obs.initialized # 1 (true)
obs.auxiliary = auxContext # OK
print obs.auxiliary
# {description="Unknown", meta=[type, creator, creationDate, description,
#  instrument, modelName, startDate, endDate, formatVersion], datasets=[],
#  history=None, refs=[]}
print obs.prepared # 0 (false)
obs.calibration = MapContext()
print obs.prepared # 1 (true)
obs.level['level0'] = productContext   # OK
print obs.level['level0']
# {description="Unknown", meta=[type, creator, creationDate, description,
#  instrument, modelName, startDate, endDate, formatVersion],
# datasets=[], history=None, refs=[]}
print obs.reduced # 0 (false)
obs.level['level1']=MapContext()
print obs.reduced # 1 (true)
```

API Summary

Jython Syntax

```
<obs>=ObservationContext()
```

Property

ObservationContext **obs** [OUTPUT, MANDATORY, default=no default value]

API details

Property

ObservationContext obs [OUTPUT, MANDATORY, default=no default value]
Returns an empty ObservationContext

See also

- MapContext
- ProductRef
- Developers Manual: herschel.ia.obs.ObservationContext

1.287. openFile

Full Name:	herschel.ia.toolbox.util.OpenFileTask
Alias:	openFile
Type:	Java Task -
Import:	from herschel.ia.toolbox.util import OpenFileTask
Category	Session utilities

Description

Opens a file in a viewer.

Allows to open via script viewers associated to variables.

To open viewers for tasks use openTask task.

To open viewers for files use openFile task.

Examples

Example 1: Opening a jython script in the jython editor

```
openFile("myscript.py")
```

Example 2: Opening a fits file with its default viewer

```
openFile("product.fits")
```

API Summary

Jython Syntax

```
openFile(&lt;file&gt; [, &lt;viewer&gt;])
```

Properties

`String file [INPUT, MANDATORY, default=no default value]`

`String viewer [INPUT, OPTIONAL, default=no default value]`

Limitations

- The work of this task is done asynchronously (in the EDT): the command may finish before the GUI is shown. This means that if you expect to access the opened viewer in the next lines of your script, you need to synchronise or at least add a reasonable pause.
- If the file could not be opened (permissions, directory ...) a warning will be issued.

API details

Properties

`String file [INPUT, MANDATORY, default=no default value]`

The path of the file to open

String viewer [INPUT, OPTIONAL, default=no default value]
The ID of the viewer to open the file with

See also

- openVariable
- openTask
- http://download.oracle.com/javase/7/docs/api/java/lang/Thread.html#sleep%28long%29
- Developers Manual: herschel.ia.toolbox.util.OpenFileTask

History

- 2009-02-08 - JDS: first release

1.288. openSE

Full Name:	herschel.ia.toolbox.spectrum.explorer.task.OpenSpectrumExplorerTask
Alias:	openSE
Type:	Java Task -
Import:	from herschel.ia.toolbox.spectrum.explorer.task import OpenSpectrumExplorerTask
Category	Spectra

Description

Opens a Spectrum Explorer in the Editor Area.

Allows to open via script viewers associated to variable types.

Example

Example 1: Opening a spectrum with the Spectrum Explorer

```
ds = Spectrum1d(Double1d.range(5), Double1d.range(5), None, None, None)
openSE(ds)
```

API Summary

Jython Syntax

```
openSE(ds [, display])
```

Properties

```
Object ds [INPUT, MANDATORY, default=no default value]
Boolean display [INPUT, OPTIONAL, default=no default value]
SpectrumPlot plot [OUTPUT, MANDATORY, default=no default value]
```

API details

Properties

`Object ds [INPUT, MANDATORY, default=no default value]`

A spectrum plot, a spectrum container, or a list of spectrum containers.

`Boolean display [INPUT, OPTIONAL, default=no default value]`

If true, all spectra will be displayed in the plot.

`SpectrumPlot plot [OUTPUT, MANDATORY, default=no default value]`

A reference to the spectrum plot that is opened in the Spectrum Explorer.

See also

- Developers Manual: herschel.ia.toolbox.spectrum.explorer.task.OpenSpectrumExplorerTask

History

- `2011-08-08 - Werner`: First release
- `2013-10-24 - Jaime`: Fix task compliancy

1.289. openTask

Full Name:	herschel.ia.toolbox.util.OpenTaskTask
Alias:	openTask
Type:	Java Task -
Import:	from herschel.ia.toolbox.util import OpenTaskTask
Category	Session utilities

Description

Opens a task in its viewer.

Allows to open via code viewers associated to tasks (tools), assigning variables to inputs. This task will not check whether the inputName(s) validate the variableName(s)

To open viewers for variables use openVariable task.

To open viewers for files use openFile task.

Examples

Example 1: Open FitsReader task

```
openTask('fitsReader')
```

Example 2: Opening FitsReader task filling its prime input (file) with variable filename

```
filename = "/home/user/my.fits"
openTask("fitsReader", variableName="filename")
```

Example 3: Opening FitsReader task filling multiple parameters with variables

```
filename = "/home/user/my.fits"
type = "image"
openTask("fitsReader", inputName="file, fitsType", variableName="filename,
 type")
```

API Summary

Jython Syntax

```
openTask(&lt;taskName&gt; [, &lt;inputName&gt;, &lt;variable-
Name&gt;, &lt;registered&gt;=True])
```

Properties

`String` **`taskName`** `[INPUT, MANDATORY, default=null]`
`String` **`inputName`** `[INPUT, OPTIONAL, default=null]`
`String` **`variableName`** `[INPUT, OPTIONAL, default=null]`
`Boolean` **`registered`** `[INPUT, OPTIONAL, default=True]`

Limitations

- The work of this task is done asynchronously (in the EDT): the command may finish before the GUI is shown. This means that if you expect to access the opened viewer in the next lines of your script, you need to synchronise or at least add a reasonable pause.

- This task will not check whether the inputName(s) validate the variableName(s). It will not even check if the variables exist! If the task has not a inputName(s), it will be skipped (and its respective variableName).
- If the task is faulty, it may reject opening its GUI (check Warnings in the log)
- Not all tasks support opening with multiple variables (all tasks should, at least, support opening passing the primary input).

API details

Properties

String taskName [INPUT, MANDATORY, default=null]

Name of the task to be opened

String inputName [INPUT, OPTIONAL, default=null]

String with the name of the input to be assigned. If null or empty and variableName has a value, it implies the prime input. You can also pass a comma separated list of input names (that must match variableName in size).

String variableName [INPUT, OPTIONAL, default=null]

String with the name of the variable to fill the inputName with. You can also pass a comma separated list of variable names (that must match inputName in size).

Boolean registered [INPUT, OPTIONAL, default=True]

If true, open only registered tasks

See also

- openVariable
- openFile
- http://download.oracle.com/javase/7/docs/api/java/lang/Thread.html#sleep%28long%29
- Developers Manual: herschel.ia.toolbox.util.OpenTaskTask

History

- `2009-11-03 - JDS`: first release
- `2011-05-31 - JDS`: added registered flag (HCSS-12824)

1.290. openVariable

Full Name:	herschel.ia.toolbox.util.OpenVariableTask
Alias:	openVariable
Type:	Java Task -
Import:	from herschel.ia.toolbox.util import OpenVariableTask
Category	Session utilities

Description

Opens a variable in a viewer.

Allows to open via script viewers associated to variable types.

Example

Example 1: Opening a product variable with its default viewer

```
p = Product()
openVariable("p")
```

API Summary

Jython Syntax

```
openVariable(&lt;variable&gt; [, &lt;viewer&gt;])
```

Properties

```
String variable [INPUT, MANDATORY, default=no default value]
String viewer [INPUT, OPTIONAL, default=no default value]
```

Limitations

- The work of this task is done asynchronously (in the EDT): the command may finish before the GUI is shown. This means that if you expect to access the opened viewer in the next lines of your script, you need to synchronise or at least add a reasonable pause.
- This task only opens global variables. It does not even "see" local ones.
- Many types (for example all basic types: strings, ints, floats ...) have no viewer. In those cases this task will not open anything and just log a warning message. Use "Show Contents" in the contextual menu in Variables View to display the contents of any variable interactively.

API details

Properties

String variable [INPUT, MANDATORY, default=no default value]

Path of the file to be opened

String viewer [INPUT, OPTIONAL, default=no default value]

Viewer to open the variable with

See also

- openTask
- openFile
- http://download.oracle.com/javase/7/docs/api/java/lang/Thread.html#sleep%28long%29
- Developers Manual: herschel.ia.toolbox.util.OpenVariableTask

History

- `2008-12-15 - JDS`: first release
- `2008-12-16 - JDS`: added optional parameter viewer

1.291. OverPlotter

Full Name:	herschel.ia.gui.explorer.table.OverPlotter
Type:	Java Class -
Import:	from herschel.ia.gui.explorer.table import OverPlotter
Category	Arrays and datasets/Display

Description

Overview of OverPlotter

```
OverPlotter is an extension of TablePlotter and  allow users to overlay data on top
 of
each other and to compare. The OverPlotter can be seen as stacking many transparent
TablePlotters as layers on top of each other. Most TablePlotter features  work in
OverPlotter, especially when working on an individual layer. OverPlotter layers have
 three
states, active, secondary active and in active.   Each OverPlotter layer has
its own personalities.  The personalities are unchanged no matter the layer is
 active,
secondary active and inactive. \
```

Examples

Example 1: Invoke OverPlotter in command line

```
from herschel.ia.dataset import Column, TableDataset
from herschel.ia.gui.explorer.table import OverPlotter
from herschel.share.component import WindowManager
from herschel.ia.numeric.toolbox.basic import Cos, Sin
#
#create a TableDataset
table = TableDataset()
x = Double1d.range(100)
y1=SIN(x)
y2=COS(x)
table["x-data"]=Column(x)
table["y1-data"] = Column(y1)
table["y2-data"] = Column(y2)
wm=WindowManager.getDefault() #Load OverPlotter
overPlotter=OverPlotter(table)
overPlotter.name = "Test OverPlotter"
wm.addWindow(overPlotter.name,overPlotter.component, 1) #add a new layer
 table1=table
overPlotter.object=table
```

Example 2: Create a new OverPlotter with one layer and then add the second layer

```
from herschel.ia.gui.explorer.table import OverPlotter
from herschel.ia.dataset import Column, TableDataset
from herschel.share.component import WindowManager
from herschel.ia.numeric.toolbox.basic import Cos, Sin
from herschel.ia.numeric import Double1d
table = TableDataset()
x = Double1d.range(100)
y1=SIN(x)
y2=COS(x)
table["x-data"]=Column(x)
table["y1-data"] = Column(y1)
table["y2-data"] = Column(y2)
# add first layer
overPlotter = OverPlotter(table)
```

Example 2: Create a new OverPlotter with one layer and then add the second layer

```
#add a new layer
overPlotter.object = table
wm=WindowManager.getDefault() #Load OverPlotter
wm.addWindow(overPlotter.name,overPlotter.component, 1)
```

API Summary

Constructors
OverPlotter
The default constructor This constructor is used to initializes
OverPlotter (TableDataset tds)
A constructor This constructor is used to initiate an instance of

Methods
JComponent **getComponent**
This method will return OverPlotter as a pluggable component This
String **getDescription**
setObject (TableDataset data)
Initiate a new instance of OverPlotter or add a new layer to the

API Details

Constructors

OverPlotter

The default constructor This constructor is used to initializes

the _layerCounter to 0.

OverPlotter (TableDataset tds)

A constructor This constructor is used to initiate an instance of

TablePlotter from the command line.

Argument

`TableDataset` **`tds`** `[INPUT, MANDATORY, default=no default value]`

-Input TableDataset

Methods

JComponent **getComponent**

This method will return OverPlotter as a pluggable component This

method allows TablePlotter to be used as a plug-ins. The user can plug OverPlotter to his/her own applications.

Return

JComponent

the OverPlotter as a component

String **getDescription**

Return

String

the description of the OverPlotter

setObject (TableDataset data)

Initiate a new instance of OverPlotter or add a new layer to the

existing OverPlotter. When this method is called the first time, it will initiate a new instance of OverPlotter and then assign the class variable _activeOverPlotter to the newly created object. When it is called again, it will add a new OverPlotter layer on to the existing OverPlotter object, _activeOverPlotter.

Argument

`TableDataset` **`data`** `[INPUT, MANDATORY, default=no default value data]`

is a TableDataset. It will be passed as an active data structure to be plotted.

See also

- Developers Manual: herschel.ia.gui.explorer.table.OverPlotter

History

- `2011-10-14 - Implemented`: HCSS-14232

1.292. PackedMask

Full Name:	herschel.ia.numeric.toolbox.mask.PackedMask
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.mask import PackedMask
Category	Mathematics/Masks

Description

PackedMask creates and stores compressed mask array data which can be set, unset and checked.

This way no memory is wasted. Also no limit on the total bits of mask (in FixedMask, the maximum number of bits in one mask data is 64.)

Examples

Example 1: Create a PackedMask:

```
#An 2x3 2-d mask array is created by
mask=PackedMask("maskname", [2,3,129])
#Note: The previous command defines a mask holding 129 bits for each element of
 a 2x3 array.
#However, since mask bits are stored as integers, and an integer is 32 bits
 long, five
#integers (129/32 + 1) are needed to store <em>at least</em> 129 bits for
#each array element (four integers would stop at 128 bits). Therefore the mask
 ends up
#holding 32 x 5 = 160 bits.
#
#The mask is stored internally as a 2x3x5 integer array, as shown by the
 following command:
print mask.getMask()
#[ [ [0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0] ], [ [0,0,0,0,0], [0,0,0,0,0],
 [0,0,0,0,0] ] ]
print mask.getName()
#maskname
#
```

Example 2: Set/Unset and check mask:

```
mask=PackedMask("maskname", [2,3,129])
#One mask data in the array can be set by
print mask.set([1,2,3])# The [1,2] mask data is set by bit offset 3
#[ [ [0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0] ], [ [0,0,0,0,0], [0,0,0,0,0],
 [8,0,0,0,0] ] ]
#The mask data can be checked by
print mask.isSet([1,2,3])
#1
print mask.set([1,1,2])# The [1,1] mask data is set by bit offset 2
#[ [ [0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0] ], [ [0,0,0,0,0], [4,0,0,0,0],
 [8,0,0,0,0] ] ]
#One mask data in the array can be unset by
print mask.unset([1,1,2])
#[ [ [0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0] ], [ [0,0,0,0,0], [0,0,0,0,0],
 [8,0,0,0,0] ] ]
#The mask data can be checked by
print mask.isSet([1,1,2])
#0
# With FixedMask, the maximum number of bits to set is 64.
# With PackedMask,  we use bit offset 0 to 31 to set the first integer, 32 to
 63 the second, ....
print mask.set([1,0,65])
#[ [ [0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0] ], [ [0,0,2,0,0], [0,0,0,0,0],
 [8,0,0,0,0] ] ]
```

Example 2: Set/Unset and check mask:

```
print mask.set([0,1,157])
#[ [ [0,0,0,0,0], [0,0,0,0,536870912], [0,0,0,0,0] ], [ [0,0,2,0,0],
 [0,0,0,0,0], [8,0,0,0,0] ] ]
#
```

Example 3: Multi dimensions (up to 5D) PackedMask array:

```
#1D:
mask1=PackedMask("m1", [33])
print mask1.getMask()
#[0,0]
print mask1.set([3])
#[8,0]
#2D:
mask2=PackedMask("m2", [2,100])
print mask2.getMask()
#[ [0,0,0,0], [0,0,0,0] ]
print mask2.set([1,4])
#[ [0,0,0,0], [16,0,0,0] ]
#3D:
mask3=PackedMask("m3", [2,3,64])
print mask3.getMask()
#[ [ [0,0], [0,0], [0,0] ], [ [0,0], [0,0], [0,0] ] ]
print mask3.set([0,1,5])
#[ [ [0,0], [32,0], [0,0] ], [ [0,0], [0,0], [0,0] ] ]
#4D:
mask4=PackedMask("m4", [2,3,4,129])
print mask4.getMask()
print mask4.set([1,2,3,150])
#5D:
mask5=PackedMask("m5", [2,3,4,5,129])
print mask5.getMask()
print mask5.set([1,2,3,4,150])
#
#End of jexample.
```

See also

- FixedMask
- Developers Manual: herschel.ia.numeric.toolbox.mask.PackedMask

History

- `2007-07-01 - SG`: Original, based on MW code.
- `2009-06-01 - SG`: SPR HCSS-6831. Set wrong mask data.
- `2009-09-01 - SG`: SCR HCSS-6832 and 6834. UM and URM.

1.293. PacketSequence

Full Name:	herschel.binstruct.PacketSequence
Alias:	PacketSequence
Type:	Java Class -
Import:	from herschel.binstruct import PacketSequence
Category	binstruct

Description

A container for telemetry and telecommand source packets.

In principle the `PacketSequence` is a general purpose container for telemetry and telecommand packets. However most of the time it will be used to group all the information of one observation or test because those packets have a natural connection.

Example

Example 1: Loading a packet sequence

```
seq = PacketSequence(PacketReader(filename))
```

See also

- Developers Manual: herschel.binstruct.PacketSequence

1.294. PadeModel

Full Name:	herschel.ia.numeric.toolbox.fit.PadeModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import PadeModel
Category	Mathematics/Fitting

Description

General Pade model of arbitrary degrees in numerator and denominator.

$$f(x:p) = \sum p_n * x^n / (1 + \sum p_{num+k} * x^k)$$

where the sum in the numerator is over n running from 0 to num (inclusive) and the sum in the denominator is over k running from 1 to den (inclusive)

All parameters are initialized at 1. It is a non-linear model.

Beware of the poles where the denominator equals zero.

See example

Example

Example 1: PadeModel

```
pade = PadeModel( 3, 1 )            # 3rd degree polynomial
print pade.getNumberOfParameters()        # 5
```

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.PadeModel

1.295. pairAvg

Full Name:	herschel.ia.toolbox.spectrum.PairAverageSpectrumTask
Alias:	pairAvg
Type:	Java Task -
Import:	from herschel.ia.toolbox.spectrum import PairAverageSpectrumTask
Category	Spectra/Analysis

Description

Task for pairwise averaging spectra.

By setting `ds1` and `ds2` you pass two spectrum containers that contain the spectrum data. The first spectrum in the first and the first spectrum in the second container are averaged, and so on. In case the size of the two containers is different, the result will contain a number of point spectra equal to the minimum size. Hence, the remaining spectra in the larger container are ignored. The behavior is different in case one of the datasets only contains a single spectrum: Here, the task always refers to single spectrum while iterating over all the spectra in the other container.

In the pairwise mode, the individual spectra are processed on a per frequency or wavelength bin (or whatever the wavescale unit is) basis. Hence, there is no check of whether the frequencies (or wavelengths) assigned to these bins are well aligned across the two spectra. If this is not the case, the spectra should first be resampled to a common wavescale grid.

The input data with the spectra to be processed needs to be an object that implements `SpectrumContainer` (e.g. `Spectrum1d, Spectrum2d, SpectralSimpleCube`). `SpectrumContainer`'s provides a uniform way to access spectrum data organized in quite different structures.

For a successful processing, the data specified with `ds1` and `ds2` should be consistent, i.e. they should have the same wavescale range and spectral sampling. Furthermore, the segments found in all the spectra in the (two) containers should be consistent (same number of segments, same lengths, same segment indices - e.g. check with `ds.getSegmentIndices()`). Otherwise an exception is thrown.

Different selection schemes are available for selecting the point spectra or the segments to be processed. The most simple scheme is to specify lists of point spectrum indices (`selection=[1,3,4,2]`). In this case, the operation (scalar- or pair-operation) is just taken over the point spectra with corresponding indices. Instead of specifying a list of indices (which requires the knowledge of these) other recipes to specify a selection are available. Typical recipes test attribute values to match certain values (e.g. `selection={"bbtype":[6031, 6032]}`) or to be included in a given interval (e.g. `selection={"LoFrequency":(999.98,1000.02)}`). These recipes we refer to as selection models. The different options to specify selections can be combined. Here, an AND logic is adopted. See the parameter descriptions and the examples below for further detail. Alternatively, look in the `SelectSpectrum`-task.

Similarly, you can specify what segments to consider. In the most simple case, you can just specify the indices of segments to be considered (for the scalar operation `segments=[1,3,4]` and for the pair operation `segments1=[1,3,4], segments2=[3,6,5]`). In more advanced situations you can use a `SegmentSelection` object. Note that the pairs of segments to be processed need to be consistent - otherwise the processing will fail.

You can specify whether the orginal datasets should be overwritten with the 'overwrite' flag. By default no data is overwritten. Note that it is not always possible to overwrite the data, though - in particular if non-trivial segment selections are specified.

There are further options that can be set by the `variant`-parameter. Available are:

- "flux": Take simple average of flux arrays, add weight arrays, propagate flags adopting a bit-wise OR and take the wave array of `ds2`;
- "flux-weight": Take weighted average of flux arrays, add weight arrays, propagate flags adopting a bit-wise OR and take the wave array of `ds2`;
- "flux-flag": Take simple average of flux arrays but ignore flagged bins, add weight arrays, propagate flags adopting a bit-wise OR and take the wave array of `ds2`;
- "flux-flag-weight": Take weighted average of flux arrays but ignore flagged bins, add weight arrays, propagate flags adopting a bit-wise OR and take the wave array of `ds2`;

For each of the above option an alternative exists with '-wave' appended. In those cases, the wave arrays of `ds1` and `ds2` are computed by computing an average consistent with the type of average applied to the flux values (simple, using weights or flags or both). Note that not all implementations of `SpectrumContainer` allow to have individual wavescale defined for each and every point spectrum. Furthermore, note that for PACS and SPIRE, we rather have error and mask instead of weights and flags, respectively. Typically, the PACS and SPIRE-specific implementations of the spectrum data containers map the error to weights (e.g. by defining the weights to be inverse squared error). Furthermore, the mask is just mapped 'as is' to flags.

Example

Example 1: Examples for pairwise averaging of spectra

```
global spectra1, spectra2 # defined elsewhere
pairAvg = PairAverageSpectrumTask()
spectraOut = pairAvg(ds1=spectra1, ds2=spectra2)
# restrict the add to a suitable selection of spectra and return the processed
# just select by index:
spectraOut = pairAvg(ds1=spectra1, ds2=spectra2, selection=[0,1,2,3])
# select by looking up suitable attributes attached to the spectra to be
 selected (here: "bbtype" matched):
spectraOut = pairAvg(ds1=spectra1, ds2=spectra2, selection={"bbtype":[6613]})
# select by looking up suitable attributes attached to the spectra to be
 selected (here: "Chopper" in given ranges):
spectraOut = pairAvg(ds1=spectra1, ds2=spectra2, selection={"Chopper":(4.,7.),
 "bbtype":[6613]})
# select by looking up suitable attributes attached to the spectra to be
 selected
# (here: "LoFrequency" in given interval):
spectraOut = pairAvg(ds1=spectra1, ds2=spectra2, selection={"LoFrequency":
(500.0,600.0)})
```

API Summary

Properties
SpectrumContainer **ds1** [INPUT, OPTIONAL, default=no default value.]
SpectrumContainer **ds2** [INPUT, OPTIONAL, default=no default value.]
Object **selection** [INPUT, OPTIONAL, default=None.]
Boolean **overwrite** [INPUT, OPTIONAL, default=False.]
Object **segments1** [INPUT, OPTIONAL, default=no default value.]
Object **segments2** [INPUT, OPTIONAL, default=no default value.]
String **variant** [INPUT, OPTIONAL, default=no default value.]
SpectrumContainer **result** [OUTPUT, OPTIONAL, default=no default value]

API details

Properties

SpectrumContainer ds1 [INPUT, OPTIONAL, default=no default value.]

First input container for pair-wise operations.

SpectrumContainer ds2 [INPUT, OPTIONAL, default=no default value.]

Second input container for pair-wise operations.

Object selection [INPUT, OPTIONAL, default=None.]

Specify what point spectra the operation should be applied to. Different ways to specify these selections are possible:

- Specify a list of indices (in jython) of the point spectra for which the pair-avg should be applied.
- Use a dictionary (in jython) to specify a selection model (lookup for attributes, filter attributes to be included in ranges or intervals).
- Pass a string representation of a jython list or a jython dictionary. The string will be parsed and the processing delegated to the two cases above.
- Pass any java instance that implements the SelectionModel interface (for the advanced user).

See the examples below or the SelectSpectrumTask for how to specify selections.

Boolean overwrite [INPUT, OPTIONAL, default=False.]

Specify whether the input data container can be reused - the values found therein is overwritten.

Object segments1 [INPUT, OPTIONAL, default=no default value.]

Specify the segment selection to be associated with 'ds1'.

Object segments2 [INPUT, OPTIONAL, default=no default value.]

Specify the segment selection to be associated with 'ds2'.

String variant [INPUT, OPTIONAL, default=no default value.]

Specify the variant of processing mode you would like to run (including flags / weights / ...). HCSS defaults include: "flux" / "flux-wave" / "flux-wave-weight" / "flux-flag-wave" / "flux-weight-flag-wave"

SpectrumContainer result [OUTPUT, OPTIONAL, default=no default value]

Result object containing the results of the operation applied.

See also

- SpectrumContainer
- pairAvg

- select
- Developers Manual: herschel.ia.toolbox.spectrum.PairAverageSpectrumTask

History

- `2011-08-08 - melchior`: renamed from PairAverageSpectrum

1.296. ParameterCube

Full Name:	herschel.ia.dataset.spectrum.ParameterCube
Type:	Java Class -
Import:	from herschel.ia.dataset.spectrum import ParameterCube

Description

A ParameterCube is a Cube where the parameters of a model fit to a spectral cube are stored.

The image layers of the ParameterCube each represent a parameter of the model.

The ParameterCube also contains a FitModelDataset, which can be used to resurrect the model of the fit.

Standard Deviations are stored in the error cube.

The chi^2 is stored as an image while the parameter names and parameter units are stored as vectors in the same order as the layers in the image.

In general ParameterCube's are obtained from a multifit with the SpectrumFitter.

Example

Example 1: Assume parcub is a ParameterCube obtained from SpectrumFitter

```
# Double3d containing all parameters
p3 = parcub.getFitParameters()
# Double1d fit parameters at image location (2, 3)
p1 = parcub.getFitParameters(2, 3)
# String1d with the names of the parameters
pn = parcub.getParameterNames()
# String1d with the units of the parameters
pu = parcub.getParameterUnits()
# AbstractModel pertaining to the parameters
m = parcub.getFitModel()
# Assume wave is a Double1d containing an input vector to the model
fc = parcub.getFittedCube(wave) # SpectralSimpleCube with the model cube
```

API Summary

Constructors
ParameterCube The construction of a new ParameterCube. It holds fit parameters instead of fluxes and each
ParameterCube (ParameterCube cube) The construction of a new ParameterCube from an existing one. The construction of a new

Methods
ParameterCube **copy** Makes a copy of this ParameterCube. Makes a copy of this ParameterCube.
setFitParameters (Double3d fitParameters) Sets the fit parameters for this ParameterCube. Sets the fit parameters for this
setFitModel (String modelVariable, String modelScript) Stores the name fit model name in the metadata. Stores the name fit model name in the

Methods
Double3d **getFitParameters** Returns the fit parameters. Returns the fit parameters for this ParameterCube.
Double1d **getFitParameters** (int row, int col) Returns the fit parameters for this ParameterCube.
Double3d **getStandardDeviations** Returns the standard deviations. Returns the standard deviations for this ParameterCube.
Double1d **getStandardDeviations** (int row, int col) Returns the standard deviations for this ParameterCube.

API Details

Constructors

ParameterCube

The construction of a new ParameterCube. It holds fit parameters instead of fluxes and each

layer holds one kind of fit parameter. E.g. in case of a GaussianModel the first layer could hold the peak flux, the second layer holds the line width, etc.

ParameterCube (ParameterCube cube)

The construction of a new ParameterCube from an existing one. The construction of a new

ParameterCube from an existing one.

Argument

ParameterCube **cube** [INPUT, MANDATORY, default=no default value]

The ParameterCube that should be copied, as ParameterCube

Methods

***ParameterCube* copy**

Makes a copy of this ParameterCube. Makes a copy of this ParameterCube.

Return

ParameterCube

A copy of this ParameterCube.

setFitParameters (Double3d fitParameters)

Sets the fit parameters for this ParameterCube. Sets the fit parameters for this

ParameterCube to the given fit parameters.

Argument

Double3d **fitParameters** [INPUT, MANDATORY, default=no default value]

The fit parameters to set as the fit parameters for this ParameterCube.

setFitModel (String modelVariable, String modelScript)

Stores the name fit model name in the metadata. Stores the name fit model name in the

metadata for this ParameterCube.

setFitModel (String modelVariable, String modelScript)

Arguments

String **modelVariable** [INPUT, MANDATORY, default=no default value]

The model variable

String **modelScript** [INPUT, MANDATORY, default=no default value]

The model script

Double3d **getFitParameters**

Returns the fit parameters. Returns the fit parameters for this ParameterCube.

Return

Double3d

The fit parameters for this ParameterCube.

Double1d **getFitParameters (int row, int col)**

Returns the fit parameters for this ParameterCube.

Arguments

int **row** [INPUT, MANDATORY, default=no default value]

the row of the spectrum

int **col** [INPUT, MANDATORY, default=no default value]

the col of the spectrum

Return

Double1d

The fit parameters for this ParameterCube.

Double3d **getStandardDeviations**

Returns the standard deviations. Returns the standard deviations for this ParameterCube.

Return

Double3d

The standard deviations for this ParameterCube.

Double1d **getStandardDeviations (int row, int col)**

Returns the standard deviations for this ParameterCube.

Arguments

int **row** [INPUT, MANDATORY, default=no default value]

the row of the spectrum

int **col** [INPUT, MANDATORY, default=no default value]

the col of the spectrum

Return

Double1d

The standard deviations for this ParameterCube.

See also

- Developers Manual: herschel.ia.dataset.spectrum.ParameterCube

1.297. Planck

Full Name:	herschel.ia.toolbox.astro.Planck
Alias:	Planck
Type:	Java Class -
Import:	from herschel.ia.toolbox.astro import Planck
Category	Astronomical utilities

Description

Computes the Planck function.

This function takes two or three arguments: wave data, a temperature in kelvin, and an optional unit (default is Hertz) specifying the unit of the wave data. It returns spectral radiance in units of $W/m^2/sr/Hz$.

If no wave data unit is specified, hertz is assumed. The unit must be a wavenumber, wavelength, or frequency; any other type will throw an exception.

Examples

Example 1: Compute the Planck function for the given wavenumbers and temperature.

```
from herschel.share.unit.WaveNumber import RECIPROCAL_CENTIMETERS
wave = Double1d.range(1001).divide(100).add(10)
temp = 88.0
result = Planck(temp, RECIPROCAL_CENTIMETERS)(wave)
```

Example 2: Compute the Planck function for the given frequencies and temperature.

```
from herschel.share.unit.Frequency import HERTZ
wave = Double1d.range(1001).divide(100).add(10)
temp = 88.0
result = Planck(temp, HERTZ)(wave)
```

Example 3: If no wave data unit is provided, the function assumes the wave data is in hertz.

```
wave = Double1d.range(1001).divide(100).add(10)
temp = 88.0
result = Planck(temp)(wave)
```

Example 4: The function supports most numeric types.

```
wave = Float1d.range(1001).divide(100).add(10)
temp = 88.0
result = Planck(temp)(wave)
# The function also supports Python lists.
wave = [10.0,10.01,10.02,10.03,10.04]
result = Planck(temp)(wave)
```

API Summary

Constructors
Planck (Number temp, Unit waveUnit) Creates a Planck object.
Planck (Number temp)

Constructors
Creates a Planck object.

Methods
Double1d **planck** (Double1d wave, double temp, Unit waveUnit) Compute the Planck function for the given wave data and temperature.
Double1d **planck** (Double1d wave, double temp) Computes the Planck function for the given wave data and temperature.
double **calc** (double wave) Computes the Planck function for the given wave data using a temperature

Limitations

- This task was originally located in `herschel.spire.ia.pipeline.spec.util`

API Details

Constructors

Planck (Number temp, Unit waveUnit)

Creates a Planck object.

Creates a Planck object with a fixed temperature and unit.

Arguments

Number **temp** [INPUT, MANDATORY, default=no default value]

A temperature in kelvin.

Unit **waveUnit** [INPUT, OPTIONAL, default=Hertz]

The unit of the wave data. The unit must be a wavenumber, wavelength, or frequency; any other type will throw an exception.

Return

Planck

a configured Planck object

Examples

Compute the Planck function for the given wavenumbers and temperature.

```
from herschel.share.unit.WaveNumber import RECIPROCAL_CENTIMETERS
wave = Double1d.range(1001).divide(100).add(10)
temp = 88.0
result = Planck(temp, RECIPROCAL_CENTIMETERS)(wave)
```

Compute the Planck function for the given frequencies and temperature.

```
from herschel.share.unit.Frequency import HERTZ
wave = Double1d.range(1001).divide(100).add(10)
temp = 88.0
result = Planck(temp, HERTZ)(wave)
```

Planck (Number temp)

Creates a Planck object.

Creates a Planck object with a fixed temperature and using Hertz as unit. See the other constructor.

`Planck (Number temp)`

Argument

`Number` **`temp`** `[INPUT, MANDATORY, default=no default value]`

A temperature in kelvin.

Return

Planck

a configured Planck object

Example

If no wave data unit is provided, the function assumes the wave data is in hertz.

```
wave = Double1d.range(1001).divide(100).add(10)
temp = 88.0
result = Planck(temp)(wave)
```

Methods

`Double1d` **`planck (Double1d wave, double temp, Unit waveUnit)`**

Compute the Planck function for the given wave data and temperature.

This is a static Java method (call directly with `Planck.planck(...)`.

Arguments

`Double1d` **`wave`** `[INPUT, MANDATORY, default=no default value]`

Wave data. Must not contain negative values.

`double` **`temp`** `[INPUT, MANDATORY, default=no default value]`

A temperature in kelvin.

`Unit` **`waveUnit`** `[INPUT, MANDATORY, default=no default value]`

The unit of the wave data. The unit must be a wavenumber, wavelength, or frequency; any other type will throw an exception.

Return

Double1d

spectral radiance in units of $W/m^2/sr/Hz$.

`Double1d` **`planck (Double1d wave, double temp)`**

Computes the Planck function for the given wave data and temperature.

See the other static Java method. The unit of the wave data is assumed to be Hz.

Arguments

`Double1d` **`wave`** `[INPUT, MANDATORY, default=no default value]`

Wave data. Must not contain negative values.

`double` **`temp`** `[INPUT, MANDATORY, default=no default value]`

A temperature in kelvin.

Return

Double1d

spectral radiance in units of $W/m^2/sr/Hz$.

`double` **`calc (double wave)`**

Computes the Planck function for the given wave data using a temperature

double **calc (double wave)**

and wave unit set when the Planck function object was constructed. See constructor.

Argument

`double` **`wave`** `[INPUT, MANDATORY, default=no default value]`

Wave data. Must not contain negative values.

Return

double

spectral radiance in units of $W/m^2/sr/Hz$.

See also

- Developers Manual: herschel.ia.toolbox.astro.Planck

History

- `2011-06-06 - DS`: First version.
- `2011-09-13 - JDS`: Moved to ia_toolbox_astro

1.298. pointHistoryDisplay

Full Name:	herschel.ia.toolbox.pointing.PointHistoryDisplayTask
Alias:	pointHistoryDisplay
Type:	Java Task -
Import:	from herschel.ia.toolbox.pointing import PointHistoryDisplayTask
Category	Astronomical utilities

Description

Task for displaying graphs of pointing information.

This takes a PointingProduct and produces graphs of the data contained in the product.

Example

Example 1: Display pointing where obs is an Observation

```
aux = obs.getAuxiliary()
pp = aux.getPointing()
graphs = String1d(["RAC-Time", "RAG-Time"])
pointHistoryDisplay(pp, graphs, "mosaic")
```

API Summary

Jython Syntax

```
pointHistoryDisplay(&lt;pointingProduct&gt; [, &lt;plot-
Pairs&gt;=["RAG-Time"], &lt;layout&gt;="mosaic"])
```

Properties

PointingProduct **PointingProduct** [INPUT, MANDATORY, default=null]

String1d **plotPairs** [INPUT, MANDATORY, default=null]

String **layout** [INPUT, MANDATORY, default="mosaic"]

API details

Properties

PointingProduct PointingProduct [INPUT, MANDATORY, default=null]

PointingProduct

String1d plotPairs [INPUT, MANDATORY, default=null]

String1d containing strings specifying parameter pairs to be plotted. These are pairs of Strings separated by a hyphen.

Allowed Strings for each element of the pairs are:

- "Time"
- "RAC" (Commanded RA)

String1d plotPairs [INPUT, MANDATORY, default=null]

- "RAG" (Gyro-propagated RA)
- "RAF" (Filtered RA)
- "DecC" (Commanded Dec)
- "DecG" (Gyro-propagated RA)
- "DecF" (Filtered RA)
- "PAC" (Commanded Position angle)
- "PAG" (Gyro-propagated position angle)
- "PAF" (Filtered position angle)
- "AngVel1" (first angular velocity value)
- "AngVel2" (second angular velocity value)
- "AngVel3" (third angular velocity value)

At least one string (containing an X-Y pair) should be in this parameter.

String layout [INPUT, MANDATORY, default="mosaic"]

String with value "mosaic" or "overlay" to specify how multiple plots should be organised.

See also

- Developers Manual: herschel.ia.toolbox.pointing.PointHistoryDisplayTask

History

- 2010-11-19 - JDS: Fixed documentation

1.299. PoissonErrorDistribution

Full Name:	herschel.ia.numeric.toolbox.fit.sample.PoissonErrorDistribution
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit.sample import PoissonErrorDistribution
Category	Mathematics/Fitting

Description

PoissonErrorDistribution models the fitting error using a Poisson distribution.

Poisson distr: $f(n,x) = x^n e^{-x} / n!$

where n are the measured counts and x are the expected counts

This distribution is used in counting processes.

See example

For a simple way to choose the distribution in the NestedSampler use NestedSampler.setPoissonDistribution().

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.sample.PoissonErrorDistribution

1.300. Polygon

Full Name:	herschel.ia.gui.image.Polygon
Type:	Java Class -
Import:	from herschel.ia.gui.image import Polygon
Category	Images/Display

Description

A polygon shape.

API Summary

Constructors
`Polygon` The construction of a new Polygon without coordinates.
`Polygon (double[] coords)` The construction of a new Polygon with the given vertices in the format [x0, y0, x1, y1,...].
`Polygon (double x, double y)` The construction of a new Polygon with a single starting point.
`Polygon (int size)` The construction of a new Polygon with space for the given number of vertices.

Method
`boolean` **`contains (double x, double y)`** Checks whether the point with the given pixel coordinates is inside this Polygon.

Miscellaneous

This class extends the existing diva.util.java2d.Polygon2D.Double, because the contains() method was implemented incorrectly there.

API Details

Constructors

`Polygon`
The construction of a new Polygon without coordinates.

`Polygon (double[] coords)`
The construction of a new Polygon with the given vertices in the format [x0, y0, x1, y1,...]. **Argument** `double[] coords [INPUT, MANDATORY, default=no default value]` The coordinates of the vertices.

`Polygon (double x, double y)`
The construction of a new Polygon with a single starting point.

`Polygon (double x, double y)`

Arguments

`double` **`x`** `[INPUT, MANDATORY, default=no default value]`

The x-pixel-coordinate of the starting point.

`double` **`y`** `[INPUT, MANDATORY, default=no default value]`

The y-pixel-coordinate of the starting point.

`Polygon (int size)`

The construction of a new Polygon with space for the given number of vertices.

Argument

`int` **`size`** `[INPUT, MANDATORY, default=no default value]`

The number of vertices.

Method

`boolean` **`contains (double x, double y)`**

Checks whether the point with the given pixel coordinates is inside this Polygon.

Arguments

`double` **`x`** `[INPUT, MANDATORY, default=no default value]`

The x-pixel-coordinate.

`double` **`y`** `[INPUT, MANDATORY, default=no default value]`

The y-pixel-coordinate.

Return

boolean

Return true if the point with the given pixel coordinates is inside this Polygon; false otherwise.

See also

- Developers Manual: herschel.ia.gui.image.Polygon

1.301. polygonHistogram

Full Name:	herschel.ia.toolbox.image.PolygonHistogramTask
Alias:	polygonHistogram
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import PolygonHistogramTask
Category	Images/Display

Description

This is a task to make a histogram of a region of interest on am image which is bounded by a polygon.

This is a task to make a histogram of a region of interest on an image which is bounded by a polygon. You must specify the number of bins for the histogram. By default the cut levels of the image will be used to construct the histogram, but it is also possible for you to specify the low and high cut level for the histogram. To define the position of the polygon, you must specify the pixel or sky coordinates of the its corners.

Examples

Example 1: Example of how you can use the polygonHistogram by specifying the corners in pixel coordinates ;

```
pyEdges = Double1d([95.80, 150.61, 129.13, 232.83, 264.68, 207.65, 252.09,
 109.13])
histogram = polygonHistogram(image = myImage,lowCut = 100.0, highCut = 150.0,
 bins = 100,
                              edgesPixel = pyedges)
```

Example 2: Example of how you can use the polygonHistogram by specifying the corners in sky coordinates :

```
pyEdges = String1d(["05:47:51.56", "-51:04:36.74", "05:47:37.37",
 "-510:59:08.06", "05:46:39.92",
                     "-51:00:48.54"])
histogram = polygonHistogram(image = myImage,lowCut = 100.0, highCut = 150.0,
 bins = 100,
                              edgesSky = pyedges)
```

API Summary

Properties
Image **image** [INPUT, MANDATORY, default=None]
Double **lowCut** [INPUT, OPTIONAL, default=None]
Double **highCut** [INPUT, OPTIONAL, default=None]
Integer **bins** [INPUT, MANDATORY, default=2000]
Double1d **edgesPixel** [INPUT, OPTIONAL, default=None]
String1d **edgesSky** [INPUT, OPTIONAL, default=None]
PolygonHistogramProduct **histogram** [OUTPUT, MANDATORY, default=None]

API details

Properties

Image image [INPUT, MANDATORY, default=None]

This is the input image.

Double lowCut [INPUT, OPTIONAL, default=None]

This is the low cut level you want to use for the histogram. If this is not specified, the image cut levels will be used.

Double highCut [INPUT, OPTIONAL, default=None]

This is the high cut level you want to use for the histogram. If this is not specified, the image cut levels will be used.

Integer bins [INPUT, MANDATORY, default=2000]

This is the number of bins in the histogram.

Double1d edgesPixel [INPUT, OPTIONAL, default=None]

This is a list with the pixel coordinates of the corners of the polygon : [x1, y1, x2, y2,...].

String1d edgesSky [INPUT, OPTIONAL, default=None]

This is a list with the sky coordinates of the corners of the polygon : [RA1, Dec1, RA2, Dec2,...]. Using this only makes sense if the input image has a valid Wcs attached to it. The input format for the right ascension is "hh mm ss.s", "hh:mm:ss.s" or "hh.hhh", and "[+/-]dd mm ss.s", "[+/-]dd:mm:ss.s" or "[+/-]dd.ddd" for the declination.

PolygonHistogramProduct histogram [OUTPUT, MANDATORY, default=None]

This is the resulting histogram.

See also

- Developers Manual: herschel.ia.toolbox.image.PolygonHistogramTask

1.302. PolygonHistogramProduct

Full Name:	herschel.ia.dataset.image.PolygonHistogramProduct
Type:	Java Class -
Import:	from herschel.ia.dataset.image import PolygonHistogramProduct
Category	Images/Display

Description

A class to deal with the results of a polygon histogram.

API Summary

Constructor
PolygonHistogramProduct The constructor of a new PolygonHistogramProduct.

Methods
setEdges (Double1d edgesPixels) Sets the edges for this PolygonHistogramProduct to the given pixel coordinates.
setEdges (Double1d edgesPixels, String1d edgesSky) Sets the edges for this PolygonHistogramProduct to the given list of pixel and sky coordinates.
CompositeDataset **getEdges** Returns the edges for this PolygonHistogramProduct.
int **getNbOfEdges** Returns the number of edges for this PolygonHistogramProduct.
TableDataset **getEdgesPixelCoordinates** Returns the edges for this PolygonHistogramProduct in pixel coordinates.
Double2d **getEdgesPixelCoordinatesDouble2d** Returns the pixel coordinates of the edges as Double2d.
TableDataset **getEdgesSkyCoordinates** Returns the edges for this PolygonHistogramProduct in sky coordinates.

API Details

Constructor

PolygonHistogramProduct
The constructor of a new PolygonHistogramProduct.

Methods

setEdges (Double1d edgesPixels)
Sets the edges for this PolygonHistogramProduct to the given pixel coordinates. **Argument**

setEdges (Double1d edgesPixels)

`Double1d` **`edgesPixels`** `[INPUT, MANDATORY, default=no default value]`

The pixel coordinates to set as the pixel coordinates of the edges, as Double1d

setEdges (Double1d edgesPixels, String1d edgesSky)

Sets the edges for this PolygonHistogramProduct to the given list of pixel and sky coordinates.

Arguments

`Double1d` **`edgesPixels`** `[INPUT, MANDATORY, default=no default value]`

The pixel coordinates to set as the pixel coordinates of the edges, as Double1d

`String1d` **`edgesSky`** `[INPUT, MANDATORY, default=no default value]`

The sky coordinates to set as the pixel coordinates of the edges, as String1d

CompositeDataset **getEdges**

Returns the edges for this PolygonHistogramProduct.

Return

CompositeDataset

Returns the edges for this PolygonHistogramProduct.

int **getNbOfEdges**

Returns the number of edges for this PolygonHistogramProduct.

Return

int

Returns the number of edges for this PolygonHistogramTask.

TableDataset **getEdgesPixelCoordinates**

Returns the edges for this PolygonHistogramProduct in pixel coordinates.

Return

TableDataset

Returns the edges for this PolygonHistogramProduct in pixel coordinates.

Double2d **getEdgesPixelCoordinatesDouble2d**

Returns the pixel coordinates of the edges as Double2d.

Return

Double2d

Returns the pixel coordinates of the edges as a Double2d.

TableDataset **getEdgesSkyCoordinates**

Returns the edges for this PolygonHistogramProduct in sky coordinates.

Return

TableDataset

Returns the edges for this Polygon in sky coordinates.

See also

- Developers Manual: herschel.ia.dataset.image.PolygonHistogramProduct

1.303. PolygonIntersect

Full Name:	herschel.ia.toolbox.image.PolygonIntersect
Type:	Java Class -
Import:	from herschel.ia.toolbox.image import PolygonIntersect
Category	Images/Display

Description

This is a class to calculate the overlap between polygons.

This is a class to calculate the overlap between polygons. This algorithm is based on the algorithm on http://cap-lore.com/MathPhys/IP/ (adapted 9-May-2006 by Lagado) and computes the integral over the plane, of the product of the winding numbers of two polygons. If neither polygon is self intersecting and they have the same orientation, then this integral is the area of their intersection.

API Summary

Method
double **getIntersectionArea** (Point2D[] polygon1Points, Point2D[] polygon2Points)
Returns the area of intersection of two polygons.

API Details

Method

double getIntersectionArea (Point2D[] polygon1Points, Point2D[] polygon2Points)

Returns the area of intersection of two polygons.

Returns the area of intersection of two given polygons.

Arguments

`Point2D[] polygon1Points [INPUT, MANDATORY, default=no default value]`

The vertices of the first polygon in pixel coordinates.

`Point2D[] polygon2Points [INPUT, MANDATORY, default=no default value]`

The vertices of the second polygon in pixel coordinates.

Return

double

The area of intersection of two given polygons.

See also

- Developers Manual: herschel.ia.toolbox.image.PolygonIntersect

1.304. PolynomialDynamicModel

Full Name:	herschel.ia.numeric.toolbox.fit.sample.PolynomialDynamicModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit.sample import PolynomialDynamicModel
Category	Mathematics/Fitting

Description

General polynomial model of an adaptable degree.

$$f(x:p) = \sum p_k * x^k$$

where the sum is over k running from 0 to degree (inclusive).

It is a linear model.

See example

Example

Example 1: PolynomialModel

```
poly = PolynomialDynamicModel( )          # polynomial with unknown degree
ran = MoreRandom()
poly.grow( ran )                          # starts at degree = 0, npar = 1
poly.grow( ran )                          # each grow() adds 1
poly.grow( ran )
poly.grow( ran )
print poly.getNumberOfParameters()         # 5
poly.shrink( ran )                         # shrink() deletes 1 degree
print poly.getNumberOfParameters()         # 4
```

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.sample.PolynomialDynamicModel

1.305. Polynomial

Full Name:	herschel.ia.numeric.toolbox.basic.Polynomial
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Polynomial
Category	Arrays and datasets

Description

Returns the value of a polynomial, given its coefficients, at a series of positions.

The coefficients of the polynomial are passed as an arrays, starting from degree zero. For example, the coefficients array for the polynomial $y = 3x^2 + 2.4x - 5.5$ is [-5.5, 2.4, 3.0].

Example

Example 1: Apply Polynomial to a Int1d

```
# Constant value y = 1
print Polynomial(Double1d([1]))(Double1d.range(5)) # [1.0,1.0,1.0,1.0,1.0]
# y = x
print Polynomial(Double1d([0,1]))(Double1d.range(5)) # [0.0,1.0,2.0,3.0,4.0]
# y = x^2
print Polynomial(Double1d([0,0,1]))(Double1d.range(5)) # [0.0,1.0,4.0,9.0,16.0]
# Alternative syntax:
print Double1d.range(5).apply(Polynomial(Double1d([1]))) #
 [1.0,1.0,1.0,1.0,1.0]
print Double1d.range(5).apply(Polynomial(Double1d([0,1]))) #
 [0.0,1.0,2.0,3.0,4.0]
print Double1d.range(5).apply(Polynomial(Double1d([0,0,1]))) #
 [0.0,1.0,4.0,9.0,16.0]
```

API Summary

Jython Syntax

```
&lt;y&gt;=Polynomial(&lt;coefficients&gt;)(&lt;x&gt;)
```

Properties

Array **x** [INPUT, MANDATORY, default=no default value]

Double array **coefficients** [INPUT, MANDATORY, default=no default value]

Array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Array x [INPUT, MANDATORY, default=no default value]

The positions at which to compute the polynomial.

Double array coefficients [INPUT, MANDATORY, default=no default value]

The coefficients of the polynomial.

Array y [OUTPUT, MANDATORY, default=no default value]
The value of the polynomial at each of the given positions.

See also

- Developers Manual: herschel.ia.numeric.toolbox.basic.Polynomial

1.306. PolynomialModel

Full Name:	herschel.ia.numeric.toolbox.fit.PolynomialModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import PolynomialModel
Category	Mathematics/Fitting

Description

General polynomial model of arbitrary degree.

$$f(x:p) = \sum p_k * x^k$$

where the sum is over k running from 0 to degree (inclusive).

It is a linear model.

See example

Example

Example 1: PolynomialModel

```
poly = PolynomialModel( 3 )            # 3rd degree polynomial
print poly.getNumberOfParameters()        # 4
```

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.PolynomialModel

1.307. PolySurfaceModel

Full Name:	herschel.ia.numeric.toolbox.fit.PolySurfaceModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import PolySurfaceModel
Category	Mathematics/Fitting

Description

General polynomial surface model of arbitrary degree.

$$f(x,y:p) = \sum_d (\sum_k p_n * x^k * y^{d-k})$$

where the first sum is over d running from 0 to degree (inclusive) and the second sum is over k running from 0 to d (inclusive). The index n is just incrementing, making all p's different.

It is a 2-dimensional linear model.

Examples of the generated polynomials:

Table 1.4. Polynomial table.

degree	nr param	polynomial	equivalent to
0	1	p_0	BinomialModel(0)
1	3	$p_0 + p_1 y + p_2 x$	BinomialModel(0) + BinomialModel(1)
2	6	$p_0 + p_1 y + p_2 x + p_3 y^2 + p_4 y x + p_5 x^2$	BinomialModel(0) + BinomialModel(1) + BinomialModel(2)
3	10	$p_0 + p_1 y + p_2 x + p_3 y^2 + p_4 y x + p_5 y^2 + p_6 y^3 + p_7 y^2 x + p_8 y x^2 + p_9 x^3$	BinomialModel(0) + BinomialModel(1) + BinomialModel(2) + BinomialModel(3)

Example

Example 1: PolySurfaceModel

```
poly = PolySurfaceModel( 3 )         # 3rd degree polynomial
print poly.getNumberOfParameters()       # 10
```

See also

- BinomialModel
- Developers Manual: herschel.ia.numeric.toolbox.fit.PolySurfaceModel

1.308. PoolManager

Full Name:	herschel.ia.pal.PoolManager
Type:	Java Class - 
Import:	from herschel.ia.pal import PoolManager
Category	Data access

Description

The PoolManager provides the means to reference Product Pools without

having to hard-code the type of the pool.

Example

Example 1: Get the pool by name

```
pool = PoolManager.getInstance().get("jexample")
```

See also

- Developers Manual: herschel.ia.pal.PoolManager

1.309. PositionList

Full Name:	herschel.ia.dataset.image.PositionList
Type:	Java Class -
Import:	from herschel.ia.dataset.image import PositionList
Category	Images/Display

Description

Position list.

A list of positions.

API Summary

Methods
double **getRa** (int row) Returns the RA at a given row.
double **getDec** (int row) Returns the Dec at a given row.
int **getSize** Returns the number of positions.

API Details

Methods

double getRa (int row)

Returns the RA at a given row.

Returns the RA for the position for this PositionList at the given row in the list.

Argument

`int` **`row`** `[INPUT, MANDATORY, default=no default value]`

The row number, as int

Return

double

Returns the RA for the position for this PositionList at the given row in the list.

double getDec (int row)

Returns the Dec at a given row.

Returns the Dec for the position for this PositionList at the given row in the list.

Argument

`int` **`row`** `[INPUT, MANDATORY, default=no default value]`

The row number, as int

Return

***double* getDec (int row)**

double

Returns the Dec for the position for this PositionList at the given row in the list.

***int* getSize**

Returns the number of positions.

Returns the number of positions for this PositionList.

Return

int

Returns the number of positions for this PositionList.

See also

- Developers Manual: herschel.ia.dataset.image.PositionList

1.310. PowerLawModel

Full Name:	herschel.ia.numeric.toolbox.fit.PowerLawModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import PowerLawModel
Category	Mathematics/Fitting

Description

General powerlaw model of arbitrary degree.

$$f(x:p) = p\_0 * (x - p\_1)^{p\_2}$$

p_0 = amplitide p_1 = x-shift p_2 = power

The parameters are initialized at {1.0, 0.0, 1.0}.

Example

Example 1: PowerLawModel

```
pl = PowerLawModel()
print pl.getNumberOfParameters()      # 3
```

Limitations

Note that the term (x - p_1) needs to be divided by a factor 1.0 in the same units as the x, to get the overall units of f(x:p) right. The factor is omitted as it does not contribute in the calculations.

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.PowerLawModel

1.311. PowerModel

Full Name:	herschel.ia.numeric.toolbox.fit.PowerModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import PowerModel
Category	Mathematics/Fitting

Description

General power model of arbitrary degree.

$$f(x:p) = p * x^{a}$$

a is an double (positive or negative).

Example

Example 1: PowerModel

```
pwr = PowerModel( -1 )
print pwr.getNumberOfParameters()       # 1
```

Limitations

Note that the x-term needs to be divided by a factor 1.0 in the same units as the x, to get the overall units of f(x:p) right. The factor is omitted as it does not contribute in the calculations.

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.PowerModel

1.312. PowerSpectrum

Full Name:	herschel.ia.toolbox.astro.PowerSpectrum
Alias:	PowerSpectrum
Type:	Java Class -
Import:	from herschel.ia.toolbox.astro import PowerSpectrum
Category	Mathematics/Signal processing

Description

A PowerSpectrum is a normalised Fourier Transform of the time series.

NOTE: the 0 frequency is removed from the result

Example

Example 1: Jython example

```
from herschel.ia.toolbox.astro import PowerSpectrum
flimit = 0.1
sigma = 0.4
deglitch= 1
#for some table
pw_table = PowerSpectrum.getPowerSpectrum(flimit, sigma, deglitch, table)
```

API Summary

Jython Syntax

```
&lt;tableOut&gt; = PowerSpectrum.getPowerSpectrum([&lt;flimit&gt;,
&lt;sigma&gt;= 0.4, &lt;deglitch&gt;=True ,] &lt;table&gt; [,
&lt;timeColumn&gt;])
```

Properties

Double **flimit** [INPUT, OPTIONAL, default=(scanRate / #rows)]
Double **sigma** [INPUT, OPTIONAL, default=0.4]
Boolean **deglitch** [INPUT, OPTIONAL, default=true]
TableDataset **table** [INPUT, MANDATORY, default=no default value]
Column **timeColumn** [INPUT, OPTIONAL, default=obtained from the table]
TableDataset **tableOut** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Double flimit [INPUT, OPTIONAL, default=(scanRate / #rows)]

The cutoff frequency

Double sigma [INPUT, OPTIONAL, default=0.4]

The width

Boolean deglitch [INPUT, OPTIONAL, default=true]
Remove glitches

TableDataset table [INPUT, MANDATORY, default=no default value]
TableDataset whose powerspectrum is to be calculated

Column timeColumn [INPUT, OPTIONAL, default=obtained from the table]
Double1d arrays only. Input is modified during the calculation.

TableDataset tableOut [OUTPUT, MANDATORY, default=no default value]
Returns a power spectrum in the form of TableDataset

See also

- Developers Manual: herschel.ia.toolbox.astro.PowerSpectrum

History

- 2008-02-22 - LZ: first version, from herschel.ia.numeric.toolbox.xform
- 2011-01-24 - JDS: JExample
- 2012-03-15 - JDS: URM entry
- 2012-11-25 - JDS: Reviewed

1.313. Pow

Full Name:	herschel.ia.numeric.toolbox.basic.Pow
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Pow
Category	Mathematics/General functions

Description

Raises a number or the elements of an array to the specified power.

If the input is an array, the output is an array of the same size, with elements raised to the specified power instead of the original elements.

Example

Example 1: Applying Pow to a Double1d

```
x = Double1d([1,2,3,4])
print Pow(2)(x) # [1.0,4.0,9.0,16.0]
```

API Summary

Jython Syntax
<y> = Pow(<power>)(<x>)

Properties
Number or array **x** [INPUT, MANDATORY, default=no default value]
Number **power** [INPUT, MANDATORY, default=no default value]
Number or array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Number or array x [INPUT, MANDATORY, default=no default value]

The number or array to raise to the specified power. Complex numbers or arrays are not allowed.

Number power [INPUT, MANDATORY, default=no default value]

The power to which to raise the input value or values.

Number or array y [OUTPUT, MANDATORY, default=no default value]

The input value or values raised to the specified power.

See also

- SQUARE
- SQRT

- Developers Manual: herschel.ia.numeric.toolbox.basic.Pow

1.314. PreviousInterpolator

Full Name:	herschel.ia.numeric.toolbox.interp.PreviousInterpolator
Alias:	PreviousInterpolator
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.interp import PreviousInterpolator
Category	Mathematics/Interpolation

Description

Creates an linear interpolation function from a set of knots (x,y), that can be applied to numeric arrays of rank 1.

A lookup-previous interpolator function.

Example

Example 1: Create and apply a PreviousInterpolator on a Double1d

```
from herschel.ia.numeric.toolbox.interp import PreviousInterpolator
# create some knots
x=Double1d.range(10) # [0.0,1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0]
f=PreviousInterpolator(x,SQUARE(x))
u=Double1d([1,1.1,2,2.6,3,3.9])
print f(u) # [1.0,1.0,4.0,4.0,9.0,9.0]
```

API Summary

Jython Syntax

<f>=PreviousInterpolator(<x>,<y>[,allowExtrapolation])

Properties

Double1d **x** [INPUT, MANDATORY, default=no default value]

Double1d **y** [INPUT, OPTIONAL, default=false]

boolean **allowExtrapolation** [INPUT, OPTIONAL, default=false]

API details

Properties

Double1d x [INPUT, MANDATORY, default=no default value]

The knots are Double1d

Double1d y [INPUT, OPTIONAL, default=false]

The knots are Double1d

boolean allowExtrapolation [INPUT, OPTIONAL, default=false]

Extrapolation is not allowed by default. Boolean.TRUE allows extrapolation.

See also

- CubicSplineInterpolator
- LinearInterpolator
- Developers Manual: herschel.ia.numeric.toolbox.interp.PreviousInterpolator

1.315. PrfGaussian

Full Name:	herschel.ia.toolbox.srcext.prf.PrfGaussian
Type:	Java Class -
Import:	from herschel.ia.toolbox.srcext.prf import PrfGaussian
Category	Astronomical utilities/Source extraction

Description

A representation of a Gaussian point response function as an Image.

The value in each pixel is the average of a two-dimensional Gaussian over that region. (The average of a Gaussian function between fixed limits is calculated using the error function.)

More detail about the methods available are in the Developer's Reference Manual.

Examples

Example 1: Create image with point sources at pixel positions

```
# FWHM = 1.0 pixels
prf = PrfGaussian([100, 100], 1.0)
#
# Source at (10, 20) with peak of 2.0
sourceImage = prf.of(10, 20, 2.0)
```

Example 2: Create image with point sources at world coordinates

```
# image = SimpleImage with WCS defined
# FWHM = 18.0 arcsec
prf = PrfGaussian(image, 18.0)
#
# Source at (ra, dec) = (180, 30) degrees with peak flux 0.1
sourceImage = prf.of(180, 30, 0.1)
```

See also

- PrfImage
- Developers Manual: herschel.ia.toolbox.srcext.prf.PrfGaussian

1.316. PrfImage

Full Name:	herschel.ia.toolbox.srcext.prf.PrfImage
Type:	Java Class -
Import:	from herschel.ia.toolbox.srcext.prf import PrfImage
Category	Astronomical utilities/Source extraction

Description

A representation of a point response function based on an image.

When a PrfImage is created, an image must be provided, which is the model for a the point response function. This image must have an odd number of pixels in both dimensions, and should peak in the central pixel. An image can be retrieved using the .of(y [row], x [col]) method, which retieves an image with the centre at (row, col), which needn't be integers. The dimensions of the output image are either the same as those of the input image, or can be different, if specified in the constructor PrfImage(...), or if specified using .setDimensions([nRows, nColumns]). The image can be rotated using .rotate(angleDegreesAnticlockwise). If the PrfImage(image, wcs, [nRows, nCols]) constructor is used, resulution of the image obtained by .of(...) can be different. The resolution of the input image is taken from image.wcs.cdelt2, and the resolution of the output image, from .of(...), is given by wcs.cdelt2. The amplitude of the image can be adjusted using .multiply(factor).

Sub-pixel interpolation of the PRF is performed in each direction separately, assuming an underlying quadratic function, and using three adjacent pixels to find the parameters of that quadratic function. That quadratic function is then integrated between the edges of the pixel to obtain the new value. This is done first in the horizontal and then the vertical directions, and then vice versa, and the two results are averaged.

Note that coordinates are always interpreted as (row, col), and never as (ra, dec).

More detail about the methods available are in the Developer's Reference Manual.

Example

Example 1: Create image with point sources at pixel positions

```
prf = PrfImage(Double2d([[0, 0, 0], [0, 1, 0], [0, 0, 0]]))
#
# Source at (row, col) = (1.2, 0.7)
sourceImage = prf.of(1.2, 0.7)
```

See also

- PrfGaussian
- Developers Manual: herschel.ia.toolbox.srcext.prf.PrfImage

1.317. PriorList

Full Name:	herschel.ia.numeric.toolbox.fit.PriorList
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import PriorList
Category	Mathematics/Fitting

Description

PriorList is a ArrayList of AbstractPrior elements.

There is one item in the list for each of the model's parameters. However if the list is shorter than the number of parameters, it repeats info from the last prior in the list.

PriorList contains several methods to set and get limits on all parameters.

Example

Example 1: PriorList

```
priorlist = PriorList( UniformPrior(), 3 )        # list of 3 priors
print priorlist.hasLimits()                        # False
uniprior = UniformPrior( Double1d([-10,20]) )    # prior with limits
priorlist = PriorList( uniprior, 4 )             # list of 4 priors
priorlist.setLimits( 2, Double1d([-100,100]) )    # reset limits on prior 2
pl2 = priorlist.get( Range( 2, 4 ) )             # select last 2
print pl2.getLowLimits()                          # [-100,-10]
print pl2.getHighLimits()                         # [100,20]
pl2.add( UniformPrior() )
pl2.add( ExponentialPrior() )
pl2.add( JeffreysPrior() )                         # add 3 more priors to list
```

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.PriorList

1.318. ProductBrowserTools

Full Name:	herschel.ia.pal.browser.ProductBrowserTools
Type:	Java Class -
Import:	from herschel.ia.pal.browser import ProductBrowserTools

Description

Open a dialog to manually select a subset of rows in a PAL browser StorageResult.

This method can be included in advanced scripts that need user interaction.

Example

Example 1: Usage

```
from herschel.ia.pal.browser import ProductBrowserTools
result = ProductStorage([PoolManager.getPool('mypool')])
    .select(herschel.ia.pal.query.MetaQuery(herschel.ia.obs.ObservationContext,
 "p", "1"))
userSelection = ProductBrowserTools.filterStorageResult(result, TRUE)
```

API Summary

Method
StorageResult **filterStorageResult** (StorageResult inputStorageResult, Boolean selectMultipleRows, ResultTableFormat tableFormat)
Open a modal dialog to select a subset of rows of a StorageResult.

API Details

Method

***StorageResult* filterStorageResult (StorageResult inputStorageResult, Boolean selectMultipleRows, ResultTableFormat tableFormat)**

Open a modal dialog to select a subset of rows of a StorageResult.

The result is returned as a new StorageResult object.

Arguments

`StorageResult` **`inputStorageResult`** `[INPUT, MANDATORY, default=no default value]`

The StorageResult (e.g. coming from a previous query to a ProductPool) that contains the rows to be selected from.

`Boolean` **`selectMultipleRows`** `[INPUT, OPTIONAL, default=True]`

Is the user allowed to select multiple rows. If set to false the result will contain 0 or 1 row.

`ResultTableFormat` **`tableFormat`** `[INPUT, OPTIONAL, default=no default value]`

A generic default table layout Expert parameter. The ResultTableFormat defines the type and order of columns to be displayed in the result table. The following example explains how the default format is built.

StorageResult **filterStorageResult (StorageResult inputStorageResult, Boolean selectMultipleRows, ResultTableFormat tableFormat)**

```
ResultTableFormat format = new ResultTableFormat();
QueryParameterRegistry qpr = QueryParameterRegistry.getInstance();
format.addColumn(new MetadataColumn(qpr.getQueryParameter("odNumber")));
format.addColumn(new MetadataColumn(qpr.getQueryParameter("obsid")));
format.addColumn(new StorageTagColumn());
format.addColumn(new ProductRefDescriptorsColumn(Descriptor.VERSION));
format.addColumn(new MetadataColumn(qpr.getQueryParameter("object")));
format.addColumn(new ProductRefDescriptorsColumn(Descriptor.TOTAL_SIZE));
format.addColumn(new MetadataColumn(qpr.getQueryParameter("instrument")));
format.addColumn(new MetadataColumn(qpr.getQueryParameter("aot")));
format.addColumn(new MetadataColumn(qpr.getQueryParameter("obsMode")));
format.addColumn(new MetadataColumn(qpr.getQueryParameter("startDate")));
format.addColumn(new MetadataColumn(qpr.getQueryParameter("proposal")));
format.addColumn(new MetadataColumn(qpr.getQueryParameter("aorLabel")));
```

Alternatively you can use method herschel.ia.pal.browser.v2.config.BrowserTemplateRegistry.addTableFormatTemplate()

```
from herschel.ia.pal.browser.v2.config.BrowserTemplateRegistry import
addTableFormatTemplate;
from herschel.ia.pal.browser.v2.table.model.column import
ProductRefDescriptorsColumn;
from
herschel.ia.pal.browser.v2.table.model.column.ProductRefDescriptorsColumn
import Descriptor;
# add the table format for the observation context layout.
format = addTableFormatTemplate(
  'observationContextResultTableFormatBuilder',
  'ObservationContext Layout',
  ['obsid', 'odNumber', 'startDate', 'aot', 'instrument', 'obsMode',
'object', 'proposal', 'aorLabel',

ProductRefDescriptorsColumn(ProductRefDescriptorsColumn.Descriptor.VERSION)]);
```

Return

StorageResult

The selected data is returned in a StorageResult. This StorageResult is bound to the same ProductStorage as the inputData. The StorageResult will be empty if the user selected no data. The StorageResult will be null if the user cancelled the selection (i.e. by not clicking on "ok").

See also

- Developers Manual: herschel.ia.pal.browser.ProductBrowserTools

1.319. PRODUCT

Full Name:	herschel.ia.numeric.toolbox.basic.Product
Alias:	PRODUCT
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Product
Category	Arrays and datasets/Manipulation

Description

Returns the product of all the elements of an array.

This function can return a global product for the whole array, or compute a product along a given dimension of a multidimensional array if the `dim` parameter is specified. For example, you could multiply the elements on each row, or column, of a two-dimensional array. See also the example below.

Example

Example 1: Apply PRODUCT on a Int2d

```
x = Int2d( [ [1,2], [-1,3] ])
print PRODUCT(x)    # -6
print PRODUCT(x, 0) # [-1, 6]
print PRODUCT(x, 1) # [ 2,-3]
```

API Summary

Jython Syntax

```
&lt;y&gt;=PRODUCT(&lt;x&gt;, [,&lt;dim&gt;])
```

Properties

Array **x** [INPUT, MANDATORY, default=no default value]

Integer **dim** [INPUT, OPTIONAL, default=no default value]

Number or array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Array x [INPUT, MANDATORY, default=no default value]

The array whose elements are to be multiplied.

Integer dim [INPUT, OPTIONAL, default=no default value]

The dimension along which to compute the product.

Number or array y [OUTPUT, MANDATORY, default=no default value]

The sum of all the elements of the array, or the sums along a given dimension if the dim parameter is specified.

See also

- SUM
- Developers Manual: herschel.ia.numeric.toolbox.basic.Product

1.320. ProductRef

Full Name:	herschel.ia.pal.ProductRef
Type:	Java Class -
Import:	from herschel.ia.pal import ProductRef
Category	Data access

Description

A ProductRef provides a reference to a product that is held in a product storage or to a product in memory.

Typically a ProductRef is returned by the load, save and select methods of a ProductStorage. A Product reference is providing a mechanism to inspect the meta data of a stored product without loading the complete product into memory.

Example

Example 1: Usage of a product reference

```
p = Product(creator="me")
ref = storage.save(p)
print ref.type # herschel.ia.dataset.Product
print ref.urn # urn:simple.default:herschel.ia.dataset.Product:23674
print ref.meta['creator'] # me
print ref.product.creator # me (product loaded into memory!)
```

API Summary

Methods
getProduct Returns the Product class to which this Product reference is
getType Returns the Product class to which this Product reference is

API Details

Methods

getProduct

Returns the Product class to which this Product reference is

pointing to.

getType

Returns the Product class to which this Product reference is

pointing to.

See also

- Developers Manual: herschel.ia.pal.ProductRef

1.321. ProductStorage

Full Name:	herschel.ia.pal.ProductStorage
Type:	Java Class -
Import:	from herschel.ia.pal import ProductStorage
Category	Data access

Description

The ProductStorage is a storage mechanism to provide read, write and

query on Products stored in registered Product Pools.

Examples

Example 1: Creating and registering

```
myStore=ProductStorage()
myStore.register(LocalStoreFactory.getStore("foo")) # the foo lstore pool
```

Example 2: Loading a product from this store

```
ref = myStore.load("urn:foo:herschel.ia.dataset.Product:0" )
```

Example 3: Saving a product

```
ref = myStore.save(product)
```

Example 4: Select products based on a query

```
query = Query("creator='Joe'")
results = myStore.select(query)
```

API Summary

Constructors
ProductStorage Creates an empty ProductStorage with no pools registered.
ProductStorage (ProductPool pool) Creates a ProductStorage with the given pool registered, which
ProductStorage (ProductPool[] pools) Creates a ProductStorage with the given pools registered.
ProductStorage (String poolName) Creates a ProductStorage with the the pools corresponding to the
ProductStorage (String[] poolNames) Creates a ProductStorage with the the pools corresponding to the

Methods
register (ProductPool pool) Registers a ProductPool to the ProductStorage. The first

Methods
register (ProductStorage. The product pools that stores) Chain storage registrations, register storages in storages. The
register (ProductStorage. The product pools that store) Chain storage registrations, register pools in another storage.
register (ProductPool array pools) Registers ProductPools to the ProductStorage. The first
getPools Provides the set of ProductPools registered, in order in which
getWritablePool Returns the writable pool, which is the first registered pool.
Set **getProductClasses** Provides all Product class definitions found in this pool.
remove (String urn) Removes a product of given URN from the storage.
ProductRef **load** (String id) Loads a Product from this store.
ProductRef **save** (Product product) Saves a Product to this store. If the product is a context that
Set **select** (StorageQuery query) Returns a set of references to products that match the specified
Set **select** (StorageQuery query, Set<ProductRef> previous) Returns a set of references to products that match the specified

API Details

Constructors

ProductStorage
Creates an empty ProductStorage with no pools registered.

ProductStorage (ProductPool pool)

Creates a ProductStorage with the given pool registered, which
happens to be the writable one.
Argument
`ProductPool` **`pool`** `[INPUT, MANDATORY, default=no default value]`
The writable pool, as ProductPool
Example
Creating a storage initialized with a pool.
`storage = ProductStorage(pool) # pool is also the writable one`

ProductStorage (ProductPool[] pools)
Creates a ProductStorage with the given pools registered.

ProductStorage (ProductPool[] pools)

The first provided pool is the writable one.

Argument

`ProductPool[] pools [INPUT, MANDATORY, default=no default value]`

The writable pools, as ProductPool[]

Example

Creating a storage initialized with pools.

```
pool1 = PoolManager.getInstance().get("pool1")
pool2 = PoolManager.getInstance().get("pool2")
pool3 = PoolManager.getInstance().get("pool3")
storage = ProductStorage([pool1, pool2, pool3]) # pool1 is the writable one
```

ProductStorage (String poolName)

Creates a ProductStorage with the the pools corresponding to the

given name. This pool is the writable one.

Argument

`String poolName [INPUT, MANDATORY, default=no default value]`

Name of writable pool, as String

Example

Creating a storage initialized with pools.

```
# the following line ...
storage = ProductStorage("pool")
# is equivalent to ...
pool = PoolManager.getInstance().get("pool")
storage = ProductStorage(pool)
```

ProductStorage (String[] poolNames)

Creates a ProductStorage with the the pools corresponding to the

given names registered. The first provided pool is the writable one.

Argument

`String[] poolNames [INPUT, MANDATORY, default=no default value]`

Names of writable pool, as String[]

Example

Creating a storage initialized with pools.

```
# the following line
storage = ProductStorage(["pool1", "pool2", "pool3"])
# is equivalent to
pool1 = PoolManager.getInstance().get("pool1")
pool2 = PoolManager.getInstance().get("pool2")
pool3 = PoolManager.getInstance().get("pool3")
storage = ProductStorage([pool1, pool2, pool3])
```

Methods

register (ProductPool pool)

Registers a ProductPool to the ProductStorage. The first

ProductPool will be the one for which save operations would write products to. The remaining pools registered will be read-only.

register (ProductPool pool)

Argument

`ProductPool` **`pool`** `[INPUT, MANDATORY, default=The product pool that]`

will be added to this product storage.

Example

How to register a product pool

```
storage.register(pool)
```

register (ProductStorage. The product pools that stores)

Chain storage registrations, register storages in storages. The

first ProductPool that registered will be the one for which save operations would write products to. The remaining pools registered will be read-only.

Argument

`ProductStorage. The product pools that` **`stores`** `[INPUT, MANDATORY, default=no default value]`

registered in that storage will be added to this product storage.

Example

How to register a product storage

```
storage.register(store)
```

register (ProductStorage. The product pools that store)

Chain storage registrations, register pools in another storage.

The first ProductPool that registered will be the one for which save operations would write products to. The remaining pools registered will be read-only.

Argument

`ProductStorage. The product pools that` **`store`** `[INPUT, MANDATORY, default=no default value]`

registered in that storage will be added to this product storage.

Example

How to register a product storage

```
storage.register([store1, store2])
```

register (ProductPool array pools)

Registers ProductPools to the ProductStorage. The first

ProductPool will be the one for which save operations would write products to. The remaining pools registered will be read-only.

Argument

`ProductPool array` **`pools`** `[INPUT, MANDATORY, default=The product pools]`

that will be added to this product storage.

Example

How to register many pools at the same time

register (ProductPool array pools)

```
storage.register([pool1, pool2, pool3])
```

getPools

Provides the set of ProductPools registered, in order in which

they were initially registered.

getWritablePool

Returns the writable pool, which is the first registered pool.

Set **getProductClasses**

Provides all Product class definitions found in this pool.

Return

Set

a collection of Product classes

remove (String urn)

Removes a product of given URN from the storage.

Argument

String **urn** [INPUT, MANDATORY, default=The URN to the Product that]

should be removed.

ProductRef **load (String id)**

Loads a Product from this store.

Argument

String **id** [INPUT, MANDATORY, default=no default value]

The URN or tag to the Product, as String

Return

ProductRef

A reference to the Product corresponding to the input URN or tag.

ProductRef **save (Product product)**

Saves a Product to this store. If the product is a context that

has descendents, all those descendents will be copied if they do not already exist in the destination storage (this is a deep-copy or cloning operation).

Argument

Product **product** [INPUT, MANDATORY, default=The product that should be]

saved.

Return

ProductRef

A reference to the Product saved into the store.

Set select (StorageQuery query)

Returns a set of references to products that match the specified

query.

Argument

StorageQuery **query** [INPUT, MANDATORY, default=The query applied to]

this store.

Return

Set

A set of references to Products selected by the query

Set select (StorageQuery query, Set<ProductRef> previous)

Returns a set of references to products that match the specified

query.

Arguments

StorageQuery **query** [INPUT, MANDATORY, default=The query applied to]

this store.

Set<ProductRef> **previous** [INPUT, MANDATORY, default=previous results to]

be refined.

Return

Set

A set of references to Products selected by the query

See also

- Developers Manual: herschel.ia.pal.ProductStorage

1.322. Profile

Full Name:	herschel.ia.dataset.image.Profile
Type:	Java Class -
Import:	from herschel.ia.dataset.image import Profile
Category	Images/Display

Description

A class to deal with the results of profile plotting.

API Summary

Constructor
`Profile` The constructor of a new Profile.

Methods
`setBegin (double beginX, double beginY)` Sets the begin to the given pixel coordinates.
`setBegin (double beginX, double beginY, String beginRA, String beginDec)` Sets the begin to the given pixel and sky coordinates.
`setEnd (double endX, double endY)` Sets the end to the given pixel coordinates.
`setEnd (double endX, double endY, String endRA, String endDec)` Sets the end to the given pixel and sky coordinates.
`setLengthInArcsec (double length)` Sets the length of the straight line.
`setProfile (Double1d profile)` Sets the profile for this Profile to the given profile.
`setUnit (Unit unit)` Sets the unit for this Profile.
`Double1d` **`getBeginPixelCoordinates`** Returns the begin for this Profile in pixel coordinates.
`String1d` **`getBeginSkyCoordinates`** Returns the begin for this Profile in sky coordinates.
`Double1d` **`getEndPixelCoordinates`** Returns the end for this Profile in pixel coordinates.
`String1d` **`getEndSkyCoordinates`** Returns the end for this Profile in sky coordinates.
`double` **`getLength`** Returns the length of the straight line in arcsec.
`Double1d` **`getProfile`**

Methods
Returns the profile for this Profile.
Double1d **getPixels**
Returns the x-axis of this Profile.
Double1d **getArcsec**
Returns the x-axis of this Profile.
String **getUnit**
Returns the unit for this Profile.

API Details

Constructor

Profile
The constructor of a new Profile.

Methods

setBegin (double beginX, double beginY)
Sets the begin to the given pixel coordinates.
Arguments
`double` **beginX** `[INPUT, MANDATORY, default=no default value]`
The x-pixel-coordinate of the begin, as double
`double` **beginY** `[INPUT, MANDATORY, default=no default value]`
The y-pixel-coordinate of the begin, as double

setBegin (double beginX, double beginY, String beginRA, String beginDec)
Sets the begin to the given pixel and sky coordinates.
Arguments
`double` **beginX** `[INPUT, MANDATORY, default=no default value]`
The x-pixel-coordinate of the begin, as double
`double` **beginY** `[INPUT, MANDATORY, default=no default value]`
The y-pixel-coordinate of the begin, as double
`String` **beginRA** `[INPUT, MANDATORY, default=no default value]`
The right ascension of the begin, as String
`String` **beginDec** `[INPUT, MANDATORY, default=no default value]`
The declination of the begin, as String

setEnd (double endX, double endY)
Sets the end to the given pixel coordinates.
Arguments
`double` **endX** `[INPUT, MANDATORY, default=no default value]`
The x-pixel-coordinate of the end, as double
`double` **endY** `[INPUT, MANDATORY, default=no default value]`

setEnd (double endX, double endY)

The y-pixel-coordinate of the end, as double

setEnd (double endX, double endY, String endRA, String endDec)

Sets the end to the given pixel and sky coordinates.

Arguments

`double` **endX** `[INPUT, MANDATORY, default=no default value]`

The x-pixel-coordinate of the end, as double

`double` **endY** `[INPUT, MANDATORY, default=no default value]`

The y-pixel-coordinate of the end, as double

`String` **endRA** `[INPUT, MANDATORY, default=no default value]`

The right ascension of the end, as String

`String` **endDec** `[INPUT, MANDATORY, default=no default value]`

The declination of the end, as String

setLengthInArcsec (double length)

Sets the length of the straight line.

Sets the length of the straight line in arcsec for this Profile.

Argument

`double` **length** `[INPUT, MANDATORY, default=no default value]`

The length to set as the length of the straight line in arcsec, as double

setProfile (Double1d profile)

Sets the profile for this Profile to the given profile.

Argument

`Double1d` **profile** `[INPUT, MANDATORY, default=no default value]`

The profile to set as the intensities for this Profile, as Double1d

setUnit (Unit unit)

Sets the unit for this Profile.

Argument

`Unit` **unit** `[INPUT, MANDATORY, default=no default value]`

The unit to set as the unit for this Profile, as Unit

Double1d **getBeginPixelCoordinates**

Returns the begin for this Profile in pixel coordinates.

Return

Double1d

Returns the begin for this Profile in pixel coordinates.

String1d **getBeginSkyCoordinates**

Returns the begin for this Profile in sky coordinates.

Return

String1d

***String1d* getBeginSkyCoordinates**

Returns the begin for this Profile in sky coordinates.

***Double1d* getEndPixelCoordinates**

Returns the end for this Profile in pixel coordinates.

Return

Double1d

Returns the end for this Profile in pixel coordinates.

***String1d* getEndSkyCoordinates**

Returns the end for this Profile in sky coordinates.

Return

String1d

Returns the end for this Profile in sky coordinates.

***double* getLength**

Returns the length of the straight line in arcsec.

Returns the length of the straight line in arcsec for this Profile.

Return

double

The length of the straight line in arcsec for this Profile.

***Double1d* getProfile**

Returns the profile for this Profile.

Return

Double1d

Returns the profile for this Profile.

***Double1d* getPixels**

Returns the x-axis of this Profile.

Returns the x-axis of this Profile. It indicates how many pixels lie between begin and end of the straight line.

Return

Double1d

The x-axis of this Profile. It indicates how many pixels lie between begin and end of the straight line.

***Double1d* getArcsec**

Returns the x-axis of this Profile.

Returns the x-axis of this Profile. It indicates how many arcsec lie between begin and end of the straight line.

Return

Double1d

Double1d **getArcsec**
The x-axis of this Profile. It indicates how many arcsec lie between begin and end of the straight line.

String **getUnit**
Returns the unit for this Profile. **Return** **String** Returns the unit for this Profile.

See also

- Developers Manual: herschel.ia.dataset.image.Profile

1.323. profile

Full Name:	herschel.ia.toolbox.image.ProfileTask
Alias:	profile
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import ProfileTask
Category	Images/Analysis

Description

This is a task to make intensity plots along a straight line.

This is a task which determines the intensity profile of the pixels along a straight line on an image. This straight line is characterised by the pixel or sky coordinates of begin and end.

Example

Example 1: This is an example of how to use the profile :

```
profile = profile(myImage, beginX = 118.875, beginY = 131.875, endX = 160.375,
 endY = 95.625)
profile =  profile(myImage, beginRa = "01:42:31.622", beginDec =
 "+51:36:06.31",
                   endRa = "01:42:04.917", endDec = "+51:32:28.79")
```

API Summary

Properties
Image **image** [INPUT, MANDATORY, default=None]
Double **beginX** [INPUT, OPTIONAL, default=NaN]
Double **beginY** [INPUT, OPTIONAL, default=NaN]
Double **endX** [INPUT, OPTIONAL, default=Default value : NaN]
Double **endY** [INPUT, OPTIONAL, default=NaN]
String **beginRa** [INPUT, OPTIONAL, default="beginRa"]
String **beginDec** [INPUT, OPTIONAL, default="beginDec"]
String **endRa** [INPUT, OPTIONAL, default="endRa"]
String **endDec** [INPUT, OPTIONAL, default="endDec"]
String **beginRA** [INPUT, OPTIONAL, default="beginRA"]
String **endRA** [INPUT, OPTIONAL, default="endRA"]
Profile **profile** [OUTPUT, MANDATORY, default=None]

API details

Properties

Image image [INPUT, MANDATORY, default=None]
This is the image on which the straight line is drawn.

Double beginX [INPUT, OPTIONAL, default=NaN]

This is the x-pixel-coordinate of the begin of the straight line.

Double beginY [INPUT, OPTIONAL, default=NaN]

This is the y-pixel-coordinate of the begin of the straight line.

Double endX [INPUT, OPTIONAL, default=Default value : NaN]

This is the x-pixel-coordinate of the end of the straight line.

Double endY [INPUT, OPTIONAL, default=NaN]

This is the y-pixel-coordinate of the end of the straight line.

String beginRa [INPUT, OPTIONAL, default="beginRa"]

This is the right ascension of the begin of the straight line. Using this only makes sense if the input image has a valid Wcs attached to it. The input format for the right ascension is "hh mm ss.s", "hh:mm:ss.s" or "hh.hhh".

String beginDec [INPUT, OPTIONAL, default="beginDec"]

This is the declination of the begin of the straight line. Using this only makes sense if the input image has a valid Wcs attached to it. The input format for the declination is "[+/-]dd mm ss.s", "[+/-]dd:mm:ss.s" or "[+/-]dd.ddd".

String endRa [INPUT, OPTIONAL, default="endRa"]

This is the right ascension of the end of the straight line. Using this only makes sense if the input image has a valid Wcs attached to it. The input format for the right ascension is "hh mm ss.s", "hh:mm:ss.s" or "hh.hhh".

String endDec [INPUT, OPTIONAL, default="endDec"]

This is the declination of the end of the straight line. Using this only makes sense if the input image has a valid Wcs attached to it. The input format for the declination is "[+/-]dd mm ss.s", "[+/-]dd:mm:ss.s" or "[+/-]dd.ddd".

String beginRA [INPUT, OPTIONAL, default="beginRA"]

This is the right ascension of the begin of the straight line. Using this only makes sense if the input image has a valid Wcs attached to it. The input format for the right ascension is "hh mm ss.s", "hh:mm:ss.s" or "hh.hhh". Deprecated : Use beginRa

String endRA [INPUT, OPTIONAL, default="endRA"]

This is the right ascension of the end of the straight line. Using this only makes sense if the input image has a valid Wcs attached to it. The input format for the right ascension is "hh mm ss.s", "hh:mm:ss.s" or "hh.hhh". Deprecated : Use endRa

Profile profile [OUTPUT, MANDATORY, default=None]

This is the resulting profile plot.

See also

- Developers Manual: herschel.ia.toolbox.image.ProfileTask

1.324. qMethod

Full Name:	herschel.ia.toolbox.pointing.qMethod
Type:	Jython Task -
Import:	from herschel.ia.toolbox.pointing import qMethod
Category	Toolboxes/Pointing

Description

Implementation of Davenport's q-method.

See RD.1 for details. References: RD.1 M.D. Shuster. The Generalized Wahba Problem. The Journal of the Astronautical Sciences, 54(2):245-259, Apr--Jun 2006.

Example

Example 1: num_stars = 5

```
stars_brf   = Double2d(18,3)
stars_meq   = Double2d(18,3)
measerr_est = 1.0e-5
for star in range(num_stars):
  stars_brf[star,:] =...
  stars_meq[star,:] =...
(quat, loss, TASTE, status) = qMethod(num_stars, stars_brf, stars_meq,
 measerr_est)
```

API Summary

Jython Syntax

```
(quat, loss, TASTE, status) = qMethod(num_stars, stars_brf,
stars_meq, measerr_est)
```

Properties

Int **num_stars** [INPUT, MANDATORY, default=NO default value]
Double2d **stars_brf** [INPUT, MANDATORY, default=NO default value]
Double2d **stars_meq** [INPUT, MANDATORY, default=NO default value]
Double **measerr_est** [INPUT, MANDATORY, default=NO default value]
Quaternion **quat** [OUTPUT, MANDATORY, default=NO default value]
Double **loss** [OUTPUT, MANDATORY, default=NO default value]
Double **TASTE** [OUTPUT, MANDATORY, default=NO default value]
Double **status** [OUTPUT, MANDATORY, default=NO default value]

API details

Properties

Int num_stars [INPUT, MANDATORY, default=NO default value]

No. of stars, N, with which to determine attitude

Double2d stars_brf [INPUT, MANDATORY, default=NO default value]

Measured star vectors (BRF)

Double2d stars_meq [INPUT, MANDATORY, default=NO default value]

Catalogue star vectors (MEQ2000)

Double measerr_est [INPUT, MANDATORY, default=NO default value]

Best estimate of star vector measurement error (1 sigma, rad.)

Quaternion quat [OUTPUT, MANDATORY, default=NO default value]

Estimated attitude quaternion

Double loss [OUTPUT, MANDATORY, default=NO default value]

Minimized loss function, J(A*) (with a_k = 1/N) in RD.1

Double TASTE [OUTPUT, MANDATORY, default=NO default value]

Shuster's TASTE variable

Double status [OUTPUT, MANDATORY, default=NO default value]

Return status (0 = success, 1 = too few stars, 2 = other error)

History

- 2014-06-04 - CAS: Initial version
- 2014-06-06 - CAS: Import some much-needed modules.
- 2014-06-06 - CAS: Import Herschel as well for good measure.
- 2014-06-06 - CAS: This function also needs the basic toolbox. Sigh!
- 2014-11-12 - CAS: Ensured that all output variables are always assigned a value.
- 2015-05-27 - CAS: Header modified.
- 2015-08-12 - CAS: Now it is ensured that the minimized loss function is never negative. Input variable (and its description) also modified.

1.325. QRDecomposition

Full Name:	herschel.ia.numeric.toolbox.matrix.QRDecomposition
Alias:	QRDecomposition
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.matrix import QRDecomposition
Category	Mathematics/Matrices

Description

QRDecomposition

Wraps the Jama class for DP. For an m-by-n matrix A with m >= n, the QR decomposition is an m-by-n orthogonal matrix Q and an n-by-n upper triangular matrix R so that A = Q*R. The QR decomposition always exists, even if the matrix does not have full rank, so the constructor will never fail. The primary use of the QR decomposition is in the least squares solution of nonsquare systems of simultaneous linear equations. This will fail if isFullRank() returns false.

Example

Example 1: Decompose a 2D array and get QR components.

```
A = Double2d([[ 2.0,  1.0,  1.0],[ 4.0, -6.0,  0.0],[-2.0,  7.0,  2.0]])
B=Double2d( [ [3.0],[-8.0],[10.0] ] )
QRD = QRDecomposition(A)
leastSquaresSolutionAXeqB = B.apply(QRD)
print leastSquaresSolutionAXeqB
# [
# [0.9999999999999971],
# [1.9999999999999978],
# [-0.9999999999999969]
# ]
isFullRank = QRD.isFullRank
print isFullRank
# 1
HouseholderVectors = QRD.h
print HouseholderVectors
# [
# [1.4082482904638631,0.0,0.0],
# [0.8164965809277261,1.4099776105529318,0.0],
# [-0.4082482904638631,-0.9120955864630134,2.0]
# ]
rightTriangularFactor = QRD.r
print rightTriangularFactor
# [
# [-4.898979485566356,7.3484692283495345,0.408248290463863],
# [0.0,5.656854249492381,2.1213203435596424],
# [0.0,0.0,-0.5773502691896262]
# ]
orthFactor = QRD.qs
print orthFactor
# [
# [-0.40824829046386313,0.7071067811865475,0.5773502691896257],
# [-0.8164965809277261,5.551115123125783E-17,-0.5773502691896256],
# [0.4082482904638631,0.7071067811865475,-0.577350269189626]
```

API Summary

Jython Syntax
`A=Double2d()`

Jython Syntax
B=Double2d() res=B.apply(QRDecomposition(A)) QRD = QRDecomposition(A) leastSquaresSolutionAXeqB = B.apply(QRD) isFullRank = QRD.isFullRank HouseholderVectors = QRD.h rightTriangularFactor = QRD.r orthFactor = QRD.qs

Property
Double2d **A** [INPUT, MANDATORY, default=no default value]

API details

Property

Double2d A [INPUT, MANDATORY, default=no default value]
Input must be a Double2d or Float2d array.

See also

- Developers Manual: herschel.ia.numeric.toolbox.matrix.QRDecomposition

1.326. QRMS

Full Name:	herschel.ia.numeric.toolbox.basic.Qrms
Alias:	QRMS
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Qrms
Category	Mathematics/Statistics

Description

Returns the quadratic root mean square of a set of values.

The quadratic root mean square is computed as follows: *QRMS = SQRT ((1/N)*SUM(x**2))*, where *N* is the number of elements in the array and *x* is an array element. In case of complex arrays, the QRMS is computed on the absolute values of the elements.

Example

Example 1: Apply QRMS to a Float1d

```
x=Float1d([1,3,2,3,4])
print QRMS(x) # 2.792848008753788
```

API Summary

Jython Syntax

```
&lt;y&gt;=QRMS(&lt;x&gt;)
```

Properties

```
Array x [INPUT, MANDATORY, default=no default value]
Double y [OUTPUT, MANDATORY, default=no default value]
```

API details

Properties

`Array x [INPUT, MANDATORY, default=no default value]`

The input array. Integer arrays are implicitly converted to floating point arrays.

`Double y [OUTPUT, MANDATORY, default=no default value]`

The quadratic root mean square of the values in the input array.

See also

- MEAN
- MEDIAN
- STDDEV
- VARIANCE

- Developers Manual: herschel.ia.numeric.toolbox.basic.Qrms

1.327. quat_to_att

Full Name:	herschel.ia.toolbox.pointing.quat_to_att
Type:	Jython Task -
Import:	from herschel.ia.toolbox.pointing import quat_to_att
Category	Toolboxes/Pointing

Description

Converts an attitude quaternion into an attitude matrix.

Example

Example 1: from herschel.share.fltdyn.math import Quaternion

```
quat = Quaternion(0, 0, 0, 1)
mat  = quat_to_att(quat)
```

API Summary

Jython Syntax

```
mat = quat_to_att(quat)
```

Properties

Quaternion **quat** [INPUT, MANDATORY, default=NO default value]

Double2d **mat** [OUTPUT, MANDATORY, default=NO default value]

API details

Properties

Quaternion quat [INPUT, MANDATORY, default=NO default value]

Attitude quaternion

Double2d mat [OUTPUT, MANDATORY, default=NO default value]

Attitude matrix

History

- 2014-06-04 - CAS: Initial version
- 2014-06-06 - CAS: Import Double2d to see if this solves problem.
- 2014-06-06 - CAS: Import Herschel as well for good measure.

1.328. raDecRoll2xyz

Full Name:	herschel.ia.toolbox.pointing.raDecRoll2xyz
Type:	Jython Task -
Import:	from herschel.ia.toolbox.pointing import raDecRoll2xyz
Category	Toolboxes/Pointing

Description

Function to go from difference in ra/dec/roll to spacecraft xyz axis angular offsets.

Example

Example 1: x,y,z = raDecRoll2xyz(1.,1.,1.,0.,0.,0.)

```
x,y,z = raDecRoll2xyz(90.,90.,45.,0.,0.,0., degrees=True)
ra = Double1d([45., 90.])
dec = Double1d([90., 45.])
roll = Double1d([90., 90.])
x,y,z = raDecRoll2xyz(ra,dec,roll,0.,0.,0., degrees=True)
ra = Double1d([45., 90.])
dec = Double1d([90., 45.])
roll = Double1d([90., 90.])
refRa = Double1d([45.1, 90.2])
refDec = Double1d([90.1, 45.2])
refRoll = Double1d([90.1, 90.2])
x,y,z = raDecRoll2xyz(ra,dec,roll,refRa,refDec,refRoll, degrees=True)
```

API Summary

Jython Syntax
`x, y, z = raDecRoll2xyx(ra, dec, roll, refRa, refDec, refRoll,degrees=False|True)`

Properties
double/Double1d **ra** [INPUT, MANDATORY, default=NO default value]
double/Double1d **dec** [INPUT, MANDATORY, default=NO default value]
double/Double1d **roll** [INPUT, MANDATORY, default=NO default value]
double/Double1d **refRa** [INPUT, MANDATORY, default=NO default value]
double/Double1d **refDec** [INPUT, MANDATORY, default=NO default value]
double/Double1d **refRoll** [INPUT, MANDATORY, default=NO default value]
Boolean **degrees** [INPUT, OPTIONAL, default=Default True]
Double1d **x** [OUTPUT, MANDATORY, default=NO default value]
Double1d **y** [OUTPUT, MANDATORY, default=NO default value]
Double1d **z** [OUTPUT, MANDATORY, default=NO default value]

API details

Properties

double/Double1d ra [INPUT, MANDATORY, default=NO default value]

right Ascension (degrees or radians)

double/Double1d dec [INPUT, MANDATORY, default=NO default value]

declination (degrees or radians)

double/Double1d roll [INPUT, MANDATORY, default=NO default value]

declination (degrees or radians)

double/Double1d refRa [INPUT, MANDATORY, default=NO default value]

reference position right Ascension (degrees or radians)

double/Double1d refDec [INPUT, MANDATORY, default=NO default value]

reference position declination (degrees or radians)

double/Double1d refRoll [INPUT, MANDATORY, default=NO default value]

reference position declination (degrees or radians)

Boolean degrees [INPUT, OPTIONAL, default=Default True]

input and output angles in degrees; if False: input / output in radians

Double1d x [OUTPUT, MANDATORY, default=NO default value]

rotation angles over x spacraft axis (degrees or radians)

Double1d y [OUTPUT, MANDATORY, default=NO default value]

rotation angles over y spacraft axis (degrees or radians)

Double1d z [OUTPUT, MANDATORY, default=NO default value]

rotation angles over z spacraft axis (degrees or radians)

History

- 2013-04-22 - BV: Initial version
- 2013-07-23 - BV: Corrected matrix multiplication order
- 2013-08-22 - BV: HCSS-18486 Use infinity norm to solve rotation ambiguity
- 2013-09-26 - BV: HCSS-18567 Review disabled jexamples

1.329. RadialV

Full Name:	herschel.ia.toolbox.astro.RadialV
Alias:	RadialV
Type:	Java Class -
Import:	from herschel.ia.toolbox.astro import RadialV
Category	Astronomical utilities

Description

Calculates S/C velocity projection in the direction of the pointing (radial velocity).

It optionally includes Sun to LSR and Earth to Sun velocities to help in Doppler correction of observed frequencies and to calculate line-of-sight speed from product data in J2000.

Pointing and Orbit products should be contained in an observation context processed by the SPG. The corrections applied from the data obtained from products can include the sum of any of:

- RadialV.NONE
- RadialV.SUNTOLSR (speed of the Sun wrt the Local Standard of Rest)
- RadialV.EARTHTOSUN (speed of the Earth wrt the Sun); can be approximate or fine when using an Ephemeris)

The sign of the radial velocity is negative towards the target and positive otherwise (Doppler effect sign convention). This code is intended to be used from Java, to avoid using a task: check the Javadoc for more information.

API Summary

Fields
int **NONE** value for flag parameter
int **SUNTOLSR** value for flag parameter (add/or to combine)
int **EARTHTOSUN** value for flag parameter (add/or to combine)

Methods
double **getProjection** (Vector3 velocity, Vector3 pointing, FineTime t, int flag, Ephem ep) Function for vectors
double **getProjection** (OrbitEphemerisProduct vt, PointingData pt, FineTime t, int flag, Ephem ep, TimeCorrProduct t2t) Function for products
double[] **getProjections** (OrbitEphemerisProduct vt, PointingData pt, FineTime[] ts, int flag, Ephem ep, TimeCorrProduct t2t)

Methods
Function for products with a list for TAIs
double[] **getProjectionsWithNans** (OrbitEphemerisProduct vt, PointingData pt, FineTime[] ts, int flag, Ephem ep, TimeCorrProduct t2t)
Function for products with a list for TAIs that will not fail on bad data (will return NaNs instead)

Limitations

The following classes in the Javadoc (UM) contain further info:

- herschel.share.fltdyn.ephem.Ephemerides
- herschel.share.fltdyn.time.FineTime
- herschel.ia.obs.auxiliary.orbitephem.OrbitEphemerisProduct
- herschel.ia.obs.auxiliary.pointing.PointingProduct
- herschel.ia.obs.auxiliary.fltdyn.Ephemerides

API Details

Fields

***int* NONE**

value for flag parameter

Do not apply any correction

Example

no correction

```
flag = RadialV.NONE
```

***int* SUNTOLSR**

value for flag parameter (add/or to combine)

Apply speed of the Sun wrt the Local Standard of Rest correction

Example

LSR correction

```
flag = RadialV.SUNTOLSR
```

***int* EARTHTOSUN**

value for flag parameter (add/or to combine)

Apply speed of the Earth wrt the Sun correction

Example

Sun and Earth speed corrections

```
flag = RadialV.SUNTOLSR + RadialV.EARTHTOSUN
```

Methods

double getProjection (Vector3 velocity, Vector3 pointing, FineTime t, int flag, Ephem ep)

Function for vectors

Get radial velocity projection in the direction of the pointing at time t, passing vectors Allows corrections for Sun's v wrt LSR and Earth's v wrt Sun via numeric flag The sign of the radial velocity is negative towards the target and positive otherwise (Doppler effect sign convention).

Arguments

`Vector3` **`velocity`** `[INPUT, MANDATORY, default=no default value]`

Velocity vector of the S/C in km/s, using J2000

`Vector3` **`pointing`** `[INPUT, MANDATORY, default=no default value]`

Pointing Vector of the S/C in km/s, using J2000

`FineTime` **`t`** `[INPUT, MANDATORY, default=no default value]`

The TAI time to get the precessed Sun velocity

`int` **`flag`** `[INPUT, MANDATORY, default=no default value]`

The corrections to apply (RadialV.NONE .. RadialV.SUNTOLSR + RadialV.EARTHTOSUN)

`Ephem` **`ep`** `[INPUT, MANDATORY, default=no default value]`

Ephemerides to assist in corrections, if null and flag has EARTHTOSUN it will use internal files

Return

double

the (signed) component of the velocity in the pointing direction, in km/s (Doppler sign convention)

double getProjection (OrbitEphemerisProduct vt, PointingData pt, FineTime t, int flag, Ephem ep, TimeCorrProduct t2t)

Function for products

Get radial velocity projection in the direction of the pointing at time t, passing auxiliary products Allows corrections for Sun's v wrt LSR and Earth's v wrt Sun via numeric flag The sign of the radial velocity is negative towards the target and positive otherwise (Doppler effect sign convention). Interpolation will be linear.

Arguments

`OrbitEphemerisProduct` **`vt`** `[INPUT, MANDATORY, default=no default value]`

An OrbitEphemeris product from where v(t) of the S/C can be obtained

`PointingData` **`pt`** `[INPUT, MANDATORY, default=no default value]`

A Pointing product from where pointing(t) of the S/C can be obtained

`FineTime` **`t`** `[INPUT, MANDATORY, default=no default value]`

The TAI time instant to get both the velocity and the pointing

`int` **`flag`** `[INPUT, MANDATORY, default=no default value]`

The corrections to apply (RadialV.NONE .. RadialV.SUNTOLSR + RadialV.EARTHTOSUN)

`Ephem` **`ep`** `[INPUT, MANDATORY, default=no default value]`

***double* getProjection (OrbitEphemerisProduct vt, PointingData pt, FineTime t, int flag, Ephem ep, TimeCorrProduct t2t)**

Ephemerides to assist in corrections, if null and flag has EARTHTOSUN it will use internal files

`TimeCorrProduct` **`t2t`** `[INPUT, MANDATORY, default=no default value]`

TimeCorrProduct needed to access Pointing product with FineTimes

Return

double

the (signed) component of the velocity in the pointing direction, in km/s (Doppler sign convention)

***double[]* getProjections (OrbitEphemerisProduct vt, PointingData pt, FineTime[] ts, int flag, Ephem ep, TimeCorrProduct t2t)**

Function for products with a list for TAIs

Get radial velocity projection in the direction of the pointing at time t, passing auxiliary products Allows corrections for Sun's v wrt LSR and Earth's v wrt Sun via numeric flag The sign of the radial velocity is negative towards the target and positive otherwise (Doppler effect sign convention). If, for some t, v could not be generated,an info message will be generated. At the end, a RuntimeException will be thrown detailing all failed FineTimes. Throws a RuntimeException if some of the items cannot be calculated. Interpolation will be linear.

Arguments

`OrbitEphemerisProduct` **`vt`** `[INPUT, MANDATORY, default=no default value]`

An OrbitEphemeris product from where v(t) of the S/C can be obtained

`PointingData` **`pt`** `[INPUT, MANDATORY, default=no default value]`

A Pointing product from where pointing(t) of the S/C can be obtained

`FineTime[]` **`ts`** `[INPUT, MANDATORY, default=no default value]`

The TAI time instant list to get both the velocity and the pointing

`int` **`flag`** `[INPUT, MANDATORY, default=no default value]`

The corrections to apply (RadialV.NONE .. RadialV.SUNTOLSR + RadialV.EARTHTOSUN)

`Ephem` **`ep`** `[INPUT, MANDATORY, default=no default value]`

Ephemerides to assist in corrections, if null and flag has EARTHTOSUN it will use internal files

`TimeCorrProduct` **`t2t`** `[INPUT, MANDATORY, default=no default value]`

TimeCorrProduct needed to access Pointing product with FineTimes

Return

double[]

the array of (signed) component of the velocity in the pointing direction, in km/s (Doppler sign convention)

***double[]* getProjectionsWithNans (OrbitEphemerisProduct vt, PointingData pt, FineTime[] ts, int flag, Ephem ep, TimeCorrProduct t2t)**

Function for products with a list for TAIs that will not fail on bad data (will return NaNs instead)

Get radial velocity projection in the direction of the pointing at time t, passing auxiliary products Allows corrections for Sun's v wrt LSR and Earth's v wrt Sun via numeric flag The sign of the radi-

`double[] getProjectionsWithNans (OrbitEphemerisProduct vt, PointingData pt, FineTime[] ts, int flag, Ephem ep, TimeCorrProduct t2t)`

al velocity is negative towards the target and positive otherwise (Doppler effect sign convention). If, for some t, v could not be generated,an info message will be generated. At the end, a warning message will be generated with a count of all failed FineTimes, the failed items will have NaN. Interpolation will be linear.

Arguments

`OrbitEphemerisProduct` **`vt`** `[INPUT, MANDATORY, default=no default value]`

An OrbitEphemeris product from where v(t) of the S/C can be obtained

`PointingData` **`pt`** `[INPUT, MANDATORY, default=no default value]`

A Pointing product from where pointing(t) of the S/C can be obtained

`FineTime[]` **`ts`** `[INPUT, MANDATORY, default=no default value]`

The TAI time instant list to get both the velocity and the pointing

`int` **`flag`** `[INPUT, MANDATORY, default=no default value]`

The corrections to apply (RadialV.NONE .. RadialV.SUNTOLSR + RadialV.EARTHTOSUN)

`Ephem` **`ep`** `[INPUT, MANDATORY, default=no default value]`

Ephemerides to assist in corrections, if null and flag has EARTHTOSUN it will use internal files

`TimeCorrProduct` **`t2t`** `[INPUT, MANDATORY, default=no default value]`

TimeCorrProduct needed to access Pointing product with FineTimes

Return

double[]

the array of (signed) component of the velocity in the pointing direction, in km/s (Doppler sign convention), can have NaNs

See also

- radialV
- Developers Manual: herschel.ia.toolbox.astro.RadialV

History

- `2010-12-02 - JDS`: New, using Doppler effect sign convention. New function returning NaNs
- `2011-03-16 - JDS`: Doc fixes

1.330. radialV

Full Name:	herschel.ia.toolbox.astro.RadialVTask
Alias:	radialV
Type:	Java Task -
Import:	from herschel.ia.toolbox.astro import RadialVTask
Category	Astronomical utilities

Description

The RadialV Task calculates S/C radial velocity projection in the direction of the pointing.

It optionally includes Sun to LSR and Earth to Sun velocities to help in Doppler correction of observed frequencies.

Pointing and Orbit products should be contained in an observation context processed by the SPG. The available corrections applied from the data obtained from products are: "NONE", "SUN", "EARTH", "SUN_AND_EARTH" (default). The sign of the radial velocity is negative towards the target and positive otherwise (Doppler effect sign convention).

Examples

Example 1: Getting the radial velocity at the start of an observation (using context)

```
aux = obs.getAuxiliary()
ft = aux.getTimeCorrelation().toFineTime(aux.getPointing().getStartTime() + 1)
ephem = herschel.ia.obs.auxiliary.fltdyn.Ephemerides(aux.getOrbitEphemeris())
v, ra, dec = radialV(context=obs, time=ft, ephemerides=ephem)
print "at %s radial velocity is %f Km/s, RA= %f, DEC= %f" % (ft, v, ra, dec)
```

Example 2: Full call with individual products at the end of an observation (using auxiliary products)

```
aux = obs.auxiliary
pointing = aux.pointing
orbit = aux.orbitEphemeris
timecorr = aux.timeCorrelation
ft = timecorr.toFineTime(pointing.endTime)
ephem = herschel.ia.obs.auxiliary.fltdyn.Ephemerides(orbit)
v, ra, dec = radialV(pointing=pointing,orbit=orbit, timecorr=timecorr, time=ft,
 ephemerides=ephem)
print "at %s, vr = %f Km/s (RA = %f, DEC = %f)" % (ft, v, ra, dec)
```

Example 3: Comparing uncorrected with corrected radial velocity

```
# Load the products from ObservationContext obs, use individual products or
 RadialV class for repeated access
aux = obs.auxiliary
pointing = aux.pointing
orbit = aux.orbitEphemeris
timecorr = aux.timeCorrelation
ephem = herschel.ia.obs.auxiliary.fltdyn.Ephemerides(orbit)
# Build the ranges for sampling 'numPoints' points
numPoints = 100
obts = range(pointing.startTime + 1, pointing.endTime, (pointing.endTime -
 pointing.startTime) / numPoints)
ids = range(numPoints)
# Outputs
vplain = Double1d() # uncorrected
```

Example 3: Comparing uncorrected with corrected radial velocity

```
vcorr = Double1d() # corrected
for i in ids:
    t = timecorr.toFineTime(obts[i])
    vcorr.append(radialV(pointing=pointing,orbit=orbit,time=t,
 timecorr=timecorr, ephemerides=ephem)[0])
    vplain.append(radialV(pointing=pointing,orbit=orbit,time=t,
 timecorr=timecorr, correct="NONE")[0])
# plot vs ids (could use obts or fts)
xs = Int1d(ids)
myPlot=PlotXY(xs, vplain)
myPlot=PlotXY(xs,vcorr)
```

API Summary

Jython Syntax

velocity, ra, dec = radialV([<storage>| <context>| (<pointing>, <orbit>, <timecorr>)], <time> \

[, <ephemerides>, <correct>])

You must use either a storage or a context or pointing, orbit and timecorr as the S/C data source.

Corrections: "NONE", "SUN", "EARTH", "SUN_AND_EARTH" (default).

Properties

ProductStorage **storage** [INPUT, OPTIONAL, default=null]
ObservationContext **context** [INPUT, OPTIONAL, default=null]
PointingData **pointing** [INPUT, OPTIONAL, default=null]
OrbitEphemerisProduct **orbit** [INPUT, OPTIONAL, default=null]
TimeCorrProduct **timecorr** [INPUT, OPTIONAL, default=null]
FineTime **time** [INPUT, MANDATORY, default=null]
Ephem **ephemerides** [INPUT, OPTIONAL, default=null]
String **correct** [INPUT, OPTIONAL, default="SUN_AND_EARTH"]
Double **velocity** [OUTPUT, MANDATORY, default=null]
Double **ra** [OUTPUT, OPTIONAL, default=null]
Double **dec** [OUTPUT, OPTIONAL, default=null]

Limitations

- If you do not provide an Ephemerides object, EARTHTOSUN correction will use internal files.
- Using pointing and orbit products requires using the timecorr product also.

The following classes in the Javadoc (UM) contain further info:

- herschel.share.fltdyn.ephem.Ephemerides
- herschel.share.fltdyn.time.FineTime
- herschel.ia.obs.auxiliary.orbitephem.OrbitEphemerisProduct
- herschel.ia.obs.auxiliary.pointing.PointingProduct

- herschel.ia.obs.auxiliary.fltdyn.Ephemerides

API details

Properties

ProductStorage storage [INPUT, OPTIONAL, default=null]

Storage that contains the time we are working with. When storage is used, context, pointing, orbit and timecorr cannot be used

ObservationContext context [INPUT, OPTIONAL, default=null]

Processed Observation Context that contains the time we are working with. When context is used, storage, pointing, orbit and timecorr cannot be used

PointingData pointing [INPUT, OPTIONAL, default=null]

The PointingProduct applicable to the time we are working with. When pointing is used, orbit and timecorr must be used too (storage and context cannot be used)

OrbitEphemerisProduct orbit [INPUT, OPTIONAL, default=null]

The OrbitEphemerisProduct applicable to the time we are working with. When orbit is used, pointing and timecorr must be used too (storage and context cannot be used)

TimeCorrProduct timecorr [INPUT, OPTIONAL, default=null]

The TimeCorrProduct applicable to the time we are working with. When timecorr is used, pointing and orbit must be used too (storage and context cannot be used)

FineTime time [INPUT, MANDATORY, default=null]

The time for which we want to get the radial velocity

Ephem ephemerides [INPUT, OPTIONAL, default=null]

The Ephemerides object used in the corrections. If not used, and correct has "SUN_AND_EARTH", it will use internal files

String correct [INPUT, OPTIONAL, default="SUN_AND_EARTH"]

The corrections flags to be applied to the velocity. Possible values are: Corrections: "NONE", "SUN", "EARTH", "SUN_AND_EARTH" (default)

Double velocity [OUTPUT, MANDATORY, default=null]

The (signed) corrected velocity projection in the pointing direction (radial velocity), in km/s. Negative towards the target, positive otherwise (Doppler convention). This is in J2000

Double ra [OUTPUT, OPTIONAL, default=null]

RA of the pointing at time. Range = 0 .. 2*PI

Double dec [OUTPUT, OPTIONAL, default=null]

DEC of the pointing at time. Range -PI/2 .. PI/2

See also

- RadialV

- Developers Manual: herschel.ia.toolbox.astro.RadialVTask

History

- `2008-10-10 - JDS`: first interface candidate release
- `2008-12-12 - JDS`: added precession algorithm
- `2009-05-15 - JDS`: moved to using a constant Sun speed vector in J2000, include Earth velocity
- `2009-06-26 - JDS`: Initial task version, with tests, using Ephemeris
- `2010-02-01 - JDS`: Using Stumpf's model (+- 40 cm/s)
- `2010-12-01 - JDS`: Changed name to RadialV, changed sign to Doppler, skip nightly tests (need observation)
- `2012-03-24 - JDS`: parameter rename corrections -> correct (String)
- `2012-05-08 - JDS`: Rewritten examples (code review)
- `2013-05-20 - JDS`: Updated: with no (null) Ephemerides (uses internal files, no error)

1.331. RandomGauss

Full Name:	herschel.ia.numeric.toolbox.random.RandomGauss
Alias:	RandomGauss
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.random import RandomGauss
Category	Mathematics/Random numbers

Description

RandomGauss generates or populates an array with normally N(0,1) distributed pseudo random numbers.

Examples

Example 1: example usage of a random function

```
f=RandomGauss()
f=RandomGauss(seed=72654)
y=f(Double1d(10))         # creates a random array with same dimensions and
 type as x
y=Double1d(10).apply(f)   # like wise, but java style
Double1d(10).perform(f)   # changes the values in x
```

Example 2: To generate 100 Double values normally distributed with mean=0 and sigma = 1.0

```
f = RandomGauss()
y = Double1d(100).apply(f)
# To make it with mean=10. and sigma=0.1
ynew = 10. + y * 0.1
```

API Summary

Jython Syntax

```
f = RandomGauss([seed=&lt;seed&gt;])
where seed is an optional input used for the random generator
```

See also

- Developers Manual: herschel.ia.numeric.toolbox.random.RandomGauss

1.332. RandomPoisson

Full Name:	herschel.ia.numeric.toolbox.random.RandomPoisson
Alias:	RandomPoisson
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.random import RandomPoisson
Category	Mathematics/Random numbers

Description

RandomPoisson generates Poisson random numbers sequence with a given mean.

Examples

Example 1: example usage of a random function

```
f=RandomPoisson(20)
f=RandomPoisson(15,seed=72654)
y=f(Long1d(10))         # creates a random array with same dimensions and type
 as x
y=Long1d(10).apply(f)   # like wise, but java style
Long1d(10).perform(f)   # changes the values in x
```

Example 2: To generate 100 Double values who follow Poisson distribution with mean 5

```
f = RandomPoisson(5.0)
y = Long1d(100).apply(f)
```

API Summary

Jython Syntax

```
f = RandomPoisson(&lt;mean&gt;,seed=&lt;seed&gt;)
where mean is a mandatory input, the Poisson expectation value
(mean);
and seed is an optional input, the seed for the random generator
```

See also

- Developers Manual: herschel.ia.numeric.toolbox.random.RandomPoisson

1.333. RandomUniform

Full Name:	herschel.ia.numeric.toolbox.random.RandomUniform
Alias:	RandomUniform
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.random import RandomUniform
Category	Mathematics/Random numbers

Description

RandomUniform generates or populates an array with uniformly-distributed pseudo-random numbers in the range 0 <= x < max.

Examples

Example 1: example usage of a random function

```
f=RandomUniform(15,seed=72654)
y=f(Double1d(20))         # creates a random array with same dimensions and
 type as x
y=Double1d(20).apply(f)   # like wise, but java style
Double1d(20).perform(f)   # changes the values in x
```

Example 2: To generate 100 Double values in [0,1)

```
f = RandomUniform()
y = Double1d(100).apply(f)
# To generate 100 Integer values in [0,100)
fx = RandomUniform(100.0)
yint = Int1d(100).apply(fx)
```

API Summary

Jython Syntax

```
f = RandomUniform([&lt;max&gt;],seed=&lt;seed&gt;) # between
[0,max), default max=1.
where max, if set the random sequence is in [0,max) otherwise in
[0,1). For Integer or Long
array types the value of max must be &gt; 1;
and the seed for the random generator
```

See also

- Developers Manual: herschel.ia.numeric.toolbox.random.RandomUniform

1.334. RealDoubleFFT

Full Name:	herschel.ia.numeric.toolbox.xform.util.RealDoubleFFT
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.xform.util import RealDoubleFFT
Category	Mathematics/Signal processing

Description

Gives the Fast Fourier Transform of real data using an FFT_PACK algorithm.

RealDoubleFFT#ft transforms real input into complex output. It returns the positive frequencies of the DFT. The remaining frequencies of the DFT can be reconstructed using conjugate symmetry (recall that for real input, $X_k = X_{N-k}$*).

If input has length N then output has length floor(N/2)+1. RealDoubleFFT#ft is fast for inputs with lengths that can be decomposed into small factors.

Use RealDoubleFFT#ft(double[]) to calculate the FFT of real data. However, if the length of input has large prime factors, use FFT instead.

Each RealDoubleFFT object is constructed with an argument N, which is equivalent to the length of input to the forward transform. If performing multiple transforms on inputs of the same length, create a single RealDoubleFFT object and use it for each transform. Processing speed is improved by minimizing the number of times RealDoubleFFT objects are created because each object performs initialization specific to the expected input length.

The inverse transform is RealDoubleFFT#bt. It transforms complex input to real output. If input has length L then output has length 2L-2 (if N, an argument for the constructor of RealDoubleFFT, is even) or 2L-1 (if N is odd).

Note that the output of RealDoubleFFT is not normalized. To normalize, divide the output of the inverse transform by N (an argument for the constructor of RealDoubleFFT). N is stored in RealDoubleFFT#normFactor.

Examples

Example 1: Shows how to get equivalent results with FFT, FFT_PACK, and RealDoubleFFT.

```
from herschel.ia.numeric.toolbox.xform.util import RealDoubleFFT
ts = 1E-6          # Sampling period (sec)
fc = 200000        # Carrier frequency (Hz)
fm = 2000          # Modulation frequency (Hz)
beta = 0.0003      # Modulation index (Hz)
N = 1000           # Number of samples
pi = java.lang.Math.PI   # define pi
t = Double1d.range(N) * ts
# a real, asymmetrical signal
signal = SIN(2 * pi * fc * t * (1 + beta * COS(2 * pi * fm * t)))
# Option 1.  FFT (slow)
spec_fft = FFT(Complex1d(signal))
# Option 2. FFT_PACK (slow)
spec_fftpack = FFT_PACK(Complex1d(signal))
# Option 3. RealDoubleFFT (fast)
rdfft = RealDoubleFFT(N)
spec_rdfft = Complex1d()
# Calculation destroys input signal
rdfft.ft(signal.copy().array,spec_rdfft)
# Compare lengths of output
print "Length of FFT output: %d"%spec_fft.size            # N
```

Example 1: Shows how to get equivalent results with FFT, FFT_PACK, and RealDoubleFFT.

```
print "Length of FFT_PACK output: %d"%spec_fftpack.size    # N
print "Length of RealDoubleFFT output: %d"%spec_rdfft.size # floor(N/2)+1
# Compare values in output (maximum modulus of difference)
print "Maximum modulus of difference output of FFT and FFT_PACK:"
print MAX(ABS(spec_fft - spec_fftpack))
print "Maximum modulus of difference between output of FFT and RealDoubleFFT
 (unextended):"
print MAX(ABS(spec_fft[0:(N/2)+1]- spec_rdfft))
# To get the full spectrum using RealDoubleFFT, use the following symmetry
 relationship
# which applies to the FFT of real input:
#    The (N-k)th complex FFT coefficient is the conjugate of kth complex FFT
 coefficient
#    where N is the length of the input.
# Extend the RealDoubleFFT output using symmetry X(N-k) = X(k)*
rdfft_extended = spec_rdfft.copy().append(Complex1d(REVERSE(spec_rdfft[1:(N-
N/2)].real), \
    REVERSE(-spec_rdfft[1:(N-N/2)].imag)))
# Compare FFT output with extended RealDoubleFFT output using maximum modulus
 of difference
print "Maximum modulus of difference between FFT output and RealDoubleFFT
 output extended by symmetry:"
print MAX(ABS(spec_fft - rdfft_extended))
```

Example 2: Recreate signal using forward and inverse transform

```
from herschel.ia.numeric.toolbox.xform.util import RealDoubleFFT
N = 1000
signal = Double1d.range(N)
# Apply forward transform to signal
rdfft = RealDoubleFFT(N)
c = Complex1d()
rdfft.ft(signal.toArray(), c)
# Recreate signal using inverse transform
signal_rdfft = Double1d(N)
rdfft.bt(c, signal_rdfft.getArray())
signal_rdfft = signal_rdfft/N
print "Maximum absolute difference between signal and recreated signal:"
print MAX(ABS(signal-signal_rdfft))
```

API Summary

Jython Syntax

`RealDoubleFFT.ft(<x>, <c>)`

Properties

`Double1d x [INPUT, MANDATORY, default=no default value]`

`Complex1d c [OUTPUT, MANDATORY, default=no default value]`

API details

Properties

`Double1d x [INPUT, MANDATORY, default=no default value]`

Double1d arrays only. Input is modified during the calculation.

`Complex1d c [OUTPUT, MANDATORY, default=no default value]`

Returns a Complex1d

See also

- Section 5.3 in *Scripting Guide*
- FFT
- FFT_PACK
- FFT_PACK_EVEN
- FFT_PACK_ODD
- FFT_AUTO
- Developers Manual: herschel.ia.numeric.toolbox.xform.util.RealDoubleFFT

1.335. Rebin

Full Name:	herschel.ia.numeric.toolbox.interp.Rebin
Alias:	Rebin
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.interp import Rebin
Category	Mathematics/Interpolation

Description

The Rebin class resizes a vector or array to dimensions given by the parameters Di.

The supplied dimensions must be integral multiples or factors of the original dimension. The expansion or compression of each dimension is independent of the others, so that each dimension can be expanded or compressed by a different value. If the dimensions of the desired result are not integer multiples of the original dimensions, use the REGRID function. The SAMPLE keyword: Normally, REBIN uses bilinear interpolation when magnifying and neighborhood averaging when minifying. Set the SAMPLE keyword to use nearest neighbor sampling for both magnification and minification. If Rebin is initialized without error, the algorithm in IDL is strictly followed. If the Rebin is initialized with error, the weighted mean is used at compression when SAMPLE is not set.

Examples

Example 1: Example in Jython. Apply Rebin on Int1d array

```
from herschel.ia.numeric.toolbox.interp import Rebin
from herschel.ia.numeric import Int1d
x = Int1d.range(12)
y = x.apply(Rebin(24))
print y
#[0,0,1,1,2,2,3,3,4,4,5,5,6,6,7,7,8,8,9,9,10,10,11,11]
```

Example 2: Example in Jython. Apply Rebin on Double2d array

```
from herschel.ia.numeric.toolbox.interp import Rebin
from herschel.ia.numeric import Double2d
x = Double2d([[1,1],[2,2]])
d1 = 4
d2 = 4
y=x.apply(Rebin(d1,d2))
print y
# [
# [1.0,1.0,1.0,1.0],
# [1.5,1.5,1.5,1.5],
# [2.0,2.0,2.0,2.0],
# [2.0,2.0,2.0,2.0]
# ]
```

Example 3: In Jython

```
from herschel.ia.numeric.toolbox.interp import Rebin
from herschel.ia.numeric import Int1d
print "In Jython"
x=Int1d.range(12)
y=x.apply(Rebin(24))
#or
y=Rebin(24)(x)
```

Example 4: Expansion

```
from herschel.ia.numeric.toolbox.interp import Rebin
from herschel.ia.numeric import Double1d
print "Expansion"
signal=Double1d.range(10)
dim = 20
rebin = Rebin(dim)
rSignal=rebin(signal)
print rSignal
#[0.0,0.5,1.0,1.5,2.0,2.5,3.0,3.5,4.0,4.5,5.0,5.5,6.0,6.5,7.0,7.5,8.0,8.5,9.0,9.0]
```

Example 5: Use SAMPLE keyword

```
from herschel.ia.numeric.toolbox.interp import Rebin
print "Use SAMPLE keyword"
signal=Double1d.range(10)
dim = 20
rebin = Rebin(dim, Rebin.SAMPLE)
rSignal=rebin(signal)
print rSignal
#[0.0,0.0,1.0,1.0,2.0,2.0,3.0,3.0,4.0,4.0,5.0,5.0,6.0,6.0,7.0,7.0,8.0,8.0,9.0,9.0]
```

Example 6: Calculate error

```
from herschel.ia.numeric.toolbox.interp import Rebin
from herschel.ia.numeric import *
from herschel.ia.numeric.toolbox.basic import *
print "Calculate error"
signal=Double1d.range(20)+1
inError = SQRT(signal)
dim = 10
rebin = Rebin(dim)
rSignal=rebin(signal)
p_error = rebin.getError(inError)
print p_error
 #[0.8660254037844387,1.3228756555322954,1.6583123951777,1.9364916731037087,
#2.179449471770337,2.3979157616563596,2.598076211353316,2.7838821814150108,2.958039891549808,3.1
```

Example 7: Use weight error

```
from herschel.ia.numeric.toolbox.interp import Rebin
from herschel.ia.numeric import *
from herschel.ia.numeric.toolbox.basic import *
print "Use weight error"
signal=Double1d.range(20)+1
inError = SQRT(signal)
dim = 10
rebin = Rebin(dim, inError)
rSignal=rebin(signal)
p_error = rebin.getError()
print p_error
 #[0.6666666666666666,1.7142857142857142,2.7272727272727275,3.7333333333333343,
#4.736842105263158,5.739130434782608,6.7407407407407405,7.741935483870969,8.742857142857142,9.74
```

API Summary

Jython Syntax
<y>=Rebin(<d1[,d2,d3,d4,d5]>)(<x>)
or
<y>=Rebin(<int d[]>)(<x>)
where <x>=input

Jython Syntax
<di> = new dimensions of output array d[] = an array containing the new dimensions <y>=output resampled onto dimensions.

Properties
int **di** [INPUT, MANDATORY, default=no default value]
int[] **x** [INPUT, MANDATORY, default=no default value]
Rebin.SAMPLE **sample** [INPUT, OPTIONAL, default=no default value]

Limitations

The supplied dimensions must be integral multiples or factors of the original dimension. Let f = n/m, the ratio of the size of the original vector, X to the size of the result. 1/f must be an integer if n < m (expansion). f must be an integer if compressing, (n > m).

API details

Properties

int di [INPUT, MANDATORY, default=no default value]

the dimension of the corresponding array index

int[] x [INPUT, MANDATORY, default=no default value]

Array of integer, short, long or double taken the form of Int1d(2,3,4,5)d, Double1d(2,3,4,5)d etc.

Rebin.SAMPLE sample [INPUT, OPTIONAL, default=no default value]

when minifying. Set the SAMPLE to Rebin.SAMPLE to use nearest neighbor sampling for both magnification and minification. Bilinear interpolation gives higher quality results but requires more time

See also

- Developers Manual: herschel.ia.numeric.toolbox.interp.Rebin

History

- `2006-08-30 - first`: version
- `2009-03-23 - error`: propagation (SCR-4833) is added.

1.336. rectangleHistogram

Full Name:	herschel.ia.toolbox.image.RectangleHistogramTask
Alias:	rectangleHistogram
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import RectangleHistogramTask
Category	Images/Display

Description

This is a task to make a histogram of a region of interest on an image which is bounded by a rectangle.

This is a task to make a histogram of a region of interest on an image which is bounded by a rectangle. You must specify the number of bins for the histogram. By default the cut levels of the image will be used to construct the histogram, but it is also possible for you to specify the low and high cut level for the histogram. To define the position of the rectangle, you must specify the pixel or sky coordinates of the lower left corner of the rectangle and its width and height. Note that the sides of the rectangle are aligned with the x- and y-axis.

Example

Example 1: This is an example of how to use the rectangleHistogram :

```
histogram = rectangleHistogram(image = myImage, bins = 200, lowCut = 100.0,
 highCut = 150.0,
                                minX = 100.0, minY = 230.0, widthArcsec = 50.0,
 heightArcsec = 30.0)
histogram = rectangleHistogram(image = myImage, bins = 200, lowCut = 100.0,
 highCut = 150.0,
                                minRa = "05:46:45.9", minDec = "-51:07:57.0",
 widthPixels = 50.0,
                                heightPixels = 30.0)
```

API Summary

Properties
Image **image** [INPUT, MANDATORY, default=None]
Double **lowCut** [INPUT, OPTIONAL, default=NaN]
Double **highCut** [INPUT, OPTIONAL, default=NaN]
Integer **bins** [INPUT, OPTIONAL, default=2000]
Double **minX** [INPUT, OPTIONAL, default=NaN]
Double **minY** [INPUT, OPTIONAL, default=NaN]
String **minRa** [INPUT, OPTIONAL, default="minRa"]
String **minDec** [INPUT, OPTIONAL, default="centerDec"]
Double **widthPixels** [INPUT, OPTIONAL, default=NaN]
Double **heightPixels** [INPUT, OPTIONAL, default=NaN]
Double **widthArcsec** [INPUT, OPTIONAL, default=NaN]
Double **heightArcsec** [INPUT, OPTIONAL, default=NaN]
CircleHistogramProduct **histogram** [OUTPUT, MANDATORY, default=None]

API details

Properties

Image image [INPUT, MANDATORY, default=None]

This is the input image.

Double lowCut [INPUT, OPTIONAL, default=NaN]

This is the low cut level you want to use for the histogram. If this is not specified, the image cut levels will be used.

Double highCut [INPUT, OPTIONAL, default=NaN]

This is the high cut level you want to use for the histogram. If this is not specified, the image cut levels will be used.

Integer bins [INPUT, OPTIONAL, default=2000]

This is the number of bins in the histogram.

Double minX [INPUT, OPTIONAL, default=NaN]

This is the x-pixel-coordinate of the reference corner of the rectangle. The reference corner is defined as the corner of the rectangle with the smallest x- and y-pixel-coordinate.

Double minY [INPUT, OPTIONAL, default=NaN]

This is the y-pixel-coordinate of the reference corner of the rectangle. The reference corner is defined as the corner of the rectangle with the smallest x- and y-pixel-coordinate.

String minRa [INPUT, OPTIONAL, default="minRa"]

This is the right ascension of the reference corner of the rectangle. Using this only makes sense if the input image has a valid Wcs attached to it. The input format for the right ascension is "hh mm ss.s", "hh:mm:ss.s" or "hh.hhh". The reference corner is defined as the corner of the rectangle with the smallest x- and y-pixel-coordinate.

String minDec [INPUT, OPTIONAL, default="centerDec"]

This is the declination of the reference corner of the rectangle. Using this only makes sense if the input image has a valid Wcs attached to it. The input format for the declination is "[+/-]dd mm ss.s", "[+/-]dd:mm:ss.s" or "[+/-]dd.ddd". The reference corner is defined as the corner of the rectangle with the smallest x- and y-pixel-coordinate.

Double widthPixels [INPUT, OPTIONAL, default=NaN]

This is the width of the rectangle in pixels.

Double heightPixels [INPUT, OPTIONAL, default=NaN]

This is the height of the ellipse in pixels.

Double widthArcsec [INPUT, OPTIONAL, default=NaN]

This is the width of the rectangle in arcsec.

Double heightArcsec [INPUT, OPTIONAL, default=NaN]

This is the height of the rectangle in arcsec.

CircleHistogramProduct histogram [OUTPUT, MANDATORY, default=None]
This is the resulting histogram.

See also

- Developers Manual: herschel.ia.toolbox.image.RectangleHistogramTask

1.337. RectangleHistogramProduct

Full Name:	herschel.ia.dataset.image.RectangleHistogramProduct
Type:	Java Class -
Import:	from herschel.ia.dataset.image import RectangleHistogramProduct
Category	Images/Display

Description

A class to deal with the results of a rectangle histogram.

Examples

Example 1: This is how you can store the results table :

```
myProduct.saveResultsTable(filename)
```

Example 2: This is how you can store the curve of growth :

```
myProduct.saveCurveOfGrowth(filename)
```

API Summary

Constructor

`RectangleHistogramProduct`
The constructor of a new RectangleHistogramProduct.

Methods

`setUpperLeftCorner (double upperLeftX, double upperLeftY)`
Sets the upper left corner for this RectangleHistogramProduct to the given pixel coordinates.

`setUpperLeftCorner (double upperLeftX, double upperLeftY, String upperLeftRA, String upperLefttDec)`
Sets the upper left corner for this RectangleHistogramProduct to the given pixel and sky coordinates.

`setDimensions (double widthPixels, double heightPixels)`
Sets the dimensions for this RectangleHistogramProduct to the given dimensions in pixels.

`setDimensions (double widthPixels, double heightPixels, double widthArcsec, double heightArcsec)`
Sets the dimensions for this RectangleHistogramProduct to the given dimensions in pixels and arcsec.

`Double1d` **`getUpperLeftCornerPixelCoordinates`**
Returns the upper left corner for this RectangleHistogramProduct in pixel coordinates.

`String1d` **`getUpperLeftCornerSkyCoordinates`**
Returns the upper left corner for this RectangleHistogramProduct in sky coordinates.

`double` **`getWidthPixels`**
Returns the width for this RectangleHistogramProduct in pixels.

`double` **`getWidthArcsec`**

Methods
Returns the width for this RectangleHistogramProduct in arcsec.
double **getHeightPixels**
Returns the height for this RectangleHistogramProduct in pixels.
double **getHeightArcsec**
Returns the height for this RectangleHistogramProduct in arcsec.

API Details

Constructor

RectangleHistogramProduct
The constructor of a new RectangleHistogramProduct.

Methods

setUpperLeftCorner (double upperLeftX, double upperLeftY)

Sets the upper left corner for this RectangleHistogramProduct to the given pixel coordinates.

Arguments

`double` **upperLeftX** `[INPUT, MANDATORY, default=no default value]`

The x-pixel-coordinate of the upper left corner, as double

`double` **upperLeftY** `[INPUT, MANDATORY, default=no default value]`

The y-pixel-coordinate of the upper left corner, as double

setUpperLeftCorner (double upperLeftX, double upperLeftY, String upperLeftRA, String upperLefttDec)

Sets the upper left corner for this RectangleHistogramProduct to the given pixel and sky coordinates.

Arguments

`double` **upperLeftX** `[INPUT, MANDATORY, default=no default value]`

The x-pixel-coordinate of the upper left corner, as double

`double` **upperLeftY** `[INPUT, MANDATORY, default=no default value]`

The y-pixel-coordinate of the upper left corner, as double

`String` **upperLeftRA** `[INPUT, MANDATORY, default=no default value]`

The right ascension of the upper left corner, as String

`String` **upperLefttDec** `[INPUT, MANDATORY, default=no default value]`

The declination of the upper left corner, as String

setDimensions (double widthPixels, double heightPixels)

Sets the dimensions for this RectangleHistogramProduct to the given dimensions in pixels.

Arguments

`double` **widthPixels** `[INPUT, MANDATORY, default=no default value]`

The width in pixels, as double

`double` **heightPixels** `[INPUT, MANDATORY, default=no default value]`

The height in pixels, as double

setDimensions (double widthPixels, double heightPixels, double widthArcsec, double heightArcsec)

Sets the dimensions for this RectangleHistogramProduct to the given dimensions in pixels and arcsec.

Arguments

`double` **widthPixels** `[INPUT, MANDATORY, default=no default value]`

The width in pixels, as double

`double` **heightPixels** `[INPUT, MANDATORY, default=no default value]`

The height in pixels, as double

`double` **widthArcsec** `[INPUT, MANDATORY, default=no default value]`

The width in arcsec, as double

`double` **heightArcsec** `[INPUT, MANDATORY, default=no default value]`

The height in arcsec, as double

Double1d **getUpperLeftCornerPixelCoordinates**

Returns the upper left corner for this RectangleHistogramProduct in pixel coordinates.

Return

Double1d

Returns the upper left corner for this RectangleHistogramProduct in pixel coordinates.

String1d **getUpperLeftCornerSkyCoordinates**

Returns the upper left corner for this RectangleHistogramProduct in sky coordinates.

Return

String1d

Returns the upper left corner for this RectangleHistogramProduct in sky coordinates.

double **getWidthPixels**

Returns the width for this RectangleHistogramProduct in pixels.

Return

double

Returns the width for this RectangleHistogramProduct in pixels.

double **getWidthArcsec**

Returns the width for this RectangleHistogramProduct in arcsec.

Return

double

Returns the width for this RectangleHistogramProduct in arcsec.

double **getHeightPixels**

Returns the height for this RectangleHistogramProduct in pixels.

Return

double

Returns the height for this RectangleHistogramProduct in pixels.

double **getHeightArcsec**
Returns the height for this RectangleHistogramProduct in arcsec.
Return
double
Returns the height for this RectangleHistogramProduct in arcsec.

See also

- Developers Manual: herschel.ia.dataset.image.RectangleHistogramProduct

1.338. rectangularSkyAperturePhotometry

Full Name:	herschel.ia.toolbox.image.RectangularSkyAperturePhotTask
Alias:	rectangularSkyAperturePhotometry
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import RectangularSkyAperturePhotTask
Category	Astronomical utilities/Photometry

Description

This is a task for aperture photometry for a rectangular target

aperture. That is, you want the pixels within a rectangular sky aperture to estimate the sky. The user needs to specify the source location (i.e. the center of the rectangular aperture) either in pixel or sky coordinates, the location of the rectangle and which of the five sky estimation algorithms must be used. Note that the sides of the rectangle are aligned with the x-and y-axisand that only the unflagged pixels are used for the aperture photometry. The five sky estimation algorithms that are offered are :

- average (0) : The average intensity of the pixels contributing to the flux in sky aperture. In case fractional pixels should be used, the pixels are weighted according to their overlap with the sky aperture. Else, we use the average of the intensities of the contributing pixels.
- median (1) : The median intensity of the pixels contributing to the flux in the sky aperture. In case fractional pixels should be used, the pixels are weighted according to their overlap with the sky aperture. Else, we use the median of the intensities of the contributing pixels.
- mean-median (2) : Only pixels which are less than 1.5 times the standard deviation (w.r.t. the average intensity) away from the median are used and the mean-median is the average of the remaining pixels. The same remarks hold concerning the fractional pixels.
- synthetic mode (3) : The mean-median algorithm is repeated in an iterative process up to ten times or until the average does not change anymore. The mean median is then (3 * median) - (2 * average) of the remaining intensities. The same remarks hold concerning the fractional pixels.
- daophot (4) : Sky estimation algorithm as adapted from the IDL mmm.pro routine. Have a look at http://idlastro.gsfc.nasa.gov/ftp/pro/idlphot/mmm.pro

The background corrected flux density in the aperture is calculated using the following pseudo code:

```
Target (bg subtr) = Total(Jy) - Background(Jy/pixel) * #pixels(in the target
 aperture)
```

where `Total(Jy)` is the total flux density in the user selected aperture, `Background (Jy/pixel)` is the average background/sky estimation in the annular ring, calculated with the selected algorithm and `#pixels` is the number of pixels in the target aperture. By default the sky estimation uses fractional pixels but there is an option to switch to entire pixels. The calculation of the error on the fluxes is calculated in the same way as in the aper.pro routine of IDL. Three factors contribute to the error on the target flux :

- scatter in sky values
- random photon noise
- uncertainty in mean sky brightness

The error for the target flux including the background is defined as the sqrt of the (absolute value of the) total flux in the target aperture (including the background). The error for the sky is defined as the

standard deviation on the sky value. Note that this standard deviation is calculated only with the sky intensities that are used to calculate the sky intensity (as there are sky estimation algorithms that reject sky values). The squared error for the target flux is thus :

- the absolute value of the target flux of which the sky contribution has been subtracted
- the standard deviation of the sky value * the surface of the target aperture
- the squared standard deviation on the sky value

Units of the output: The unit of the integrated flux is the unit of the image, multiplied with "pixel". So aperture photometry on an image in Jy/pixel will yield an integrated flux in Jy. For images in Jy/beam, the integrated flux will be expressed in Jy/beam*pixel, if you don't convert the image to Jy/pixel yourself beforehand. This can be done with the ConvertImageUnitTask. Additionally, if the input unit is MJy/sr, the output will still be in Jy (the task does the conversion).

Example

Example 1: This is an example how you can use the

```
rectangularSkyAperturePhotometry : result =
rectangularSkyAperturePhotometry(image = myImage, minX =
133.5, minY = 82.0, widthArcsec = 60.0, heightArcsec = 41.5,
centerX = 250.4, centerY = 100.5, radiusArcsec = 5.0,
fractional = 1, algorithm = 4, centroid = True) result =
rectangularSkyAperturePhotometry(image = myImage, minRa =
"05:47:35.5", minDec = "-51:09:09.6", widthPixels = 60.0,
heightPixels = 41.5, centerRa = "05:46:45.9", centerDec =
"-51:07:57.0", radiusPixels = 5.0, fractional = 0, algorithm
= 0)
```

API Summary

Properties
Image **image** [INPUT, MANDATORY, default=None This is the image on which]
Double **centerX** [INPUT, OPTIONAL, default=NaN This is the]
Double **centerY** [INPUT, OPTIONAL, default=NaN This is the]
String **centerRa** [INPUT, OPTIONAL, default="centerRa" This is the right]
String **centerDec** [INPUT, OPTIONAL, default="centerDec" This is]
Double **radiusPixels** [INPUT, OPTIONAL, default=NaN This is the target]
Double **radiusArcsec** [INPUT, OPTIONAL, default=NaN This the target radius]
boolean **centroid** [INPUT, OPTIONAL, default=false This parameter]
Integer **algorithm** [INPUT, OPTIONAL, default=4 The sky estimation]
Double **minX** [INPUT, OPTIONAL, default=NaN This is the x-pixel-co-ordinate]
Double **minY** [INPUT, OPTIONAL, default=NaN This is the y-pixel-co-ordinate]
String **minRa** [INPUT, OPTIONAL, default="upperLeftRa" This is the right]

Properties
String **minDec** [INPUT, OPTIONAL, default="upperLeftDec" This is the]
Double **widthPixels** [INPUT, OPTIONAL, default=NaN This is the width of]
Double **heightPixels** [INPUT, OPTIONAL, default=NaN This is the height of]
Double **widthArcsec** [INPUT, OPTIONAL, default=NaN This is the width of]
Double **heightArcsec** [INPUT, OPTIONAL, default=NaN This is the height of]
boolean **fractional** [INPUT, OPTIONAL, default=true This indicates whether]
RectangularSkyAperturePhotometryProduct **result** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Image image [INPUT, MANDATORY, default=None This is the image on which]

to perform aperture photometry.

Double centerX [INPUT, OPTIONAL, default=NaN This is the]

x-pixel-coordinate of the target center.

Double centerY [INPUT, OPTIONAL, default=NaN This is the]

y-pixel-coordinate of the target center.

String centerRa [INPUT, OPTIONAL, default="centerRa" This is the right]

ascension of the target center. Using this only makes sense if the input image has a valid Wcs attached to it. The input format for the right ascension is "hh mm ss.s", "hh:mm:ss.s" or "dd.d-dd".

String centerDec [INPUT, OPTIONAL, default="centerDec" This is]

declination of the target center. Using this only makes sense if the input image has a valid Wcs attached to it. The input format for the declination is "[+/-]dd mm ss.s", "[+/-]dd:mm:ss.s" or "[+/-]dd.ddd".

Double radiusPixels [INPUT, OPTIONAL, default=NaN This is the target]

radius (i.e. the radius of the circular target aperture) in pixels.

Double radiusArcsec [INPUT, OPTIONAL, default=NaN This the target radius]

in arcsec. Using this only makes sense if the input image has a valid Wcs attached to it and the pixel scaling is the same along both axis (in absolute value).

boolean centroid [INPUT, OPTIONAL, default=false This parameter]

indicates whether centroiding on the target is needed.

Integer algorithm [INPUT, OPTIONAL, default=4 The sky estimation]

algorithm. Possible values are 0 (average), 1 (median), 2 (mean-median), 3 (sythetic mode) and 4 (daophot).

Double minX [INPUT, OPTIONAL, default=NaN This is the x-pixel-coordinate]

of the reference corner of the rectangular sky aperture. The reference corner is defined as the corner of the rectangular sky aperture with the smallest x- and y-pixel-coordinate.

Double minY [INPUT, OPTIONAL, default=NaN This is the y-pixel-coordinate]

of the reference corner of the rectangular sky aperture. The reference corner is defined as the corner of the rectangular sky aperture with the smallest x- and y-pixel-coordinate.

String minRa [INPUT, OPTIONAL, default="upperLeftRa" This is the right]

ascension of the reference corner of the rectangular sky aperture. The reference corner is defined as the corner of the rectangular sky aperture with the smallest x- and y-pixel-coordinate. Using this only makes sense if the input image has a valid Wcs attached to it. The input format for the right ascension is "hh mm ss.s", "hh:mm:ss.s" or "dd.ddd".

String minDec [INPUT, OPTIONAL, default="upperLeftDec" This is the]

declination of the reference corner of the rectangular sky aperture. The reference corner is defined as the corner of the rectangular sky aperture with the smallest x- and y-pixel-coordinates. Using this only makes sense if the input image has a valid Wcs attached to it. The input format for the declination is "[+/-]dd mm ss.s", "[+/-]dd:mm:ss.s" or "[+/-]dd.ddd".

Double widthPixels [INPUT, OPTIONAL, default=NaN This is the width of]

the rectangular sky aperture in pixels.

Double heightPixels [INPUT, OPTIONAL, default=NaN This is the height of]

the rectangular sky aperture in pixels.

Double widthArcsec [INPUT, OPTIONAL, default=NaN This is the width of]

the rectangular sky aperture in arcsec. Using this only makes sense if the input image has a valid Wcs attached to it and the pixel scaling is the same along both axis (in absolute value).

Double heightArcsec [INPUT, OPTIONAL, default=NaN This is the height of]

the rectangular sky aperture in arcsec. Using this only makes sense if the input image has a valid Wcs attached to it and the pixel scaling is the same along both axis (in absolute value).

boolean fractional [INPUT, OPTIONAL, default=true This indicates whether]

you want to use fractional pixel to estimate the sky intensity.

RectangularSkyAperturePhotometryProduct result [OUTPUT, MANDATORY, default=no default value]

MANDATORY, None This is the result of the aperture photometry. Here you can find the flux in the target aperture (with and without backgroud contribution) and the sky aperture, the number of pixels in the apertures and the error on the fluxes. From the curve of growth the user can judge whether a good value for the target radius was chosen (under the assumption that the given sky value was accurate enough).

See also

- Developers Manual: herschel.ia.toolbox.image.RectangularSkyAperturePhotTask

1.339. RectangularSkyAperturePhotometryProduct

Full Name:	herschel.ia.dataset.image.RectangularSkyAperturePhotometryProduct
Type:	Java Class -
Import:	from herschel.ia.dataset.image import RectangularSkyAperturePhotometryProduct
Category	Astronomical utilities/Photometry

Description

This is a class to deal with the results of aperture photometry with a circular target aperture and an rectangular sky aperture.

This class is used to store the output of the {@link herschel.ia.toolbox.image.RectangularSkyAperturePhotTask RectangularSkyAperturePhotometryTask}. Aperture specific information is stored in metadata, all other data (results of calculations) are stored in TableDatasets.

Example

Example 1: Script demonstrating usage of a RectangularSkyAperturePhotometryProduct

```
# get hold of the pixel coordinates of the target center:
coords = product.getTargetCenterPixelCoordinates()
row = coords[1]
column = coords[0]
# get hold of the sky coordinates of the target center:
coords = product.getTargetCenterSkyCoordinates()
ra = coords[0]
dec = coords[1]
# get hold of the target radius in pixels:
print product.getTargetRadiusPixels()
# get hold of the target radius in arcsec:
print product.getTargetRadiusArcsec()
# get hold of the pixel coordinates of the upper left corners of the sky
 rectangle:
coords = product.getUpperLeftCornerPixelCoordinates()
row = coords[1]
column = coords[0]
# get hold of the sky coordinates of the upper left corner of the sky
 rectangle:
coords = product.getUpperLeftCornerSkyCoordinates()
ra = coords[0]
dec = coords[1]
# get hold of the width of the sky rectangle in pixels:
print product.getWidthPixels()
# get hold of the height of the sky rectangle in pixels :
print product.getHeightPixels()
# get hold of the width of the sky rectangle in arcec:
print product.getWidthArcsec()
# get hold of the height of the sky rectangle in arcsec:
print product.getHeightArcsec()
# check which sky estimation algorithm was used:
print product.getAlgorithm()
# check whether you are using fractional or entire pixels:
print product.getPixels()
# get hold of the intensity unit (i.e. the unit of the image):
print product.getUnit()
# get hold of the (integrated) flux unit:
print product.getFluxUnit()
# get hold of the results table and save it (can be saved with the
 asciiTableWriter):
table = product.getTable()
# get hold of the total flux in the target aperture (incl. background):
```

Example 1: Script demonstrating usage of a RectangularSkyAperturePhotometryProduct

```
print product.getTargetPlusSkyTotal()
# get hold of the total flux in the target aperture (background subtracted):
print product.getTargetTotal()
# get hold of the curve of growth as a table (can be saved with the
 asciiTableWriter):
table = product.getCurveOfGrowth()
# plot the curve of growth:
plot = PlotXY(product.getGrowthRadius(), product.getGrowthFlux())
plot.setXtitle("Target radius [pixels]");
plot.setYtitle("Target flux (sky subtr.) [" + product.getFluxUnit() + "]")
plot.getSubPlot(0).getLayer(0).getStyle().setLine(0)
```

API Summary

Constructor
RectangularSkyAperturePhotometryProduct The constructor of a new RectangularSkyAperturePhotometryProduct.

Methods
setUpperLeftCorner (double upperLeftX, double upperLeftY) Setting the upper left corner of the sky rectangle in pixel coordinates.
setUpperLeftCorner (double upperLeftX, double upperLeftY, String upperLeftRA, String upperLeftDec) Setting the upper left corner of the sky rectangle in pixel and sky coordinates.
setDimensions (double widthPixels, double heightPixels) Setting the dimensions of the sky rectangle to the given dimensions in pixels.
setDimensions (double widthPixels, double heightPixels, double widthArcsec, double heightArcsec) Setting the dimensions of the sky rectangle to the given dimensions in pixels and arcsec.
Double1d **getUpperLeftCornerPixelCoordinates** Returns the upper left corner of the sky rectangle in pixel coordinates.
String1d **getUpperLeftCornerSkyCoordinates** Returns the upper left corner of the sky rectangle in sky coordinates.
double **getWidthPixels** Returns the width of the sky rectangle in pixels.
double **getWidthArcsec** Returns the width of the sky rectangle in arcsec.
double **getHeightPixels** Returns the height of the sky rectangle in pixels.
double **getHeightArcsec** Returns the height of the sky rectangle in arcsec.

API Details

Constructor

RectangularSkyAperturePhotometryProduct
The constructor of a new RectangularSkyAperturePhotometryProduct.

RectangularSkyAperturePhotometryProduct

A new SkyAperturePhotometryProduct is constructed.

Methods

setUpperLeftCorner (double upperLeftX, double upperLeftY)

Setting the upper left corner of the sky rectangle in pixel coordinates.

Sets the upper left corner of the sky rectangle for this RectangularSkyAperturePhotometryProduct to the given pixel coordinates.

Arguments

`double` **upperLeftX** `[INPUT, MANDATORY, default=no default value]`

The x-pixel-coordinate for the upper left corner of the sky rectangle, as double

`double` **upperLeftY** `[INPUT, MANDATORY, default=no default value]`

The y-pixel-coordinate for the upper left corner of the sky rectangle, as double

setUpperLeftCorner (double upperLeftX, double upperLeftY, String upperLeftRA, String upperLeftDec)

Setting the upper left corner of the sky rectangle in pixel and sky coordinates.

Sets the upper left corner for this RectangularSkyAperturePhotometryProduct.

Arguments

`double` **upperLeftX** `[INPUT, MANDATORY, default=no default value]`

The x-pixel-coordinate for the upper left corner of the sky rectangle, as double

`double` **upperLeftY** `[INPUT, MANDATORY, default=no default value]`

The y-pixel-coordinate for the upper left corner of the sky rectangle, as double

`String` **upperLeftRA** `[INPUT, MANDATORY, default=no default value]`

The right ascension for the upper left corner of the sky rectangle, as String

`String` **upperLeftDec** `[INPUT, MANDATORY, default=no default value]`

The declination for the upper left corner of the sky rectangle, as String

setDimensions (double widthPixels, double heightPixels)

Setting the dimensions of the sky rectangle to the given dimensions in pixels.

Sets the dimensions of the sky rectangle for this RectangularSkyAperturePhotometryProduct to the given dimensions in pixels.

Arguments

`double` **widthPixels** `[INPUT, MANDATORY, default=no default value]`

The width of the sky rectangle in pixels, as double

`double` **heightPixels** `[INPUT, MANDATORY, default=no default value]`

The height of the sky rectangle in pixels, as double

setDimensions (double widthPixels, double heightPixels, double widthArcsec, double heightArcsec)

Setting the dimensions of the sky rectangle to the given dimensions in pixels and arcsec.

Sets the dimensions of the sky rectangle for this RectangularSkyAperturePhotometryProduct to the given dimensions in pixels and arcsec.

Arguments

setDimensions (double widthPixels, double heightPixels, double widthArcsec, double heightArcsec)

double **widthPixels** [INPUT, MANDATORY, default=no default value]

The width of the sky rectangle in pixels, as double

double **heightPixels** [INPUT, MANDATORY, default=no default value]

The height of the sky rectangle in pixels, as double

double **widthArcsec** [INPUT, MANDATORY, default=no default value]

The width of the sky rectangle in arcsec, as double

double **heightArcsec** [INPUT, MANDATORY, default=no default value]

The width of the sky rectangle in arcsec, as double

Double1d **getUpperLeftCornerPixelCoordinates**

Returns the upper left corner of the sky rectangle in pixel coordinates.

Returns the upper left corner of the sky rectangle for this RectangularSkyAperturePhotometryProduct in pixel coordinates.

Return

Double1d

The upper left corner of the sky rectangle for this RectangularSkyAperturePhotometryProduct in pixel coordinates.

String1d **getUpperLeftCornerSkyCoordinates**

Returns the upper left corner of the sky rectangle in sky coordinates.

Returns the upper left corner of the sky rectangle for this RectangularSkyAperturePhotometryProduct in sky coordinates.

Return

String1d

The upper left corner for this RectangularSkyAperturePhotometryProduct in sky coordinates.

double **getWidthPixels**

Returns the width of the sky rectangle in pixels.

Returns the width of the sky rectangle for this RectangularSkyAperturePhotometryProduct in pixels.

Return

double

The width of the sky rectangle for this RectangularSkyAperturePhotometryProduct in pixels.

double **getWidthArcsec**

Returns the width of the sky rectangle in arcsec.

Returns the width of the sky rectangle for this RectangularSkyAperturePhotometryProduct in arcsec.

Return

double

The width of the sky rectangle for this RectangularSkyAperturePhotometryProduct in arcsec.

***double* getHeightPixels**

Returns the height of the sky rectangle in pixels.

Returns the height of the sky rectangle for this RectangularSkyAperturePhotometryProduct in pixels.

Return

double

The height of the sky rectangle for this RectangularSkyAperturePhotometryProduct in pixels.

***double* getHeightArcsec**

Returns the height of the sky rectangle in arcsec.

Returns the height of the sky rectangle for this RectangularSkyAperturePhotometryProduct in arcsec.

Return

double

The height of the sky rectangle for this RectangularSkyAperturePhotometryProduct in arcsec.

See also

- Developers Manual: herschel.ia.dataset.image.RectangularSkyAperturePhotometryProduct

1.340. Regrid

Full Name:	herschel.ia.numeric.toolbox.interp.Regrid
Alias:	Regrid
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.interp import Regrid
Category	Mathematics/Interpolation

Description

The Regrid class shrinks or expands the size of an array by an arbitrary amount.

This class is similar to "Rebin" in that it can resize a one, two, or three dimensional array. "Rebin", however, requires that the new array size must be an integer multiple of the original size. Regrid will resize an array to any arbitrary size (Rebin is somewhat faster, however). Rebin averages multiple points when shrinking an array, while Regrid just resamples the array. The returned array has the same number of dimensions as the original array and is of the same data type.

Example

Example 1: Use Regrid

```
#1-d example: resize a 1d array 10 elements long to one that is 6 elements long
from herschel.ia.numeric.toolbox.interp import Regrid
x = Double1d([0,1,2,3,4,5,6,7,8,9])
y = Regrid(6)(x) # default allowExtrapolation is true.
print x # [0.0,1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0]
print y #
 #[0.0,1.6666666666666667,3.3333333333333335,5.0,6.666666666666667,8.333333333333334]
#1-d example: resize a 1d array 10 elements long to one that is 6 elements long
x = Double1d([0,1,2,3,4,5,6,7,8,9])
y= Regrid(6,Regrid.MINUS_ONE)(x) #allowExtrapolation is false. IDL
 CONGRID(x,6,/MINUS_ONE,/INTERP)
print x # [0.0,1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0]
print y # [0.0,1.8,3.6,5.4,7.2,9.0]
# do the same, but this time use a different scheme
y = Regrid(6,Regrid.NEAREST_NEIGHBOR)(x)
print y #[0.0,2.0,3.0,5.0,7.0,8.0]
#resize a 2d array 2x3 elements long to  one that is 5x8 elements long
x = Double2d([[2.2, 3.1, 11.9],[109.2, 135.7, 210.8]])
y = Regrid(5,8,Regrid.LINEAR)(x)
#note the first column resamples between 2.2 and 173.4.
#i.e. going from 2 points to 5. this can be checked by :
print y[:,0]
 #2.2,45.00000000000001,87.80000000000001,130.60000000000002,173.4]
# or concentrating on the first row, which goes
# from 2.2 to 11.9 in three steps, the out is resampled to 8 points.
print y[0:]
#[2.2,2.4571428571428573,2.7142857142857144,2.9714285714285715,
# 4.357142857142857,6.87142857142857,9.385714285714286,11.9]
#[2.2,2.5375,2.875,4.2,7.5,10.8,14.100000000000001,17.4,45.00000000000001,49.17750000000001,53.
#
 87.045,100.29000000000002,113.535,87.80000000000001,95.81750000000001,103.835,116.91,140.1,163
#
 186.48000000000002,209.67000000000002,130.60000000000002,142.45750000000004,154.315,173.265,20
#
 239.53500000000005,272.6700000000001,305.80500000000006,173.4,189.09750000000003,204.795,229.6
# 272.70000000000005,315.78000000000003,358.86,401.94000000000005]
```

API Summary

Constructors
Regrid (int[] d)
Regrid (int[] d, String interpMethod)
Regrid (int[] d, String interpMethod, String allowExtrapolation)
Regrid (int d1)
Regrid (int d1, String interpMethod)
Regrid (int d1, String interpMethod, String allowExtrapolation)
Regrid (int d1, int d2)
Regrid (int d1, int d2, String interpMethod)
Regrid (int d1, int d2, String interpMethod, String allowExtrapolation)
Regrid (int d1, int d2, int d3)
Regrid (int d1, int d2, int d3, String interpMethod)
Regrid (int d1, int d2, int d3, String interpMethod, String allowExtrapolation)
Regrid (int d1, int d2, int d3, int d4)
Regrid (int d1, int d2, int d3, int d4, String interpMethod)
Regrid (int d1, int d2, int d3, int d4, String interpMethod, String allowExtrapolation)
Regrid (int d1, int d2, int d3, int d4, int d5)
Regrid (int d1, int d2, int d3, int d4, int d5, String interpMethod)
Regrid (int d1, int d2, int d3, int d4, int d5, String interpMethod, String allowExtrapolation)

Limitations

The x, INPUT, array of Int(n)d, Short(n)d, Long(n)d, Float(n)d or Double(n)d, (1<= n <=5)

Miscellaneous

Synopsis:

resample rank=d.length array to new dimensions in d.

y = Regrid(d[])(x)

resamples a one to five dimension array using default (linear) interpolation.

y = Regrid(d1,...,d5)(x)

resamples a one to five dimension array using linear interpolation.

y = Regrid(d1,...,d5,Regrid.LINEAR)(x)

resamples an array using nearest neighbor interpolation.

y = Regrid(d1,...,d5,Regrid.NEAREST_NEIGHBOR)(x)

resamples an array using cubic spline interpolation.

y = Regrid(d1,...,d5,Regrid.CUBIC_SPLINE)(x)

allows Extrapolation.

y = Regrid(d1,...,d5,Regrid.LINEAR,Regrid.MINUS_ONE)(x)

API Details

Constructors

`Regrid (int[] d)`

Argument

`int[] d [INPUT, MANDATORY, default=no default value]`

array of Int(n)d, Short(n)d, Long(n)d, Float(n)d or Double(n)d, (1<= n <=5), MANDATORY.

Error

`UnsupportedOperationException`

Bad dimension array! It should be 1<= d <=5.

`Regrid (int[] d, String interpMethod)`

Arguments

`int[] d [INPUT, MANDATORY, default=no default value]`

array of Int(n)d, Short(n)d, Long(n)d, Float(n)d or Double(n)d, (1<= n <=5), MANDATORY.

`String interpMethod [INPUT, OPTIONAL, default=Regrid.LINEAR]`

allowable interpolation schemes are: CUBLIC_SPLINE, NEAREST_NEIGHBOR, and LINEAR (which is the default).

Error

`UnsupportedOperationException`

Bad dimension array! It should be 1<= d <=5.

`Regrid (int[] d, String interpMethod, String allowExtrapolation)`

Arguments

`int[] d [INPUT, MANDATORY, default=no default value]`

array of Int(n)d, Short(n)d, Long(n)d, Float(n)d or Double(n)d, (1<= n <=5), MANDATORY.

`String interpMethod [INPUT, OPTIONAL, default=Regrid.LINEAR]`

allowable interpolation schemes are: CUBLIC_SPLINE, NEAREST_NEIGHBOR, and LINEAR (which is the default).

`String allowExtrapolation [INPUT, OPTIONAL, default=Regrid.EXTRAPOLATION]`

default is no extrapolation allowed.

Error

`UnsupportedOperationException`

Bad dimension array! It should be 1<= d <=5.

`Regrid (int d1)`

Regrid (int d1)

Argument

`int` **d1** [INPUT, MANDATORY, default=no default value]

d1 input integer

Regrid (int d1, String interpMethod)

Arguments

`int` **d1** [INPUT, MANDATORY, default=no default value]

d1 input integer

`String` **interpMethod** [INPUT, OPTIONAL, default=Regrid.LINEAR]

allowable interpolation schemes are: CUBLIC_SPLINE, NEAREST_NEIGHBOR, and LINEAR (which is the default).

Regrid (int d1, String interpMethod, String allowExtrapolation)

Arguments

`int` **d1** [INPUT, MANDATORY, default=no default value]

d1 input integer

`String` **interpMethod** [INPUT, OPTIONAL, default=Regrid.LINEAR]

allowable interpolation schemes are: CUBLIC_SPLINE, NEAREST_NEIGHBOR, and LINEAR (which is the default).

`String` **allowExtrapolation** [INPUT, OPTIONAL, default=Regrid.EXTRAPOLATION]

default is no extrapolation allowed.

Regrid (int d1, int d2)

Arguments

`int` **d1** [INPUT, MANDATORY, default=no default value]

d1 input integer

`int` **d2** [INPUT, MANDATORY, default=no default value]

d2 input integer

Regrid (int d1, int d2, String interpMethod)

Arguments

`int` **d1** [INPUT, MANDATORY, default=no default value]

d1 input integer

`int` **d2** [INPUT, MANDATORY, default=no default value]

d2 input integer

`String` **interpMethod** [INPUT, OPTIONAL, default=Regrid.LINEAR]

allowable interpolation schemes are: CUBLIC_SPLINE, NEAREST_NEIGHBOR, and LINEAR (which is the default)

Regrid (int d1, int d2, String interpMethod, String allowExtrapolation)

Arguments

`int` **d1** [INPUT, MANDATORY, default=no default value]

Regrid (int d1, int d2, String interpMethod, String allowExtrapolation)

d1 input integer

`int` **d2** [INPUT, MANDATORY, default=no default value]

d2 input integer

`String` **interpMethod** [INPUT, OPTIONAL, default=Regrid.LINEAR]

allowable interpolation schemes are: CUBLIC_SPLINE, NEAREST_NEIGHBOR, and LINEAR (which is the default).

`String` **allowExtrapolation** [INPUT, OPTIONAL, default=Regrid.EXTRAPOLATION]

default is no extrapolation allowed.

Regrid (int d1, int d2, int d3)

Arguments

`int` **d1** [INPUT, MANDATORY, default=no default value]

d1 input integer

`int` **d2** [INPUT, MANDATORY, default=no default value]

d2 input integer

`int` **d3** [INPUT, MANDATORY, default=no default value]

d3 input integer

Regrid (int d1, int d2, int d3, String interpMethod)

Arguments

`int` **d1** [INPUT, MANDATORY, default=no default value]

d1 input integer

`int` **d2** [INPUT, MANDATORY, default=no default value]

d2 input integer

`int` **d3** [INPUT, MANDATORY, default=no default value]

d3 input integer

`String` **interpMethod** [INPUT, OPTIONAL, default=Regrid.LINEAR]

allowable interpolation schemes are: CUBLIC_SPLINE, NEAREST_NEIGHBOR, and LINEAR (which is the default).

Regrid (int d1, int d2, int d3, String interpMethod, String allowExtrapolation)

Arguments

`int` **d1** [INPUT, MANDATORY, default=no default value]

d1 input integer

`int` **d2** [INPUT, MANDATORY, default=no default value]

d2 input integer

`int` **d3** [INPUT, MANDATORY, default=no default value]

d3 input integer

`String` **interpMethod** [INPUT, OPTIONAL, default=Regrid.LINEAR]

allowable interpolation schemes are: CUBLIC_SPLINE, NEAREST_NEIGHBOR, and LINEAR (which is the default).

Regrid (int d1, int d2, int d3, String interpMethod, String allowExtrapolation)

String **allowExtrapolation** [INPUT, OPTIONAL, default=Regrid.EXTRAPOLATION]

default is no extrapolation allowed.

Regrid (int d1, int d2, int d3, int d4)

Arguments

int **d1** [INPUT, MANDATORY, default=no default value]

d1 input integer

int **d2** [INPUT, MANDATORY, default=no default value]

d2 input integer

int **d3** [INPUT, MANDATORY, default=no default value]

d3 input integer

int **d4** [INPUT, MANDATORY, default=no default value]

d4 input integer

Regrid (int d1, int d2, int d3, int d4, String interpMethod)

Arguments

int **d1** [INPUT, MANDATORY, default=no default value]

d1 input integer

int **d2** [INPUT, MANDATORY, default=no default value]

d2 input integer

int **d3** [INPUT, MANDATORY, default=no default value]

d3 input integer

int **d4** [INPUT, MANDATORY, default=no default value]

d4 input integer

String **interpMethod** [INPUT, OPTIONAL, default=Regrid.LINEAR]

allowable interpolation schemes are: CUBLIC_SPLINE, NEAREST_NEIGHBOR, and LINEAR (which is the default).

Regrid (int d1, int d2, int d3, int d4, String interpMethod, String allowExtrapolation)

Arguments

int **d1** [INPUT, MANDATORY, default=no default value]

d1 input integer

int **d2** [INPUT, MANDATORY, default=no default value]

d2 input integer

int **d3** [INPUT, MANDATORY, default=no default value]

d3 input integer

int **d4** [INPUT, MANDATORY, default=no default value]

d4 input integer

String **interpMethod** [INPUT, OPTIONAL, default=Regrid.LINEAR]

allowable interpolation schemes are: CUBLIC_SPLINE, NEAREST_NEIGHBOR, and LINEAR (which is the default).

Regrid (int d1, int d2, int d3, int d4, String interpMethod, String allowExtrapolation)

String **allowExtrapolation** [INPUT, OPTIONAL, default=Regrid.EXTRAPOLATION]

default is no extrapolation allowed.

Regrid (int d1, int d2, int d3, int d4, int d5)

Arguments

int **d1** [INPUT, MANDATORY, default=no default value]

d1 input integer

int **d2** [INPUT, MANDATORY, default=no default value]

d2 input integer

int **d3** [INPUT, MANDATORY, default=no default value]

d3 input integer

int **d4** [INPUT, MANDATORY, default=no default value]

d4 input integer

int **d5** [INPUT, MANDATORY, default=no default value]

d5 input integer

Regrid (int d1, int d2, int d3, int d4, int d5, String interpMethod)

Arguments

int **d1** [INPUT, MANDATORY, default=no default value]

d1 input integer

int **d2** [INPUT, MANDATORY, default=no default value]

d2 input integer

int **d3** [INPUT, MANDATORY, default=no default value]

d3 input integer

int **d4** [INPUT, MANDATORY, default=no default value]

d4 input integer

int **d5** [INPUT, MANDATORY, default=no default value]

d5 input integer

String **interpMethod** [INPUT, OPTIONAL, default=Regrid.LINEAR]

allowable interpolation schemes are: CUBLIC_SPLINE, NEAREST_NEIGHBOR, and LINEAR (which is the default).

Regrid (int d1, int d2, int d3, int d4, int d5, String interpMethod, String allowExtrapolation)

Arguments

int **d1** [INPUT, MANDATORY, default=no default value]

d1 input integer

int **d2** [INPUT, MANDATORY, default=no default value]

d2 input integer

int **d3** [INPUT, MANDATORY, default=no default value]

d3 input integer

Regrid (int d1, int d2, int d3, int d4, int d5, String interpMethod, String allowExtrapolation)

int **d4** [INPUT, MANDATORY, default=no default value]

d4 input integer

int **d5** [INPUT, MANDATORY, default=no default value]

d5 input integer

String **interpMethod** [INPUT, OPTIONAL, default=Regrid.LINEAR]

allowable interpolation schemes are: CUBLIC_SPLINE, NEAREST_NEIGHBOR, and LINEAR (which is the default).

String **allowExtrapolation** [INPUT, OPTIONAL, default=Regrid.EXTRAPOLATION]

default is no extrapolation allowed.

See also

- Rebin
- Developers Manual: herschel.ia.numeric.toolbox.interp.Regrid

History

- 2006-10-17 - JX: Deprecated EXTRAPOLATION
- 2006-10-17 - JX: Set extrapolation as a default behavior
- 2006-10-17 - JX: Introduce MINUS_ONE to reflect the fact the IDL conter-parternership.
- 2006-10-17 - JX: When MINUS_ONE is true, I enforced the new coordinate at newDim -1 is the same as oldDim-1. Mathematically, RegridFact = (oldDim - 1) / (newDim - 1); new_coordinate(i) = i * RegridFact when i = newDim -1, new_coordinate(newDim-1) = oldDim -1. But due to the numerically error, there may be some small detla value introduced. Therefore, I need to enforce it to oldDim -1
- 2006-10-17 - JX: Replaced several false to _allowExtrapolation in calling, for example, RegridFor1dArray(d1, _interp, _allowExtrapolation);
- 2006-10-17 - JX: Change the RegridConsts class to interface
- 2009-10-10 - LZ: Modified by Lijun HCSS-8078 Since I did not find the algorithm used, I checked IDL and found Congrid funtion in IDL. I believe that Regrid used the same algorithm as Congrid. Therefore I made the following changes:

1.341. regrid

Full Name:	herschel.ia.toolbox.image.RegridTask
Alias:	regrid
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import RegridTask
Category	Images/Analysis

Description

This is a task to regrid an image.

This is a task to regrid one image ("source") onto the spatial grid of another image ("target") or onto a given spatial grid ("wcs"). Flux conservation is ensured. Do not use both target and wcs parameters at the same time. Use one or the other.

Examples

Example 1: Regrid an image (myImage1) onto the spatial grid of another image (myImage2) :

```
regridded = regrid(source = myImage1, target = myImage2)
```

Example 2: This is an example of how you can regrid an image onto a given spatial grid :

```
regridded = regrid(source = myImage, wcs = myWcs)
```

API Summary

Properties
Image **source** [INPUT, MANDATORY, default=None]
Image **target** [INPUT, OPTIONAL, default=None]
Wcs **wcs** [INPUT, OPTIONAL, default=None]

API details

Properties

Image source [INPUT, MANDATORY, default=None]

This is the image to regrid.

Image target [INPUT, OPTIONAL, default=None]

This is the image of which the spatial grid must be used for the regridding.

Wcs wcs [INPUT, OPTIONAL, default=None]

This is the wcs that must be used for the regridding.

See also

- Developers Manual: herschel.ia.toolbox.image.RegridTask

1.342. removeBaselineFromCube

Full Name:	herschel.ia.toolbox.cube.RemoveBaselineFromCubeTask
Alias:	removeBaselineFromCube
Type:	Java Task -
Import:	from herschel.ia.toolbox.cube import RemoveBaselineFromCubeTask
Category	Data cubes/Analysis

Description

Simple baseline subtraction task.

This task fits a polynomial to the baseline/continuum of a spectrum in a given wavelength range. Secondly, the cube is subtracted/divided by this fitted continuum. You can specify this with the "divide" option. The meta data of the result cube will contain a parameter removeBaseMethod which may be either 'subtraction' or 'division'.

API Summary

Properties
SpectralSimpleCube **cube** [INPUT, MANDATORY, default=no default value]
Double1d **startOfRanges** [INPUT, MANDATORY, default=no default value]
Double1d **endOfRanges** [INPUT, MANDATORY, default=no default value]
Integer **polyDegree** [INPUT, OPTIONAL, default=2]
Boolean **divide** [INPUT, OPTIONAL, default=False]
Boolean **saveFitsInfo** [INPUT, OPTIONAL, default=False]
SpectralSimpleCube **subCube** [OUTPUT, MANDATORY, default=no default value]
SpectralSimpleCube **baseCube** [OUTPUT, MANDATORY, default=no default value]
PRODUCT **fitInfo** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

SpectralSimpleCube cube [INPUT, MANDATORY, default=no default value]

The input spectral cube.

Double1d startOfRanges [INPUT, MANDATORY, default=no default value]

The start wavelengths/frequencies/wavenumbers of the continuum ranges.

Double1d endOfRanges [INPUT, MANDATORY, default=no default value]

The end wavelengths/frequencies/wavenumbers of the continuum ranges.

Integer polyDegree [INPUT, OPTIONAL, default=2]

The degree of the polynomial fits (default = 2).

Boolean divide [INPUT, OPTIONAL, default=False]

If True then the cube will be divided by the fitted continuum, otherwise the continuum will be subtracted.

Boolean saveFitsInfo [INPUT, OPTIONAL, default=False]

If true a product with the information about the fits will be saved as well (parameters, chi^2, parameter standard deviation).

SpectralSimpleCube subCube [OUTPUT, MANDATORY, default=no default value]

The baseline/continuum subtracted/divided cube.

SpectralSimpleCube baseCube [OUTPUT, MANDATORY, default=no default value]

A cube with the fitted baselines.

PRODUCT fitInfo [OUTPUT, MANDATORY, default=no default value]

An optional product with information about the fits (parameters, chi^2, parameter standard deviation).

See also

- Developers Manual: herschel.ia.toolbox.cube.RemoveBaselineFromCubeTask

1.343. REPEAT

Full Name:	herschel.ia.numeric.toolbox.array.Repeat
Alias:	REPEAT
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.array import Repeat
Category	Arrays and datasets/Manipulation

Description

Creates a new array which contains the specified repetitions of the original array.

Example

Example 1: Apply REPEAT on a Int1d

```
x=Int1d([0,1,2,3,4,5,6,7,8,9,10,11])
print REPEAT(x,3)     # Int2d = [[0,1,2,3,4,5,6,7,8,9,10,11],
[0,1,2,3,4,5,6,7,8,9,10,11],[0,1,2,3,4,5,6,7,8,9,10,11]]
```

API Summary

Jython Syntax

```
<y>=REPEAT(<x>,<repetitions=1>)
```

Properties

any array of any rank **x** [INPUT, MANDATORY, default=no default value]

integer **repetitions** [INPUT, MANDATORY, default=no default value]

array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

any array of any rank x [INPUT, MANDATORY, default=no default value]

The input array must be an array of rank 1 to 4 (both included)

integer repetitions [INPUT, MANDATORY, default=no default value]

The repetitions to apply to this array. It must be greater than 0.

array y [OUTPUT, MANDATORY, default=no default value]

Returns an increased dimensional array with the specified repetitions.

See also

- Developers Manual: herschel.ia.numeric.toolbox.array.Repeat

1.344. replace

Full Name:	herschel.ia.toolbox.spectrum.ReplaceFreqRangesTask
Alias:	replace
Type:	Java Task -
Import:	from herschel.ia.toolbox.spectrum import ReplaceFreqRangesTask
Category	Spectra/Analysis

Description

Task for inserting and/or replacing the ranges of given spectra by the ranges spanned by other spectra.

Assume you would like to replace a range `r=(3200.0,3500.0)` (here in MHz) in the spectra `ds1` by an associated range in the spectra `ds2`, then you can achieve that as follows:

```
ds3 = extract(ds=ds2, ranges=(3200.0,3500.0))

result = replace(ds=ds1, by=ds3)
```

Various different modes are available for how to do this replacement and it can be configured using the `mode`-parameter:

- Resample the slices to be inserted to the wavescale grid of the original container (ds) and replace the original ranges by the resampled slices (`mode="replace"`). The default resampling scheme is adopted here - see the `resample`-task for further details.
- Resample the slices to be inserted to the wavescale grid of the original container (`ds`) and replace the original ranges by the average of the original ranges and the resampled slices ((`mode="avg"`). Note that this combines the information contained in the two spectra and generally will reduce noise.
- Insert brute-force the slices by replacing the ranges of the spectra `ds` by the associated wavescale range(s) in the spectra `by` - irrespective of the wavescale grid (`mode="insert"`).

The task works with whatever unit the wavescale is expressed in - as long as both input datasets are expressed in the same unit.

Note that the task assumes that the original spectrum data and the data that should replace the original should have the same dimensions, i.e. for a spectrum dataset the same number of rows with the same number of segments and for cubes the same dimensions of the cube images. In particular, this means that in general you cannot use this task to replace a single point spectrum in a spectrum container.

Example

Example 1: some examples using replace:

```
global spectra # the original spectra, defined elsewhere
global slices  # the spectra to insert into the original, defined elsewhere
# slices: the spectra to insert into the original
replaced = replace(ds=spectra, by=slices, mode="replace")
replaced = replace(ds=spectra, by=slices, mode="avg")
replaced = replace(ds=spectra, by=slices, mode="insert")
```

API Summary

Properties
SpectrumContainer **ds** [INPUT, OPTIONAL, default=No default value.]

Properties
SpectrumContainer **by** [INPUT, OPTIONAL, default=No default value.]
SpectrumContainer **result** [OUTPUT, OPTIONAL, default=No default value.]
String **mode** [INPUT, OPTIONAL, default=replace.]

API details

Properties

SpectrumContainer ds [INPUT, OPTIONAL, default=No default value.]

The container in which the spectra of the other input container should be inserted. Examples of `SpectrumContainer` are `Spectrum1d, Spectrum2d, SpectralSimpleCube`.

SpectrumContainer by [INPUT, OPTIONAL, default=No default value.]

The container with the slices that should be inserted into the first input container.

SpectrumContainer result [OUTPUT, OPTIONAL, default=No default value.]

The output container.

String mode [INPUT, OPTIONAL, default=replace.]

A mode specifying how the module should actually do the replacement. Three options are available:

- "replace": Resample the slices to be inserted to the wavescale grid of the original container (ds) and replace the original ranges by the resampled slices. The default resampling scheme is adopted here - see the `resample`-task for further details.
- "avg": Resample the slices to be inserted to the wavescale grid of the original container (`ds`) and replace the original ranges by the average of the original ranges and the resampled slices. Note that this combines the information contained in the two spectra and generally will reduce noise.
- "insert": Insert brute-force the slices by replacing the ranges of the spectra `ds` by the associated wavescale range(s) in the spectra `by` - irrespective of the wavescale grid.

See also

- SpectrumContainer
- replace
- Developers Manual: herschel.ia.toolbox.spectrum.ReplaceFreqRangesTask

1.345. resample

Full Name:	herschel.ia.toolbox.spectrum.ResampleFrequencyTask
Alias:	resample
Type:	Java Task -
Import:	from herschel.ia.toolbox.spectrum import ResampleFrequencyTask
Category	Spectra/Analysis

Description

Task for resampling the flux values of spectra with respect to a modified wavescale grid.

Different schemes are available - all conserve total flux. The schemes are typically a filter method in combination with an interpolation scheme and an integration scheme to assure flux conservation.

The standard (default) scheme uses a box filter in combination with trapezoidal integration and linear interpolation. An alternative scheme consists of using a box filter in combination with euler integration and nearest neighbor interpolation. A third scheme is based on a Gaussian filter.

Flags (mask) and weights (error) are processed if available (note that for PACS and SPIRE 'flags' and 'weights are usually referred to as 'inverse square error' and 'mask', respectively):

- For the weights, the same integration/interpolation scheme is applied as is used for the flux. For the weights, we always assume that the weights are defined 'per channel' ('wavescale bin'). (For the flux, we assume, by default, that it is given on a 'per wavescale unit' basis - if not otherwise stated by the 'isDensity' quantity.

- The flag for an output channel ('wavescale bin') is defined by combining the flags of all the input channels that are considered (for the given output channel) by an bitwise OR logic. This allows to recover all the distinct flag values that are involved in the computation of the given channel.

Hence, flags and weights are processed - but are not incorporated in the formulas to define the resampled flux.

For output bins for which there are no contributing input bins we set as flux and weights values 'NaN' and put the flag equal to 'NotSampled' (64).

The task works for any wavescale unit set in the spectra - as long as the unit the resolution is specified with is consistent with the underlying wavescale unit of the spectra. Note that the task will fail if the underlying wavescale is not monotonous (in-/decreasing). For PacsCube this task will in general not be applicable for that reason.

Other attributes found in the spectra are copied to the result container without any change.

The meta data parameter 'resolution_resampled' is adjusted by the new resampling width (w) according to the formula `SQRT( r0*r0+w*w )` where r0 is the resolution_resampled before applying the resampling operation. In case there was no 'resolution_resampled' but just a 'resolution' parameter a new parameter 'resolution_resampled' parameter is added as a copy of the 'resolution'. In case the spectra and the resolution (resampling width) is expressed at the velocity scale (km/s) the meta data parameter 'resolution_resampled' is expressed at the frequency though. To achieve that the resolution is transformed to a frequency scale by making use of the reference frequency (looked up from the meta data with key 'referenceFrequency'). If some information is missing (e.g. the reference frequency or its unit) the parameter 'resolution_resampled' is set to NaN.

Example

Example 1: some examples using resample:

```
# the following examples all produce the same result assuming that the
 wavescale is expressed in "MHz":
resampled = resample(ds=spectra, density=True, resolution=1.0)
resampled = resample(ds=spectra, density=True, resolution=1.0, unit="MHz")
resampled = resample(ds=spectra, density=True, resolution=0.001, unit="GHz")
resampled = resample(ds=spectra, density=True, scheme="euler", resolution=1.0)
# use a different scheme, the trapezoidal scheme:
resampled = resample(ds=spectra, density=True, scheme="trapezoidal",
 resolution=1.0)
# resample to a custom grid that may even be non-linear
# - note that in the example the spectra are assumed to have four segments:
grid = [4000.0 + 2.0*Double1d.range(500), 5000.0 + 5.0*Double1d.range(200),
        6000.0 + 1.0*Double1d.range(1000), 7000.0 + 2.0*Double1d.range(500)]
resampled = resample(ds=spectra, density=True, grid=grid)
# or when specifying the grid in GHz while the spectra are expressed at the MHz
 scale:
for i in range(len(grid)):
   grid[i] = grid[i] * 0.001
   pass
resampled = resample(ds=spectra, density=True, grid=grid, unit="GHz")
# resample to a grid that is delivered by a reference spectrum
resampled = resample(ds=spectra, density=True, grid=refSpectrum)
```

API Summary

Properties
SpectrumContainer **ds** [INPUT, MANDATORY, default=no default value]
SpectrumContainer **result** [OUTPUT, MANDATORY, default=no default value]
String **scheme** [INPUT, OPTIONAL, default=no default value]
Double1d[]|Double1d|PoinSpectrum|SpectrumContainer **grid** [INPUT, OPTIONAL, default=no default value.]
double **resolution** [INPUT, OPTIONAL, default=no default value.]
Object **params** [INPUT, OPTIONAL, default=no default value.]
String **unit** [INPUT, OPTIONAL, default=no default value.]
Boolean **density** [INPUT, OPTIONAL, default=no default value.]

API details

Properties

SpectrumContainer ds [INPUT, MANDATORY, default=no default value]

The input container to be resampled. Examples of `SpectrumContainer` are `Spectrum1d`, `Spectrum2d`, `SpectralSimpleCube`.

SpectrumContainer result [OUTPUT, MANDATORY, default=no default value]

The output container with the re-sampled spectra.

String scheme [INPUT, OPTIONAL, default=no default value]

The scheme to be adopted for the resampling - available are "standard" (or equivalently "trapezoidal") and "simple" (or equivalently "euler").

String scheme [INPUT, OPTIONAL, default=no default value]

- "trapezoidal" (or "standard"): combines a linear interpolation with a trapezoidal integration scheme.
- "euler" (or "simple"): combines a nearest neighbor interpolation and an Euler integration scheme.
- "gaussian": uses a Gaussian filter. Its parameter is related to the standard deviation of a Gaussian according to `param=SQRT(2*PI)*sigma`. See `herschel.ia.toolbox.spectrum.operations.segments.resampling.GaussianResampler` for further details.
- "sinc": resamples using a cardinal sine or sinc function. See `herschel.spire.ia.pipeline.spec.regrid.SincResampler for further details.`

All these schemes are defined to conserve flux. See javadoc for the classes `TrapezoidalLinIntpol, EulerNNResampler, GaussianResampler` in `herschel.ia.toolbox.spectrum.operations.segments.resampling` for further details on how these schemes work.

Double1d[]|Double1d|PoinSpectrum|SpectrumContainer grid [INPUT, OPTIONAL, default=no default value.]

The new frequency grid the spectra should be resampled to. It is specified as an array of `Double1d`'s, one for each sub-segment. In case there is only one single sub-segment, a `Double1d` can be passed directly. Alternatively, a `SpectrumContainer` with a single `PointSpectrum` or a `PointSpectrum` can be passed. In case a `SpectrumContainer` ('reference') with more than a single `PointSpectrum` is passed, the task resamples the input spectra to the wavescales of the reference, on a point spectrum by point spectrum basis. The grid (or these reference spectra) should be specified in consistent as the original spectra. The grid may be non-equidistant.

double resolution [INPUT, OPTIONAL, default=no default value.]

In case no output frequency grid is specified, the task creates an output grid with this given resolution (2 * width) by determining, for each sub-segment, the smallest minimum frequency and the largest maximum frequency. The resolution is expressed in the unit specified as `unit`-parameter. If no `unit`-parameter is specified the unit of the original frequency grid is assumed. In case the width is specified as a negative number the resulting grid will be in decreasing order.

Object params [INPUT, OPTIONAL, default=no default value.]

Custom parameter(s) needed by the specific resampler set (by setting a specific 'scheme'). An example is the Gaussian resampler that needs an additional smoothing width parameter.

String unit [INPUT, OPTIONAL, default=no default value.]

The unit the resolution or the grid is expressed in.

Boolean density [INPUT, OPTIONAL, default=no default value.]

If set to true the flux data is treated as a flux density (per wavescale unit) - otherwise the flux is treated as a per wavescale bin quantity (i.e. the integrated flux per bin). When setting the `density=True` and resampling to a grid with double wavescale width the flux values roughly remain at the same level. In contrast, when setting `density=False` and resampling to a grid with double wavescale width the flux values are doubled. The task does not infer from the unit the flux quantity is expressed in whether the parameter `density` is set to true or false.

See also

- SpectrumContainer

- Developers Manual: herschel.ia.toolbox.spectrum.ResampleFrequencyTask

History

- `2011-08-08 - melchior`: renamed from ResampleFrequency

1.346. RESHAPE

Full Name:	herschel.ia.numeric.toolbox.basic.Reshape
Alias:	RESHAPE
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Reshape
Category	Arrays and datasets/Manipulation

Description

Creates a new array of the same type as the input array but with a different shape.

The input array can be an array of any rank, and the output array has a rank defined by the specified shape argument.

The shape argument must be an array of integers such as [3,4], and the resulting array must have the same number of elements as the input array. If the shape argument is not specified, the result will always be a 1d-array.

Example

Example 1: Apply RESHAPE to a Int1d

```
x=Int1d.range(12)
print RESHAPE(x,[2,6])   # [[0,1,2,3,4,5],[6,7,8,9,10,11]]
print RESHAPE(x,[6,2])   # [[0,1],[2,3],[4,5],[6,7],[8,9],[10,11]]
print RESHAPE(x,[2,2,3]) # [[[0,1,2],[3,4,5]],[[6,7,8],[9,10,11]]]
print RESHAPE(TRANSPOSE(RESHAPE(x,[4,3])))
# [0,3,6,9,1,4,7,10,2,5,8,11]
```

API Summary

Jython Syntax

```
&lt;y&gt;=RESHAPE(&lt;x&gt;, [&lt;shape&gt;])
```

Properties

- `Array x [INPUT, MANDATORY, default=no default value]`
- `Integer array shape [INPUT, OPTIONAL, default=no default value]`
- `Array y [OUTPUT, MANDATORY, default=no default value]`

API details

Properties

`Array x [INPUT, MANDATORY, default=no default value]`

The array to be reshaped.

`Integer array shape [INPUT, OPTIONAL, default=no default value]`

The dimensions of the reshaped array. It must be an array of integers such as [3,4]. If this parameter is not specified, the output is a one-dimensional array.

Array y [OUTPUT, MANDATORY, default=no default value]

The reshaped array. This array has the same elements as the input array, but arranged according to the dimensions specified by the *shape* parameter. If the *shape* parameter is not specified, the output is a one-dimensional array.

See also

- CONCATENATE
- Developers Manual: herschel.ia.numeric.toolbox.basic.Reshape

1.347. Restarter

Full Name:	herschel.ia.toolbox.fit.Restarter
Type:	Java Class -
Import:	from herschel.ia.toolbox.fit import Restarter

Description

Restarter is Product which can store the information of a SampleList.

The information is a SampleList is stored as a TableDataset inside this Product.

Restoration of this Product into a SampleList is complicated by the need to recreate the AbstractModels inside the list. It is limited to those cases where the AbstractModels are all the same. An instance of them needs to be provided as parameter to the restoration method.

See also

- Developers Manual: herschel.ia.toolbox.fit.Restarter

1.348. restore

Full Name:	herschel.ia.toolbox.util.RestoreTask
Alias:	restore
Type:	Java Task -
Import:	from herschel.ia.toolbox.util import RestoreTask
Category	Session utilities

Description

Restore variables from a file to a jython session.

The restore task is used to restore one or more variables from a file to a jython session. The restore task is used to save one or more variable from a previously saved file into the specified namespace.

Please note that when used for restoring values using the local() option the result might be unpredictable if not executed as first command. Please note that only Serializable variable can be saved.

Examples

Example 1: Restore variables (names and values) from a file into the global namespace

```
restore("test.ser")
```

Example 2: Update locals() with variables (names and values) from a file

```
def my_function():
 print locals()
 restore("test.ser", space=locals())
 print locals()
my_function()
```

Example 3: Update locals() does not overwrite local variables

```
def my_function():
 x=100
 save("test.ser", space=locals())
 x = 1
 mySpace = locals()
 print 'Space before restore', mySpace, 'x is ',  x
 restore("test.ser", mySpace)
 print 'Space after restore', mySpace, 'x is still',  x
my_function()
```

API Summary

Jython Syntax

```
space = restore(&lt;filename&gt;[, &lt;space&gt;])
```

Properties

`String filename [INPUT, MANDATORY, default=No default value]`

`PyStringMap space [INOUT, OPTIONAL, default=globals()]`

Limitations

- Only Serializable variable saved can be restored,
- Jython does not allow overwriting the local namespace (locals() return a copy). After restoring into the copy you need to access the returned namespace explicitly with something like**locals()["x"]**. So, to save-restore **locals** your code should be designed to work with the locals() dictionary.
- Note that **locals()**, when you call it outside of a function, returns the global namespace.

API details

Properties

String filename [INPUT, MANDATORY, default=No default value]

The name of the file where the variables (with their values) are stored

PyStringMap space [INOUT, OPTIONAL, default=globals()]

A jython dictionary (PyStringMap) with the namespace to load the variables into, The choices available are:

1. "locals()" for reading variables into a copy of the local namespace (i.e. from where the function is used)
2. "globals()" for reading variables into the module where the function is used

When no space is specified the top level namespace is updated.

See also

- save
- Developers Manual: herschel.ia.toolbox.util.RestoreTask

History

- 2004-07-13 - NdC: first release.

1.349. REVERSE

Full Name:	herschel.ia.numeric.toolbox.basic.Reverse
Alias:	REVERSE
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Reverse
Category	Arrays and datasets/Manipulation

Description

Reverses the order of the elements of a one-dimensional array.

This function does not work on native Jython arrays. Use Numeric arrays instead, such as Int1d.

Example

Example 1: Apply REVERSE to a Double1d

```
print REVERSE(Double1d.range(3)) # [2.0,1.0,0.0]
```

API Summary

Jython Syntax
<y>=REVERSE(<x>)

Properties
1-D array **x** [INPUT, MANDATORY, default=no default value]
1-D array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

1-D array x [INPUT, MANDATORY, default=no default value]

The array to be reversed.

1-D array y [OUTPUT, MANDATORY, default=no default value]

The reversed array.

See also

- Developers Manual: herschel.ia.numeric.toolbox.basic.Reverse

1.350. RgbSimpleImage

Full Name:	herschel.ia.dataset.image.RgbSimpleImage
Type:	Java Class -
Import:	from herschel.ia.dataset.image import RgbSimpleImage
Category	Images

Description

A Product describing three colour images.

Example

Example 1: Creating a RgbSimpleImage

```
myByte2dRedImage = Byte2d(10, 10)
myByte2dGreenImage = Byte2d(10, 10)
myByte2dBlueImage = Byte2d(10, 10)
s = RgbSimpleImage()
s.setRedImage(myByte2dRedImage)
s.setGreenImage(myByte2dGreenImage)
s.setBlueImage(myByte2dBlueImage)
del(myByte2dRedImage, myByte2dGreenImage, myByte2dBlueImage)
```

API Summary

Constructors
RgbSimpleImage The standard constructor.
RgbSimpleImage (String description) Constructor with a description.
RgbSimpleImage (AbstractOrdered2dData red, AbstractOrdered2dData green, AbstractOrdered2dData blue) The standard constructor.
RgbSimpleImage (Byte2d red, Byte2d green, Byte2d blue) The standard constructor.
RgbSimpleImage (RgbSimpleImage copy) Copy constructor.

Methods
copy Returns a copy.
setRedImage (Ordered2dData red) Sets the red part of the image.
setGreenImage (Ordered2dData green) Sets the green part of the image.
setBlueImage (Ordered2dData blue) Sets the blue part of the image.

Methods
RgbImage **getPreview** (float res) Returns a preview of the image Returns a new RgbSimpleImage.
int **getHeight** Returns the height of this RgbSimpleImage.
AbstractOrdered2dData **getRedImage** Returns the red image as a Ordered2dData.
AbstractOrdered2dData **getGreenImage** Returns the green image as an Ordered2dData.
AbstractOrdered2dData **getBlueImage** Returns the blue image as an Ordered2d.
Unit **getUnit** Returns the unit of the image.
int **getWidth** Returns the width of this SimpleImage.
setUnit (Unit<?> unit) Sets the unit of the image.
int[] **getDimensions** Returns the dimension (width, height) of this RgbSimpleImage.
Wcs **getWcs** Returns the World Coordinates System of the image.
setWcs (Wcs wcs) Sets the world coordinates system of the image.

API Details

Constructors

RgbSimpleImage
The standard constructor. A constructor which creates a standard RgbSimpleImage.

RgbSimpleImage (String description)
Constructor with a description. Creates an RgbSimpleImage with a given description. **Argument** String **description** [INPUT, MANDATORY, default=no default value] The description of the image, as String

RgbSimpleImage (AbstractOrdered2dData red, AbstractOrdered2dData green, AbstractOrdered2dData blue)
The standard constructor. A constructor which creates a standard RgbSimpleImage. The standard SimpleImage consists of 3 Numeric2ds for the image.

RgbSimpleImage (AbstractOrdered2dData red, AbstractOrdered2dData green, AbstractOrdered2dData blue)

Arguments

`AbstractOrdered2dData` **`red`** `[INPUT, MANDATORY, default=no default value]`

The red Ordered2dData, as AbstractOrdered2dData

`AbstractOrdered2dData` **`green`** `[INPUT, MANDATORY, default=no default value]`

The green Ordered2dData, as AbstractOrdered2dData

`AbstractOrdered2dData` **`blue`** `[INPUT, MANDATORY, default=no default value]`

The blue Ordered2dData, as AbstractOrdered2dData

RgbSimpleImage (Byte2d red, Byte2d green, Byte2d blue)

The standard constructor.

A constructor which creates a standard RgbSimpleImage. The standard SimpleImage consists of 3 Byte2ds for the image

Arguments

`Byte2d` **`red`** `[INPUT, MANDATORY, default=no default value]`

The red Byte2d, as Byte2d

`Byte2d` **`green`** `[INPUT, MANDATORY, default=no default value]`

The green Byte2d, as Byte2d

`Byte2d` **`blue`** `[INPUT, MANDATORY, default=no default value]`

The blue Byte2d, as Byte2d

RgbSimpleImage (RgbSimpleImage copy)

Copy constructor.

Constructor which makes a copy from an existent RgbSimpleImage.

Argument

`RgbSimpleImage` **`copy`** `[INPUT, MANDATORY, default=no default value]`

The RgbSimpleImage to copy, as RgbSimpleImage

Methods

copy

Returns a copy.

Returns a copy from this RgbSimpleImage.

setRedImage (Ordered2dData red)

Sets the red part of the image.

Argument

`Ordered2dData` **`red`** `[INPUT, MANDATORY, default=no default value]`

An Ordered2dData describing the red part of image, as Ordered2dData

setGreenImage (Ordered2dData green)

Sets the green part of the image.

setGreenImage (Ordered2dData green)

Argument

`Ordered2dData` **`green`** `[INPUT, MANDATORY, default=no default value]`

A Numeric2d describing the green part of image, as Ordered2dData

setBlueImage (Ordered2dData blue)

Sets the blue part of the image.

Argument

`Ordered2dData` **`blue`** `[INPUT, MANDATORY, default=no default value]`

A Numeric2d describing the blue part of image, as Ordered2dData

RgbImage **getPreview (float res)**

Returns a preview of the image Returns a new RgbSimpleImage.

Returns an RgbSimpleImage with a lower resolution if res is smaller than 1. A higher resolution is returned, if res is larger than 1. The spatial regridding is based on linear interpolation which will, in general, not be suitable for scientific purposes.

Argument

`float` **`res`** `[INPUT, MANDATORY, default=no default value]`

The factor to multiply the resolution, as float

Return

RgbImage

The preview of the image

int **getHeight**

Returns the height of this RgbSimpleImage.

Return

int

The height of this RgbSimpleImage.

AbstractOrdered2dData **getRedImage**

Returns the red image as a Ordered2dData.

Returns the image as an Ordered2dData containing the data of the red image.

Return

AbstractOrdered2dData

The red image as a numeric2d

AbstractOrdered2dData **getGreenImage**

Returns the green image as an Ordered2dData.

Returns the image as an Ordered2dData containing the data of the green image.

Return

AbstractOrdered2dData

The green image as an Ordered2dData

***AbstractOrdered2dData* getBlueImage**

Returns the blue image as an Ordered2d.

Returns the image as an Ordered2dData containing the data of the blue image.

Return

AbstractOrdered2dData

The blue image as an Ordered2dData

***Unit* getUnit**

Returns the unit of the image.

The unit of the errors of this image is the same as the unit of the image.

Return

Unit

The unit

***int* getWidth**

Returns the width of this SimpleImage.

Return

int

The width of this SimpleImage.

setUnit (Unit<?> unit)

Sets the unit of the image.

Argument

`Unit&lt;?&gt;` **`unit`** `[INPUT, MANDATORY, default=no default value]`

The unit of the image

***int[]* getDimensions**

Returns the dimension (width, height) of this RgbSimpleImage.

Return

int[]

The dimension (width, height) of this RgbSimpleImage.

***Wcs* getWcs**

Returns the World Coordinates System of the image.

Return

Wcs

The World Coordinates System of the image.

setWcs (Wcs wcs)

Sets the world coordinates system of the image.

Argument

`Wcs` **`wcs`** `[INPUT, MANDATORY, default=no default value]`

setWcs (Wcs wcs)
The Wcs object.

See also

- Developers Manual: herschel.ia.dataset.image.RgbSimpleImage

1.351. RMS

Full Name:	herschel.ia.numeric.toolbox.basic.Rms
Alias:	RMS
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Rms
Category	Mathematics/Statistics

Description

Returns the root mean square of a set of values.

The root mean square is computed as follows: *RMS = SQRT ((1/N)*SUM((x-x_{avg})**2))*, where *N* is the number of elements in the array, *x* is an array element and x_{avg} is the average of the elements. In case of complex arrays, the QRMS is computed on the absolute values of the elements.

Example

Example 1: Apply RMS to a Float1d

```
x = Float1d([1,3,2,3,4])
print RMS(x) # 1.019803902718557
```

API Summary

Jython Syntax

```
&lt;y&gt;=RMS(&lt;x&gt;)
```

Properties

```
Array x [INPUT, MANDATORY, default=no default value]
Double y [OUTPUT, MANDATORY, default=no default value]
```

API details

Properties

`Array x [INPUT, MANDATORY, default=no default value]`

The input array. Integer arrays are implicitly converted to floating point arrays.

`Double y [OUTPUT, MANDATORY, default=no default value]`

The root mean square of the values in the input array.

See also

- MEAN
- MEDIAN
- STDDEV
- VARIANCE

- QRMS
- Developers Manual: herschel.ia.numeric.toolbox.basic.Rms

1.352. RobustClipWeight

Full Name:	herschel.ia.numeric.toolbox.fit.robust.RobustClipWeight
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit.robust import RobustClipWeight
Category	Mathematics/Fitting

Description

Clip weighting scheme. Also known as sigma-clipping.

For all schemes the deviant is calculated as the difference between data and fit divided by noisescale and domainsize:

d = (data - fit) / (noisescale * domainsize)

default domainsize = 4

The weighting scheme is either "in" (wgt=1) or "out" (wgt=0):

Table 1.5. Weighting scheme table.

w = 1		for |d| < 1
w = 0		elsewhere

This is a usefull scheme if you know how far the outliers are with respect to the noise scale.

It has the danger of jittering when one of the (outlying) points is moving in and out of the clip domain.

Limitations

Robust fitting is even more dangerous than ordinary fitting. Never trust what you get without thorough checking.

Miscellaneous

Beware of jittering behaviour when some points move in and out of the weighting scheme

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.robust.RobustClipWeight

1.353. RobustCosineWeight

Full Name:	herschel.ia.numeric.toolbox.fit.robust.RobustCosineWeight
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit.robust import RobustCosineWeight
Category	Mathematics/Fitting

Description

Cosine weighting scheme.

For all schemes the deviant is calculated as the difference between data and fit divided by noisescale and domainsize:

d = (data - fit) / (noisescale * domainsize)

default domainsize = 6

The weights are calculated as a truncated cosine:

Table 1.6. Weighting scheme table.

w = cos(0.5 * pi * d)		for |d| < 1
w = 0		elsewhere

Limitations

Robust fitting is even more dangerous than ordinary fitting. Never trust what you get without thorough checking.

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.robust.RobustCosineWeight

1.354. RobustMedianWeight

Full Name:	herschel.ia.numeric.toolbox.fit.robust.RobustMedianWeight
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit.robust import RobustMedianWeight
Category	Mathematics/Fitting

Description

Median weighting scheme; also known as Huber.

For all schemes the deviant is calculated as the difference between data and fit divided by noisescale and domainsize:

d = (data - fit) / (noisescale * domainsize)

default domainsize = 2

Contrary to the other schemes the datapoints never loose their influence completely. Within the domain the influence is quadratic (least squares) outside the domain it is as a median.

Table 1.7. Weighting scheme table.

w = 1		for |d| < 1
w = 1 / d		elsewhere

When the domainsize is very close to zero the result is the same as finding the minimum in the absolute residuals in stead of least squares. It is equivalent to finding the median.

Limitations

Robust fitting is even more dangerous than ordinary fitting. Never trust what you get without thorough checking.

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.robust.RobustMedianWeight

1.355. RobustShell

Full Name:	herschel.ia.numeric.toolbox.fit.robust.RobustShell
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit.robust import RobustShell
Category	Mathematics/Fitting

Description

RobustShell is a shell around another fitter to robustify the fit of the inner fitter.

Technically it is in itself a Fitter, but with limited possiblities.

A RobustShell tries to make a fit more robust in the presence of outliers. It does it by manipulating the weights: outliers are downweighted. "Normal" points keep their weights, more or less.

Apart from methods specific to the robustification, RobustShell has a fit method and little else from the Fitter family. Methods to get the χ^2, the covariance matrix, the evidence, the noise scale etc. should be taken from the embedded fitter.

The theory behind robust fitting can be found in Wikipedia: Robust Statistics, and the references therein.

To determine which points are "normal" and which are "outliers" we need a previous fit. The difference to this earlier fit determines the normalcy. Its amount is used to adjust the weights.

The fact that we need a previous iteration makes robust estimation a iterative procedure. This class performs **one** step in the iteration. Typically the procedure should stabilize in half a dozen steps maximally. It is upto the user how many steps are used.

There are several schemes to adjust the weights. But before we go into that we need two values in each scheme. Firstly the amount of noise present in the data. By default the noise is taken from the previous fit via Fitter.getScale(). The alternative is a user supplied value setNoiseScale(). The second value needed is the size of the influence domain in terms of the noise scale.

For all schemes the deviant is calculated as the difference between data and fit divided by noisescale and domainsize: d = (data - fit) / (noisescale * domainsize)

A number of weighting schemes are provided in the links below:

Example

Example 1: RobustShell

```
someModel = PolynomialModel( 1 )              # some model
someX = Double1d.range(100) / 10              # some x values
someY = Double1d( Int1d.range(100)/4 )        # some y values
someY[Selection([1,11,35,67])] += 10          # create outliers
someFtr = Fitter( someX, someModel )          # a fitter, a model and a x
rs = RobustShell( someFtr )                   # robust shell around someFtr
rs.setDomainSize( 7 )                         # set the domain
par = rs.fit( someY )                         # solution
print rs                                      #
print rs.getWeights()                         # print final weights
print someFtr.getChiSquared()                 # get from someFtr
#
```

Limitations

Robust fitting is even more dangerous than ordinary fitting. Never trust what you get without thorough checking.

See also

- RobustTukeyWeight
- RobustCosineWeight
- RobustMedianWeight
- RobustClipWeight
- Developers Manual: herschel.ia.numeric.toolbox.fit.robust.RobustShell

1.356. RobustTukeyWeight

Full Name:	herschel.ia.numeric.toolbox.fit.robust.RobustTukeyWeight
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit.robust import RobustTukeyWeight
Category	Mathematics/Fitting

Description

Tukey's weighting scheme.

For all schemes the deviant is calculated as the difference between data and fit divided by noisescale and domainsize:

d = (data - fit) / (noisescale * domainsize)

default domainsize = 6

The weights are calculated as a quartic polynome of the deviants.

Table 1.8. Weighting scheme table.

w = (1 - d^2)^2		for |d| < 1
w = 0		elsewhere

This is the default weighting scheme of the RobustShell.

Limitations

Robust fitting is even more dangerous than ordinary fitting. Never trust what you get without thorough checking.

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.robust.RobustTukeyWeight

1.357. Rotate

Full Name:	herschel.ia.numeric.toolbox.basic.Rotate
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Rotate
Category	Arrays and datasets/Manipulation

Description

Rotates a Numeric two- or three-dimensional array.

The result is a Double2d or Double3d array. The returned array is usually larger than the original, because a rotation of a rectangular array creates empty corners, unless the rotation angle is a multiple of 90 degrees. Empty corners are filled with 0 values.

The rotation is always around the centre of the array. This means that the result contains the complete rotated array without any cuts.

Possible interpolation methods are nearest neighbor, bilinear and bicubic.

The interpolation quality depends on the rotation angle and on the interpolation method. It can be measured by summing all the elements in the original and rotated arrays, and comparing the results. The following is an example using a 200x255 Int2d array:

```
array = Int2d()
for i in range(200):
    array.append(Int1d.range(255),1)
```

The sum of the input array is 6477000.

With a rotation angle of 54 degrees clockwise, the sums are the following:

Nearest neighbor: 6477268 (0.004% difference)

Bilinear: 6447345 (0.46% difference)

Bicubic: 6451030 (0.40% difference)

Example

Example 1: apply Rotate to a Int2d array

```
# Clockwise nearest neighbour rotation (default)
result = Rotate(54.0)( array )
# Counterclockwise nearest neighbour rotation
result = Rotate(-54.0)( array )
# Clockwise bicubic rotation
result = Rotate(54.0, Rotate.BICUBIC) ( array ) # or
result = Rotate(54.0, 3) ( array )
# Counterclockwise bilinear rotation
result = Rotate(54.0, Rotate.BILINEAR, False) ( array )
array2 = Int2d()
for i in range(5):
  array2.append(Int1d.range(5),1)
print array2
# [
# [0,0,0,0,0],
# [1,1,1,1,1],
# [2,2,2,2,2],
```

Example 1: apply Rotate to a Int2d array

```
# [3,3,3,3,3],
# [4,4,4,4,4]
# ]
print Rotate(90) ( array2 )
# [
# [4.0,3.0,2.0,1.0,0.0],
# [4.0,3.0,2.0,1.0,0.0],
# [4.0,3.0,2.0,1.0,0.0],
# [4.0,3.0,2.0,1.0,0.0],
# [4.0,3.0,2.0,1.0,0.0]
# ]
```

API Summary

Jython Syntax

<y> = Rotate(<angle>, <interpolation method>, <clockwise>)(<x>)

Alternative syntax:

<y> = <x>.apply(Rotate(<angle>, <interpolation method>, <clockwise>))

Properties

2-D or 3-D array **x** [INPUT, MANDATORY, default=no default value]

Double **angle** [INPUT, MANDATORY, default=no default value]

Integer **interpolation method** [INPUT, OPTIONAL, default=Rotate.NEAREST_NEIGHBOR]

Boolean **clockwise** [INPUT, OPTIONAL, default=True]

Double2d or Double3d array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

2-D or 3-D array x [INPUT, MANDATORY, default=no default value]

The array to be rotated. Complex arrays are not allowed.

Double angle [INPUT, MANDATORY, default=no default value]

The rotation angle in degrees. It can be positive or negative. A negative angle corresponds to a counterclockwise rotation.

Integer interpolation method [INPUT, OPTIONAL, default=Rotate.NEAREST_NEIGHBOR]

The interpolation method. Three options are possible. You can pass either the number or the corresponding expression such as Rotate.NEAREST_NEIGHBOR:

- Rotate.NEAREST_NEIGHBOR = 1
- Rotate.BILINEAR = 2
- Rotate.BICUBIC = 3

Boolean clockwise [INPUT, OPTIONAL, default=True]

Whether to rotate the array clockwise. True for clockwise rotation, false otherwise. Note that you can also set a counterclockwise rotation with a negative rotation angle.

Double2d or Double3d array y [OUTPUT, MANDATORY, default=no default value]

The rotated array. It has the same dimensions as the input array. Three-dimensional arrays are treated as a stack of two-dimensional arrays. Rotation is done around the depth (3rd) dimension.

See also

- Developers Manual: herschel.ia.numeric.toolbox.basic.Rotate

1.358. rotate

Full Name:	herschel.ia.toolbox.image.RotateTask
Alias:	rotate
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import RotateTask
Category	Images/Analysis

Description

This is a task to rotate an image.

This is a task to rotate an image, and adapts its Wcs. It is possible to use 4 types of interpolation :

- INTERP_NEAREST : nearest-neighbor interpolation. Nearest-neighbor interpolation is simply pixel copying
- INTERP_BILINEAR : Bilinear interpolation requires a neighborhood extending one pixel to the right and below the central sample. If the fractional subsample position is given by (xfrac, yfrac), the resampled pixel value will be:

```
(1 - yfrac) * [(1 - xfrac)*s00 + xfrac*s01]
+ yfrac * [(1 - xfrac)*s10 + xfrac*s11]
```

  A neighborhood extending one sample to the right of, and one sample below the central sample is required to perform bilinear interpolation. This implementation maintains equal subsampleBits in x and y.

- INTERP_BICUBIC : InterpolationBicubic is a subclass of Interpolation that performs interpolation using the piecewise cubic polynomial:

```
 r(x) = (a + 2)|x|^3 - (a + 3)|x|^2  +  1 , 0 <= |x| < 1
r(x) = a|x|^3 - 5a|x|^2 + 8a|x| - 4a , 1 <= |x| < 2
r(x) = 0, otherwise
```

  with 'a' set to -0.5. This definition is also sometimes known as "cubic convolution", using the parameter 'a' recommended by Rifman. (Reference: Digital Image Warping, George Wolberg, 1990, pp 129-131, IEEE Computer Society Press, ISBN 0-8186-8944-7)

  A neighborhood extending one sample to the left of and above the central sample, and two samples to the right of and below the central sample is required to perform bicubic interpolation.

- INTERP_BICUBIC_2 : InterpolationBicubic2 is a subclass of Interpolation that performs interpolation using the piecewise cubic polynomial:

```
 r(x) = (a + 2)|x|^3 - (a + 3)|x|^2  +  1 , 0 <= |x| < 1
r(x) = a|x|^3 - 5a|x|^2 + 8a|x| - 4a , 1 <= |x| < 2
r(x) = 0, otherwise
```

  with 'a' set to -1.0. This definition is also sometimes known as "cubic convolution", using the parameter 'a' recommended by Keys. This interpolator may produce somewhat sharper results than InterpolationBicubic, but that result is image dependent. (Reference: Digital Image Warping, George Wolberg, 1990, pp 129-131, IEEE Computer Society Press, ISBN 0-8186-8944-7)

  A neighborhood extending one sample to the left of and above the central sample, and two samples to the right of and below the central sample is required to perform bicubic interpolation.

When no interpolation is set, BiLinear interpolation (INTERP_BILINEAR) is taken as standard. INTERP_BICUBIC and INTERP_BICUBIC_2 need an extra parameter (subsample precision, in bits)

to be set. If the subsample precision is not set explicitely, the value will be 16. The errors are not calculated. At the moment, the same operation is executed on the errors. The Wcs is not always calculated exactly.

Examples

Example 1: This is how you can rotate an image over 12.2 degrees:

```
rotated = rotate(image = myImage, angle = 12.2)
```

Example 2: This is how you can rotate an image over 12.2 degrees, using Bicubic interpolation, using 32 bits

```
rotated = rotate(image = myImage, angle = 12.2, interpolation =
 rotate.INTERP_BICUBIC, subsampleBits = 32)
```

API Summary

Jython Syntax

```
rotate(image, 12.2, Rotate.INTERP_BICUBIC)
```

Properties

Image **image** [INPUT, MANDATORY, default=None]
float **angle** [INPUT, OPTIONAL, default=0.0]
int **interpolation** [INPUT, OPTIONAL, default=Rotate.INTERP_NEAREST]
int **subsampleBits** [INPUT, OPTIONAL, default=16.0]
Image **rotatedImage** [OUTPUT, MANDATORY, default=None]

API details

Properties

Image image [INPUT, MANDATORY, default=None]

The Image to rotate.

float angle [INPUT, OPTIONAL, default=0.0]

This is the amount of degrees to rotate the image (counterclockwise for astronomical image, clockwise for other images).

int interpolation [INPUT, OPTIONAL, default=Rotate.INTERP_NEAREST]

This is the type of interpolation to use (INTERP_NEAREST, INTERP_NEAREST, INTERP_BICUBIC, INTERP_BICUBIC_2).

int subsampleBits [INPUT, OPTIONAL, default=16.0]

This is the number of bits to use for the interpolation (only if interpolation is INTERP_BICUBIC or INTERP_BICUBIC_2).

Image rotatedImage [OUTPUT, MANDATORY, default=None]

This is the rotated image.

See also

- Developers Manual: herschel.ia.toolbox.image.RotateTask

1.359. ROUND

Full Name:	herschel.ia.numeric.toolbox.basic.Round
Alias:	ROUND
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Round
Categories	Arrays and datasets/Manipulation Mathematics/Nearest integer

Description

Returns the input number or array rounded to the nearest integer or decimal position.

If the input is an array, the output is an array of the same type and size, with the computed values instead of the original elements.

Example

Example 1: Applying ROUND to a Double1d

```
x = Double1d([-0.5001, -0.5, -0.4999, 0.4999, 0.5, 0.5001, 1.1234, 2.6236])
print ROUND(x)   # [-1.0, 0.0,  0.0,  0.0, 1.0, 1.0, 1.0,   3.0]
print ROUND(x,3) # [-0.5, -0.5, -0.5, 0.5, 0.5, 0.5, 1.123, 2.624]
```

API Summary

Jython Syntax

```
&lt;y&gt; = ROUND(&lt;x&gt; [,&lt;n&gt;])
```

Properties

Number or array **x** [INPUT, MANDATORY, default=no default value]

Integer **n** [INPUT, OPTIONAL, default=0]

Number or array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Number or array x [INPUT, MANDATORY, default=no default value]

The value or values to round. It cannot be a Complex array.

Integer n [INPUT, OPTIONAL, default=0]

The decimal position at which to round the input. By default, the input is rounded to the nearest integer. You can only use this parameter if the input is an array.

Number or array y [OUTPUT, MANDATORY, default=no default value]

The rounded value or values. A float number or array for float input, and a double number or array for any other input.

See also

- CEIL
- FLOOR
- Developers Manual: herschel.ia.numeric.toolbox.basic.Round

1.360. SampleList

Full Name:	herschel.ia.numeric.toolbox.fit.sample.SampleList
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit.sample import SampleList
Category	Mathematics/Fitting

Description

SampleList is a ArrayList of Samples generated by a Sampler.

SampleList is the main result of the NestedSampler. It contains all information to calculate averages, medians, modi or maximum likihood solutions of the parameters, or of any function of the parameters; in particular of the AbstractModel.

To make averages one has to take into account the weights. Each Sample has a weight and all weights sum to 1.0. So the average of any function f of the parameters p is

$E(f(p)) = \sum w_k f(p_k)$

where the sum is over all samples k.

A large set of utility functions is provided to extract the information from the SampleList.

See example for an script, where the SampleList is well used.

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.sample.SampleList

1.361. SampleProduct

Full Name:	herschel.ia.toolbox.fit.SampleProduct
Type:	Java Class -
Import:	from herschel.ia.toolbox.fit import SampleProduct

Description

SampleProduct is Product which can store the information of a SampleList.

The information is a SampleList is stored as a TableDataset inside this Product.

Restoration of this Product into a SampleList is complicated by the need to recreate the AbstractModels inside the list. It is limited to those cases where the AbstractModels are all the same. An instance of them needs to be provided as parameter to the restoration method.

See also

- Developers Manual: herschel.ia.toolbox.fit.SampleProduct

1.362. save

Full Name:	herschel.ia.toolbox.util.SaveTask
Alias:	save
Type:	Java Task -
Import:	from herschel.ia.toolbox.util import SaveTask
Category	Session utilities

Description

Save variables from the jython session to a file.

The save task is used to save one or more variable from a jython namespace into the specified file. Variables can be specified with a string containing a comma separated list of variable names, if no variables are specified the entire namespace is saved. Note that saving globals includes all internal configuration.

Please note that only Serializable variables will be saved.

Examples

Example 1: save all global variable names and values

```
save("foo.ser")
```

Example 2: save specific variable names and values

```
save("foo.ser","x,y,z")
```

Example 3: save specified variable names and values from the local namespace of a function

```
def my_function():
 x=100
 y=23
 save("test.ser","x,y", space=locals())
my_function()
```

API Summary

Jython Syntax

```
save(&lt;file&gt; [, &lt;variable&gt;="", &lt;space&gt;])
```

Properties

String **filename** [INPUT, MANDATORY, default=no default value]
String **variable** [INPUT, OPTIONAL, default=""]
PyStringMap **space** [INPUT, OPTIONAL, default=globals()]

Limitations

- Only serializable variables can be saved
- The system will complain unless you use ".ser" as the file extension (recognised by Navigator)

API details

Properties

String filename [INPUT, MANDATORY, default=no default value]

The name of the file to store the values of variables. Extension should be .ser

String variable [INPUT, OPTIONAL, default=""]

The comma-separated list of variables to save.

PyStringMap space [INPUT, OPTIONAL, default=globals()]

A jython dictionary (PyStringMap) containing the namespace of the variables, The choices available are:

1. locals() for saving variables from the local namespace (i.e. from where the function is used)
2. globals() for saving variables from the module where the function is used

When no space is specified data from the top level namespace is saved.

See also

- restore
- Developers Manual: herschel.ia.toolbox.util.SaveTask

History

- `2004-07-13 - NdC`: first release.

1.363. saveObservation

Full Name:	herschel.ia.toolbox.util.jython.saveobs
Alias:	saveObservation
Type:	Jython Task -
Import:	from herschel.ia.toolbox.util.jython import saveObservation
Category	Input-output

Description

Save an observation into a local pool.

The observation is saved into a pool on your local system. If only the observation is given as an argument, a local pool will be created in the default location, i.e. ~/.hcss/lstore. The default name of the pool is the OBSID for the given observation.

The calibration tree that is attached to the observation is **not** saved by default. If you want to save the calibration tree with the observation, set the argument saveCalTree=True.

Beware that saving an observation might take a long time since all the products associated to the observation will be saved locally, except the calibration tree.

Example

Example 1: Saving an observation from your HIPE session

```
saveObservation(obs)
saveObservation(obs, poolName="od144")
```

API Summary

Jython Syntax

```
saveObservation(obs[, verbose=False|True]
[, poolLocation=<directory>] [, poolName=<pool name>]
[, saveCalTree=False|True])
```

Properties

ObservationContext **obs** [INPUT, MANDATORY, default=no default]
boolean **verbose** [INPUT, OPTIONAL, default=False]
String **poolName** [INPUT, OPTIONAL, default=see description]
String **poolLocation** [INPUT, OPTIONAL, default=see description]
boolean **saveCalTree** [INPUT, OPTIONAL, default=False]

API details

Properties

ObservationContext obs [INPUT, MANDATORY, default=no default]

The observation that must be saved

boolean verbose [INPUT, OPTIONAL, default=False]

A flag for more verbose output.

String poolName [INPUT, OPTIONAL, default=see description]

The name of the pool to be used to save the requested observation

String poolLocation [INPUT, OPTIONAL, default=see description]

The directory where the pool for the given observation must be created

boolean saveCalTree [INPUT, OPTIONAL, default=False]

Indicates whether the calibration tree associated with the observation must be saved as well. By default the calTree is not saved in the local pool.

1.364. saveProduct

Full Name:	herschel.ia.obs.gui.SaveBrowseImageTool
Alias:	saveProduct
Type:	Java Class -
Import:	from herschel.ia.obs.gui import SaveBrowseImageTool
Category	Data access

Description

The product is saved to the pool you pass.

In typical usage, this will be a local pool, but it can be any kind of PAL pool. If the product that is passed is a context (e.g. an ObservationContext), it will be saved to the pool, including all its children.

Optionally, a tag can be passed as well, which will be assigned to the product.

The method returns a ProductRef to the saved product.

API Summary

Jython Syntax
`image = saveBrowseImage(product, file[, format])`

Properties
`BrowseImageProduct` **`product`** `[INPUT, MANDATORY, default=No default value]`
`String` **`file`** `[INPUT, MANDATORY, default=No default value]`
`String` **`format`** `[INPUT, OPTIONAL, default=jpg]`

API details

Properties

`BrowseImageProduct product [INPUT, MANDATORY, default=No default value]`

The browse image product whose image we want to save.

`String file [INPUT, MANDATORY, default=No default value]`

Path to the file to save.

`String format [INPUT, OPTIONAL, default=jpg]`

Image format to save: jpg, png...

See also

- Developers Manual: herschel.ia.obs.gui.SaveBrowseImageTool

1.365. saveProduct

Full Name:	herschel.ia.pal.gui.views.SaveProductsTool
Alias:	saveProduct
Type:	Java Class -
Import:	from herschel.ia.pal.gui.views import SaveProductsTool
Category	Data access

Description

The product is saved to the pool you pass.

In typical usage, this will be a local pool, but it can be any kind of PAL pool. If the product that is passed is a context (e.g. an ObservationContext), it will be saved to the pool, including all its children.

Optionally, a tag can be passed as well, which will be assigned to the product.

The method returns a ProductRef to the saved product.

API Summary

Jython Syntax
`productRef = saveProduct(product, pool [, tag])`

Properties
`Product product [INPUT, MANDATORY, default=No default value]`
`String pool [INPUT, MANDATORY, default=No default value]`
`String tag [INPUT, OPTIONAL, default=No default value]`
`ProductRef result [OUTPUT, OPTIONAL, default=no default value]`

API details

Properties

`Product product [INPUT, MANDATORY, default=No default value]`

The product variable to save.

`String pool [INPUT, MANDATORY, default=No default value]`

The (ID of the) pool to save the product to.

`String tag [INPUT, OPTIONAL, default=No default value]`

The tag or alias to assign to the product when saving it.

`ProductRef result [OUTPUT, OPTIONAL, default=no default value]`

The save operation returns a ProductRef, a light-weight reference to the saved product.

See also

- ProductRef

- Developers Manual: herschel.ia.pal.gui.views.SaveProductsTool

1.366. scale

Full Name:	herschel.ia.toolbox.image.ScaleTask
Alias:	scale
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import ScaleTask
Category	Images/Analysis

Description

This is a task to scale an image, and adapts its Wcs.

This is a task to scale an image, and adapts its Wcs. It is possible to use 4 types of interpolation :

- INTERP_NEAREST : nearest-neighbor interpolation. Nearest-neighbor interpolation is simply pixel copying
- INTERP_BILINEAR : Bilinear interpolation requires a neighborhood extending one pixel to the right and below the central sample. If the fractional subsample position is given by (xfrac, yfrac), the resampled pixel value will be:

```
(1 - yfrac) * [(1 - xfrac)*s00 + xfrac*s01] + yfrac * [(1 - xfrac)*s10 +
 xfrac*s11]
```

  A neighborhood extending one sample to the right of, and one sample below the central sample is required to perform bilinear interpolation. This implementation maintains equal subsampleBits in x and y.

- INTERP_BICUBIC : InterpolationBicubic is a subclass of Interpolation that performs interpolation using the piecewise cubic polynomial:

```
r(x) = (a + 2)|x|^3 - (a + 3)|x|^2 +  1 , 0 <= |x| < 1
r(x) = a|x|^3 - 5a|x|^2 + 8a|x| - 4a , 1 <= |x| < 2
r(x) = 0, otherwise
```

  with 'a' set to -0.5. This definition is also sometimes known as "cubic convolution", using the parameter 'a' recommended by Rifman. (Reference: Digital Image Warping, George Wolberg, 1990, pp 129-131, IEEE Computer Society Press, ISBN 0-8186-8944-7)

  A neighborhood extending one sample to the left of and above the central sample, and two samples to the right of and below the central sample is required to perform bicubic interpolation.

- INTERP_BICUBIC_2 : InterpolationBicubic2 is a subclass of Interpolation that performs interpolation using the piecewise cubic polynomial:

```
 r(x) = (a + 2)|x|^3 - (a + 3)|x|^2 +  1 , 0 <= |x| < 1
r(x) = a|x|^3 - 5a|x|^2 + 8a|x| - 4a , 1 <= |x| < 2
r(x) = 0, otherwise
```

  with 'a' set to -1.0. This definition is also sometimes known as "cubic convolution", using the parameter 'a' recommended by Keys. This interpolator may produce somewhat sharper results than InterpolationBicubic, but that result is image dependent. (Reference: Digital Image Warping, George Wolberg, 1990, pp 129-131, IEEE Computer Society Press, ISBN 0-8186-8944-7)

  A neighborhood extending one sample to the left of and above the central sample, and two samples to the right of and below the central sample is required to perform bicubic interpolation.

When no interpolation is set, BiLinear interpolation (INTERP_BILINEAR) is taken as standard. INTERP_BICUBIC and INTERP_BICUBIC_2 need an extra parameter (subsample precision, in bits) to be set. If the subsample precision is not set explicitely, the value will be 16. The errors are not calculated. At the moment, the same operation is executed on the errors. A negative scaling factoring indicates that the image must be flipped and scaled in that direction. The ScaleTask does not conserve flux!

Examples

Example 1: This is how you can scale an image by a factor 1.4 in the x-direction and -0.4 in the y-direction ;

```
scaled = scale(image = myImage, x = 1.4, y = 0.4)
```

Example 2: Scale an image by a factor 1.4 in x-direction, and 0.4 in y-direction, using Bicubic interpolation

```
scaled = scale(image = myImage, x = 1.4, y = 0.4, interpolation =
 scale.INTERP_BICUBIC, subsampleBits = 32)
```

API Summary

Jython Syntax

```
scale(image, 1.4, 0.4, ScaleTask.INTERP_BICUBIC, 32)
```

Properties

Image **image** [INPUT, MANDATORY, default=None]
float **x** [INPUT, OPTIONAL, default=1.0]
float **y** [INPUT, OPTIONAL, default=1.0]
int **interpolation** [INPUT, OPTIONAL, default=Rotate.INTERP_NEAREST]
int **subsampleBits** [INPUT, OPTIONAL, default=16.0]
Image **scaledImage** [OUTPUT, MANDATORY, default=None]

API details

Properties

Image image [INPUT, MANDATORY, default=None]

This is the input image.

float x [INPUT, OPTIONAL, default=1.0]

This is the scale factor in x-direction (the columns). A negative value indicates the image must be flipped.

float y [INPUT, OPTIONAL, default=1.0]

This is the scale factor in y-direction (the rows). A negative value indicates the image must be flipped.

int interpolation [INPUT, OPTIONAL, default=Rotate.INTERP_NEAREST]

This is the type of interpolation to use (INTERP_NEAREST, INTERP_NEAREST, INTERP_BICUBIC, INTERP_BICUBIC_2).

int subsampleBits [INPUT, OPTIONAL, default=16.0]

This is the number of bits to use for the interpolation (only if interpolation is INTERP_BICUBIC or INTERP_BICUBIC_2).

Image scaledImage [OUTPUT, MANDATORY, default=None]

This is the scaled image.

See also

- Developers Manual: herschel.ia.toolbox.image.ScaleTask

1.367. select

Full Name:	herschel.ia.toolbox.spectrum.SelectSpectrumTask
Alias:	select
Type:	Java Task -
Import:	from herschel.ia.toolbox.spectrum import SelectSpectrumTask
Category	Spectra/Analysis

Description

Task for selecting individual spectra and pack them in a new so called spectrum container.

The input data with the spectra to select from can be provided in different forms: Either any data object that implements `SpectrumContainer` (e.g. `Spectrum1d, Spectrum2d, SpectralSimpleCube`) or any array of such. `SpectrumContainer`'s provides a uniform way to access spectrum data organized in quite different structures.

For a successful processing of the task, the data included in the containers should all be consistent. This means that the number of segments found in all the spectra in the containers and their lengths should be equal; furthermore, the segment indices should be the same for all the containers (check with `ds.getSegmentIndices()`). Otherwise an exception is thrown.

Different schemes are available for selecting the point spectra or the segments. The most simple scheme is to specify lists of point spectrum indices (`selection=[1,3,4,2]`). In this case, the point spectra with corresponding indices are selected. Instead of specifying a list of indices (which requires the knowledge of these) other recipes to specify a selection are available. Typical recipes test attribute values to match certain values (e.g. `selection={"bbtype":[6031, 6032]}`) or to be included in a given interval (e.g. `selection={"LoFrequency":(999.98,1000.02)}`). These recipes we refer to as selection models. See also the URM entries on SpectrumContainer and PointSpectrum for more information. The available attributes can be checked by writing `print ds.getAttributeNames()` on the command line. Selections can be formulated only for attributes with primitive data type (i.e. double, long, int or string).

The different options to specify selections can be combined. Here, an AND logic is adopted. See the parameter descriptions and the examples below for further detail. Yet another alternative is to use `segments` where you specify which subsegments are selected from all the input spectra.

Note that when a list of input spectrum container is specified and the `segments` and `selection` parameter refer to all containers. You cannot specify specific selection and segments parameters for each input container. Apply the task for each individual container if you want to do that.

Example

Example 1: selection examples

```
global spectra # defined elsewhere
from herschel.ia.toolbox.spectrum import SelectSpectrumTask
from herschel.ia.toolbox.spectrum.selections.models import RangesSelectionModel
select = SelectSpectrumTask()
# select from all the spectra the segments, here we only ask for the first
 segment
selected = select(ds=spectra, segments=[0])
# select the spectra that have a 'bbtype' attribute set to 6031 or 6613 AND a
 'buffer' attribute set to 2
selected = select(ds=spectra, selection={"bbtype":[6031, 6613], "buffer":[2]})
# select the first four spectra
selected = select(ds=spectra, selection=[0,2])
```

Example 1: selection examples

```
# select the spectra that have a 'LoFrequency' attribute in the range
 (550.0,570.0).
selected = select(ds=spectra, selection={"LoFrequency":(550.0,570.0)})
# select the spectra that have a 'Chopper' attribute centered within a
 tolerance of 0.1 around -4.4 or 5.9.
selected = select(ds=spectra, selection={"Chopper":([-4.4,5.9],0.1)})
from herschel.ia.toolbox.spectrum.selections.models import RangesSelectionModel
selected = select(ds=spectra, selection=RangesSelectionModel("Chopper",
 [-4.4,5.9], 0.1)) # same result as above
# note: for cubes you can specify the selection by spaxel coordinates - just
 pass a list of tuples (col,row)
# where 'col' is the column and 'row' the row index:
## spectraOut = select(ds=spectra, selection=[(1,3),(0,4),(4,3)])
```

API Summary

Properties
Object **ds** [INPUT, OPTIONAL, default=no default value.]
Object **segments** [INPUT, OPTIONAL, default=no default value.]
Object **selection** [INPUT, OPTIONAL, default=None.]
PyDictionary|Map<String,Set<Object>> **lookup_selection** [INPUT, OPTIONAL, default=no default value.]
PyList **index_selection** [INPUT, OPTIONAL, default=No default value.]
PyDictionary **meta_selection** [INPUT, OPTIONAL, default=No default value.]
SpectrumContainer **result** [OUTPUT, OPTIONAL, default=no default value]

API details

Properties

Object ds [INPUT, OPTIONAL, default=no default value.]

Input data to be processed by the task. Several types are possible:

- SpectrumContainer
- Array of SpectrumContainer, i.e. SpectrumContainer[] (e.g. `[ds1,ds2,ds3]`)
- List of SpectrumContainer, i.e. List<SpectrumContainer>
- (Any product with implementations of `SpectrumContainer` inside.)

Examples of `SpectrumContainer` are `Spectrum1d, Spectrum2d, SpectralSimpleCube`.

Object segments [INPUT, OPTIONAL, default=no default value.]

Specify what segments to restrict on. There are two options available:

- Specify a PyList of segment indices or
- pass an instance of a SegmentSelection which gives the information on what segments for each point spectrum included in the container.

Object selection [INPUT, OPTIONAL, default=None.]

Specification of what point spectra select. Different ways to specify these selections are possible:

- Specify a list of indices (in jython) of the point spectra to select. For cubes, you can alternatively also pass a list of spaxel coordinates (a list of integer tuples `(row,col)`).
- Use a dictionary (in jython) to specify a selection model (lookup for attributes, filter attributes to be included in ranges or intervals).
- Pass a string representation of a jython list or a jython dictionary. The string will be parsed and the processing delegated to the two cases above.
- Pass any java instance that implements the SelectionModel interface (for the advanced user).

See the examples below for how to specify selections according to the first two bullets.

PyDictionary|Map<String,Set<Object>> lookup_selection [INPUT, OPTIONAL, default=no default value.]

Specify a PyDictionary with keys given by the attribute name to look up and a list of admissible values. Behind the scenes, a DiscreteValueSelectionModel is instantiated. This parameter is actually obsolete but is kept for historical reasons (use `selection`).

PyList index_selection [INPUT, OPTIONAL, default=No default value.]

Specify a PyList with the indices of the point spectra to be considered. This parameter is actually obsolete but is kept for historical reasons (use `selection`).

PyDictionary meta_selection [INPUT, OPTIONAL, default=No default value.]

Specify criteria to restrict the operation on spectrum containers that have meta data matching given values. This only applies in case more than one container is passed as input data to the task.

SpectrumContainer result [OUTPUT, OPTIONAL, default=no default value]

Result object containing the results of the operation applied.

See also

- Developers Manual: herschel.ia.toolbox.spectrum.SelectSpectrumTask

History

- `2011-08-08 - melchior`: renamed from SelectSpectrum
- `2011-09-28 - melchior`: add selection by spaxels - tuples (row,col).

1.368. SerialArchive

Full Name:	herschel.ia.io.serial.SerialArchive
Alias:	SerialArchive
Type:	Java Class -
Import:	from herschel.ia.io.serial import SerialArchive
Category	Input-output

Description

A class for writing and reading serialised objects.

The SerialArchive provides a transparent way to store and retrieve Products as defined within the Herschel Data Processing software.

Example

Example 1: default usage:

```
from herschel.ia.io.serial import SerialArchive
# write/read a Product in a serialised file.
s = SerialArchive()
s.save(path, Product())
p = s.load(path)
```

API Summary

Methods
Product **load** (InputStream stream) Loads a Product from an input stream.
Product **load** (String fileName) Loads a Product from a serialised file.
save (String fileName, [Optionally derived] Product product) Saves a Product to a serialised file.
createOutputStream (String fileName) Creates an object output stream for storing in a file.
createOutputStream (OutputStream file) Creates an object output stream for storing in a file.
createOutputStream (OutputStream stream) Creates an object output stream for storing in a file.
createInputStream (String fileName) Creates an object input stream for reading from a file.
createInputStream (String file) Creates an object input stream for reading from a file.
createInputStream (InputStream stream) Creates an object input stream for reading from a file.

API Details

Methods

Product load (InputStream stream)

Loads a Product from an input stream.

Loads a Product from an input stream using a serial reader.

Argument

`InputStream` **`stream`** `[INPUT, MANDATORY, default=no default value]`

Input stream from where the Product is to be read.

Return

Product

An object of the herschel.ia.dataset.Product family.

Product load (String fileName)

Loads a Product from a serialised file.

Loads a Product from a serialised file using a serial reader.

Argument

`String` **`fileName`** `[INPUT, MANDATORY, default=no default value]`

Name of the serialised file.

Return

Product

An object of the herschel.ia.dataset.Product family.

save (String fileName, [Optionally derived] Product product)

Saves a Product to a serialised file.

Saves a Product to a serialised file with sufficient information to preserve the quantities of the data as well as the original dataset type and contents.

Arguments

`String` **`fileName`** `[INPUT, MANDATORY, default=no default value]`

Name of the serialised file

`[Optionally derived] Product` **`product`** `[INPUT, MANDATORY, default=no default value]`

An object of the herschel.ia.dataset.Product family.

createOutputStream (String fileName)

Creates an object output stream for storing in a file.

Creates an object output stream for storing objects in a file, by calling its writeObject method. The returned stream treats Products and Datasets as as special case, so that intermediate classes (resolvers) can manage them appropriately, even in the presence of different package names or class versions.

Argument

`String` **`fileName`** `[INPUT, MANDATORY, default=no default value]`

Name of the file in which the data is to be serialised

createOutputStream (OutputStream file)

Creates an object output stream for storing in a file.

Creates an object output stream for storing objects in a file, by calling its writeObject method. The returned stream treats Products and Datasets as as special case, so that intermediate classes (resolvers) can manage them appropriately, even in the presence of different package names or class versions.

Argument

`OutputStream` **`file`** `[INPUT, MANDATORY, default=no default value]`

File in which the data is to be serialised

createOutputStream (OutputStream stream)

Creates an object output stream for storing in a file.

Creates an object output stream for storing objects in a file, by calling its writeObject method. The returned stream treats Products and Datasets as as special case, so that intermediate classes (resolvers) can manage them appropriately, even in the presence of different package names or class versions.

Argument

`OutputStream` **`stream`** `[INPUT, MANDATORY, default=no default value]`

Stream to be wrapped

createInputStream (String fileName)

Creates an object input stream for reading from a file.

Creates an object input stream for reading objects from a file, by calling its readObject method. The returned stream treats Products and Datasets as as special case, so that intermediate classes (resolvers) can manage them appropriately, even in the presence of different package names or class versions.

Argument

`String` **`fileName`** `[INPUT, MANDATORY, default=no default value]`

Name of the serialised file

createInputStream (String file)

Creates an object input stream for reading from a file.

Creates an object input stream for reading objects from a file, by calling its readObject method. The returned stream treats Products and Datasets as as special case, so that intermediate classes (resolvers) can manage them appropriately, even in the presence of different package names or class versions.

Argument

`String` **`file`** `[INPUT, MANDATORY, default=no default value]`

Name of the serialised file

createInputStream (InputStream stream)

Creates an object input stream for reading from a file.

Creates an object input stream for reading objects from a file, by calling its readObject method. The returned stream treats Products and Datasets as as special case, so that intermediate classes (resolvers) can manage them appropriately, even in the presence of different package names or class versions.

Argument

createInputStream (InputStream stream)

InputStream **stream** [INPUT, MANDATORY, default=no default value]

Stream to be wrapped

See also

- FitsArchive
- Developers Manual: herschel.ia.io.serial.SerialArchive

1.369. SHIFT

Full Name:	herschel.ia.numeric.toolbox.basic.Shift
Alias:	SHIFT
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Shift
Category	Arrays and datasets/Manipulation

Description

Shifts the elements of an array along the specified dimension.

The shift is circular: if an element is pushed past the last position in an array, it is reintroduced at the first position. This function does not work on native Jython arrays. Use Numeric arrays instead, such as Int1d.

Examples

Example 1: Apply SHIFT to an Int1d

```
x=Int1d([0,1,2,3,4,5,6,7,8,9,10,11])
print SHIFT(x,-3)     # [3,4,5,6,7,8,9,10,11,0,1,2]
```

Example 2: Apply SHIFT to an Int2d

```
x=Int2d([[0,1,2,3],[4,5,6,7],[8,9,10,11]])
# Dimension 0: rows are shifted.
print SHIFT(x,2)     # [[4,5,6,7],[8,9,10,11],[0,1,2,3]]
# Dimension 1: columns are shifted.
print SHIFT(x,2,1)   # [[2,3,0,1],[6,7,4,5],[10,11,8,9]]
```

API Summary

Jython Syntax

<y>=SHIFT(<x>, <shift>, <dimension>)

Properties

Array **x** [INPUT, MANDATORY, default=no default value]

Integer **shift** [INPUT, MANDATORY, default=no default value]

Integer **dimension** [INPUT, OPTIONAL, default=0]

Array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Array x [INPUT, MANDATORY, default=no default value]

The array whose elements are to be shifted.

Integer shift [INPUT, MANDATORY, default=no default value]

The number of elements to shift.

Integer dimension [INPUT, OPTIONAL, default=0]
The dimension along which to apply the shift.

Array y [OUTPUT, MANDATORY, default=no default value]
The output array with shifted elements.

See also

- Developers Manual: herschel.ia.numeric.toolbox.basic.Shift

1.370. Short1d

Full Name:	herschel.ia.numeric.Short1d
Type:	Java Class -
Import:	from herschel.ia.numeric import Short1d
Category	Arrays and datasets

Description

A rectangular numeric short array of rank 1.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Short1d

1.371. Short2d

Full Name:	herschel.ia.numeric.Short2d
Type:	Java Class -
Import:	from herschel.ia.numeric import Short2d
Category	Arrays and datasets

Description

A rectangular numeric short array of rank 2.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Short2d

1.372. Short3d

Full Name:	herschel.ia.numeric.Short3d
Type:	Java Class -
Import:	from herschel.ia.numeric import Short3d
Category	Arrays and datasets

Description

A rectangular numeric short array of rank 3.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Short3d

1.373. Short4d

Full Name:	herschel.ia.numeric.Short4d
Type:	Java Class -
Import:	from herschel.ia.numeric import Short4d
Category	Arrays and datasets

Description

A rectangular numeric short array of rank 4.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Short4d

1.374. Short5d

Full Name:	herschel.ia.numeric.Short5d
Type:	Java Class -
Import:	from herschel.ia.numeric import Short5d
Category	Arrays and datasets

Description

A rectangular numeric short array of rank 5.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.Short5d

1.375. SideBandGED

Full Name:	herschel.ia.numeric.toolbox.fit.sample.SideBandGED
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit.sample import SideBandGED
Category	Mathematics/Fitting

Description

SideBandGED is a special case (with a sideband) of the GaussErrorDistribution.

Gauss distr: $f(x) = (2 \pi s^2)^{-0.5} \exp(- 0.5 (x / s)^2)$

where s is the scale and x is the residual

For a simple way to choose the distribution in the NestedSampler use NestedSampler.setGaussDistribution().

It is the default.

See example

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.sample.SideBandGED

1.376. SideBandModel

Full Name:	herschel.ia.numeric.toolbox.fit.sample.SideBandModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit.sample import SideBandModel

Description

Free Shape Model.

f(x:p) = p (expanded)

where p is a array of amplitudes of the same size as the input x divided by the number of pixels per bin (ppb). When ppb > 1, each p is repeated ppb times to fill the complete length of x.

By default ppb = 5.

The parameters are initialized at {0}.

Although this is a LinearModel it will not work very well with the (linear) Fitter. Its exponential prior ensures that all parameters are kept positive.

Example

Example 1: creating a SideBandModel

```
from herschel.ia.numeric.toolbox.fit.sample import SideBandModel
F = DataFormatter()
nn = 100
x = Double1d.range( nn )
fsm = SideBandModel( nn )
```

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.sample.SideBandModel

1.377. SideBandRatioGED

Full Name:	herschel.ia.numeric.toolbox.fit.sample.SideBandRatioGED
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit.sample import SideBandRatioGED
Category	Mathematics/Fitting

Description

SideBandRatioGED is a special case of the GaussErrorDistribution that considers the sideband ratio.

Gauss distr: $f(x) = (2 \pi s^2)^{-0.5} \exp(- 0.5 (x / s)^2)$

where s is the scale and x is the residual

For a simple way to choose the distribution in the NestedSampler use NestedSampler.setGaussDistribution().

It is the default.

See example

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.sample.SideBandRatioGED

1.378. SideBandRatioModel

Full Name:	herschel.ia.numeric.toolbox.fit.sample.SideBandRatioModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit.sample import SideBandRatioModel

Description

Free Shape Model.

f(x:p) = p (expanded)

where p is a array of amplitudes of the same size as the input x divided by the number of pixels per bin (ppb). When ppb > 1, each p is repeated ppb times to fill the complete length of x.

By default ppb = 5.

The parameters are initialized at {0}.

Although this is a LinearModel it will not work very well with the (linear) Fitter. Its exponential prior ensures that all parameters are kept positive.

Example

Example 1: creating a SideBandRatioModel

```
#
from herschel.ia.numeric.toolbox.fit.sample import SideBandRatioModel
F = DataFormatter()
nn = 100
x = Double1d.range( nn )
fsm = SideBandRatioModel( nn )
#end
```

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.sample.SideBandRatioModel

1.379. Sigclip

Full Name:	herschel.ia.numeric.toolbox.basic.Sigclip
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Sigclip
Category	Mathematics/Statistics

Description

Finds and removes outliers in an array.

Two criteria can be used to determine outliers:

- Values that differ from the mean more than a certain number of standard deviations.
- Values that differ from the median more than a certain number of median absolute deviations.

The mean or median are computed by default in a box of given size around each data item. The data item being checked is not included in the computation. Alternatively, the mean or median can be computed over the whole array.

The output can be an array with clipped values or an array of booleans where *True* values indicate the position of outliers. For one-dimensional arrays, you can also choose to remove outliers instead of clipping them.

Example

Example 1: Apply Sigclip to an Int1d array

```
array = Int1d.range(9)
array.set(5, 20)
print array
# [0,1,2,3,4,20,6,7,8]
print array.apply(Sigclip())
# [0,1,2,3,4,5,6,7,8]
print array.apply(Sigclip(returnmode = Sigclip.RETURN_BOOL))
# [false,false,false,false,false,true,false,false,false]
print array.apply(Sigclip(env=4, nsigma=2.5, mode=Sigclip.MEDIAN,
 edge=Sigclip.TRUNCATE, \
      returnmode=Sigclip.RETURN_BOOL))
# [false,false,false,false,false,true,false,false,false]
array = Int1d(20, 9)
array.set(7, 20)
array.set(17, 0)
print array
# [9,9,9,9,9,9,9,20,9,9,9,9,9,9,9,9,9,0,9,9]
print array.apply(Sigclip(reduce1d = True, outliers = "both"))
# [9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9]
```

API Summary

Jython Syntax

```
&lt;y&gt; = &lt;x&gt;.apply(Sigclip([env = env] [, nsigma =
nsigma] [, returnmode = returnmode] [, mode = mode]))
```

Properties

1-D to 3-D array **x** [INPUT, MANDATORY, default=no default value]

Properties
Integer **env** [INPUT, OPTIONAL, default=3]
Double **nsigma** [INPUT, OPTIONAL, default=3]
Integer **returnmode** [INPUT, OPTIONAL, default=0]
Integer **mode** [INPUT, OPTIONAL, default=1]
String **outliers** [INPUT, OPTIONAL, default=both]
String **behavior** [INPUT, OPTIONAL, default=filter]
Boolean **reduce1d** [INPUT, OPTIONAL, default=False]
Integer **edges** [INPUT, OPTIONAL, default=0]
Integer **n** [INPUT, OPTIONAL, default=1]
1-D to 3-D array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

1-D to 3-D array x [INPUT, MANDATORY, default=no default value]

The array whose values are to be checked for outliers.

Integer env [INPUT, OPTIONAL, default=3]

The size of the box used to compute the mean or median around each array element.

If *i* is the coordinate in a given dimension of the element being tested, the box boundaries are *[i-env, i+env]* in that dimension.

Double nsigma [INPUT, OPTIONAL, default=3]

The minimum number of standard deviations, or median absolute deviations, for an array value to be considered an outlier.

You should adapt the value of this parameter after switching between the median and the mean mode. In median mode, this parameter can be considerably larger than in mean mode. This is because the standard deviation is influenced by an outlier, while the median absolute deviation remains unaffected.

Integer returnmode [INPUT, OPTIONAL, default=0]

If set to 0 or Sigclip.RETURN_ARRAY, the function returns a copy of the input array with the clipped values.

If set to 1 or Sigclip.RETURN_BOOL, the function returns a boolean array where the clipped locations are marked as *True*.

Integer mode [INPUT, OPTIONAL, default=1]

Determines whether the mean or median is used to define and replace outliers.

If set to 0 or Sigclip.MEDIAN, the median is used.

If set to 1 or Sigclip.MEAN, the mean is used.

String outliers [INPUT, OPTIONAL, default=both]

Determines whether only positive, only negative or both positive and negative outliers are removed.

String outliers [INPUT, OPTIONAL, default=both]

Accepted values are *positive*, *negative* and *both*. These values are strings, so you must enclose them within single or double quotes.

String behavior [INPUT, OPTIONAL, default=filter]

Determines whether Sigclip acts as a filter or as a classical clip.

If set to *filter*, Sigclip uses a box of size *env* around each data value to compute mean and median.

If set to *clip*, Sigclip uses the whole array to compute mean and median.

Boolean reduce1d [INPUT, OPTIONAL, default=False]

If *True*, outliers are removed rather than clipped for one-dimensional arrays.

Integer edges [INPUT, OPTIONAL, default=0]

Determines how the box for computing local mean and median is handled close to array edges. Four values are allowed.

If set to 0 or Sigclip.TRUNCATE, the box is truncated whenever it goes past the array boundaries.

If set to 1 or Sigclip.FILL_EDGEVALUE, empty indices of the box are filled with the last value of the array.

If set to 2 or Sigclip.FILL_MEAN, empty indices of the box are filled with the mean of the values inside the rest of the box.

If set to 3 or Sigclip.FILL_MEDIAN, empty indices of the box are filled with the median of the values inside the rest of the box.

Integer n [INPUT, OPTIONAL, default=1]

The maximum number of clipping iterations. Sigclip will stop before reaching this limit if it finds no new outliers during an iteration.

1-D to 3-D array y [OUTPUT, MANDATORY, default=no default value]

Depending on the *returnmode* parameter, either a copy of the input array with clipped outliers, or a boolean array with the same size as the input array, with *True* indication the positions of outliers.

See also

- Developers Manual: herschel.ia.numeric.toolbox.basic.Sigclip

1.380. SIGNUM

Full Name:	herschel.ia.numeric.toolbox.basic.Signum
Alias:	SIGNUM
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Signum
Category	Mathematics/General functions

Description

Returns the sign function of a number or array.

The sign function is 1 for positive input and -1 for negative input. The sign function of zero is zero.

For complex numbers, the sign function is computed as follows:

- (0+0j) if both real and imaginary part are zero.
- (1+0j) if the real part is positive, or if the real part is zero and the imaginary part is positive.
- (-1+0j) if the real part is negative, or if the real part is zero and the imaginary part is negative.

If the input is an array, the output is an array of the same type and size, with the sign function values instead of the original elements.

Example

Example 1: Applying SIGNUM to an Int1d

```
x = Int1d([-1,0,2])
print SIGNUM(x) # [-1,0,1]
```

API Summary

Jython Syntax

```
&lt;y&gt; = SIGNUM(&lt;x&gt;)
```

Properties

Number or array **x** [INPUT, MANDATORY, default=no default value]

Number or array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Number or array x [INPUT, MANDATORY, default=no default value]

The value or values of which to compute the sign function.

Number or array y [OUTPUT, MANDATORY, default=no default value]

The sign function of the value or values.

See also

- Developers Manual: herschel.ia.numeric.toolbox.basic.Signum

1.381. SimpleCube

Full Name:	herschel.ia.dataset.image.SimpleCube
Type:	Java Class -
Import:	from herschel.ia.dataset.image import SimpleCube
Category	Data cubes

Description

A Product describing cubes.

The Simplecube is a way to store and work with Cubes in the HCSS.

Example

Example 1: Creating a SimpleCube

```
s = SimpleCube()
s.setImage(myDouble3dImage)
```

API Summary

Constructors
SimpleCube The standard constructor.
SimpleCube (SimpleCube copy) The copy constructor.
SimpleCube (String description) Constructor with a description.

Methods
copy Returns a copy.
Wcs **getWcs** Returns the World Coordinates System.
setImage (AbstractOrdered3dData image, Unit unit, String description) Sets the cube.
setImage (AbstractOrdered3dData image, Unit unit) Sets the cube.
setWcs (Wcs wcs) Sets the world coordinates system.
double **getIntensity** (int depth, double row, double column) Returns the intensity.
double **getIntensityWorldCoordinates** (int depth, double x, double y) Returns the intensity of the cube.
setIntensity (int depth, int row, int column, Number value)

Methods
Sets the intensity of the image.
Number **getPixel** (int depth, int row, int column)
Returns 1 pixel value.
int **getDepth**
Returns the number of layers.
Cube **getPreview** (float res)
Returns a preview of the cube.
AbstractOrdered3dData **getError**
Returns the error of the cube as a Numeric3d.
AbstractOrdered3dData **getExposure**
Returns the exposure of the cube as a Numeric3d.
AbstractOrdered3dData **getCoverage**
Returns the coverage of the cube as an AbstractOrdered3dData.
Flag **getFlag**
Returns the flag of the cube.
int **getHeight**
Returns the height.
AbstractOrdered3dData **getImage**
Returns the cube as a Numeric3d.
Unit **getUnit**
Returns the unit.
Unit **getUnit** (String identifier)
Returns the unit.
int **getWidth**
Returns the width.
boolean **hasError**
Checks whether the cube has an error.
boolean **hasExposure**
Checks whether the cube has an exposure.
boolean **hasCoverage**
Checks whether the cube has a coverage.
boolean **hasFlag**
Checks when the cube has a flag.
setUnit (Unit<?> unit)
Sets the unit.
setUnit (String unit)
Sets the unit.
setUnit (String identifier, Unit<?> unit)
Sets the unit.
setUnit (String identifier, String unit)
Sets the unit.

Methods
removeError Removes the error.
removeExposure Removes the exposure.
removeCoverage Removes the coverage.
removeFlag Removes the flag.
setImage (AbstractOrdered3dData cube) Sets the cube.
setError (AbstractOrdered3dData error) Sets the error.
setError (AbstractOrdered3dData error, String description) Sets the error.
setFlag (Flag flag, String description) Sets the flag.
setFlag (Flag flag) Sets the flag.
setExposure (AbstractOrdered3dData exposure, String description, Unit<?> unit) Sets the exposure.
setExposure (AbstractOrdered3dData exposure, String description) Sets the exposure.
setExposure (AbstractOrdered3dData exposure) Sets the exposure.
setCoverage (AbstractOrdered3dData coverage, String description, Unit<?> unit) Sets the coverage.
setCoverage (AbstractOrdered3dData coverage, String description) Sets the coverage.
setCoverage (AbstractOrdered3dData coverage) Sets the coverage.
int[] **getDimensions** Returns the dimension.

API Details

Constructors

SimpleCube
The standard constructor. A constructor which creates a standard SimpleCube. The standard SimpleCube consists of a Numeric3d for the image. The SimpleCube has the depth as the first (most slowly varying) index.

SimpleCube

Example

Typical example on how to create an SimpleCube.

```
wcs = Wcs(3)
im = Double3d(3, 10, 10)
er = Double3d(3, 10, 10)
image=SimpleCube(description="ngc 6992", image=im, error=er, wcs=wcs)
```

SimpleCube (SimpleCube copy)

The copy constructor.

Creates a new SimpleCube with the same contents of the given SimpleCube.

Argument

`SimpleCube` **`copy`** `[INPUT, MANDATORY, default=no default value]`

The SimpleCube that should be copied, as SimpleCube

SimpleCube (String description)

Constructor with a description.

Creates a SimpleCube with a given description.

Argument

`String` **`description`** `[INPUT, MANDATORY, default=no default value]`

The descriprion of the SimpleCube, as String

Methods

copy

Returns a copy.

Returns a copy from this SimpleCube.

Wcs **getWcs**

Returns the World Coordinates System.

Returns the World Coordinates System of the image.

Return

Wcs

The World Coordinates System of the image.

setImage (AbstractOrdered3dData image, Unit unit, String description)

Sets the cube.

Sets the cube. You can give a description to the cube and give a unit. The given unit will automatically also be used for the errors on this cube. If the new cube has other dimensions than the original cube, the errors, the Exposure and the Flag are removed. If the new cube has the same dimensions, then the old values for the errors, exposure and flag are kept (but the unit of the errors is not).

Arguments

`AbstractOrdered3dData` **`image`** `[INPUT, MANDATORY, default=no default value]`

A Numeric3d describing the cube, as AbstractOrdered3dData

setImage (AbstractOrdered3dData image, Unit unit, String description)

`Unit` **`unit`** `[INPUT, MANDATORY, default=no default value]`

The unit of the cube, as Unit

`String` **`description`** `[INPUT, MANDATORY, default=no default value]`

A string describing the cube, as String

setImage (AbstractOrdered3dData image, Unit unit)

Sets the cube.

Sets the cube. You can give a unit. The given unit will automatically also be used for the errors on this cube. If the new cube has other dimensions than the original cube, the errors, exposure and the flag are removed. If the new cube has the same dimensions, then the old values for the errors, exposure and flag are kept (but the unit of the errors is not).

Arguments

`AbstractOrdered3dData` **`image`** `[INPUT, MANDATORY, default=no default value]`

A Numeric3d describing the cube, as AbstractOrdered3dData

`Unit` **`unit`** `[INPUT, MANDATORY, default=no default value]`

The unit of the cube, as Unit

setWcs (Wcs wcs)

Sets the world coordinates system.

Sets the world coordinates system of the cube.

Argument

`Wcs` **`wcs`** `[INPUT, MANDATORY, default=no default value]`

The Wcs object, as Wcs

double **getIntensity (int depth, double row, double column)**

Returns the intensity.

Returns the intensity of the SimpleCube at a given point. If the pixel is flagged out, NaN is given back.

Arguments

`int` **`depth`** `[INPUT, MANDATORY, default=no default value]`

The depth of the image in the cube, as int

`double` **`row`** `[INPUT, MANDATORY, default=no default value]`

The row of the image, as double

`double` **`column`** `[INPUT, MANDATORY, default=no default value]`

The column of the image, as double

Return

double

The intensity of the image at the given point.

double **getIntensityWorldCoordinates (int depth, double x, double y)**

Returns the intensity of the cube.

double getIntensityWorldCoordinates (int depth, double x, double y)

Returns the intensity of the vube at a given point in world coordinates. If the pixel is flagged out, NaN is given back.

Arguments

`int` **depth** `[INPUT, MANDATORY, default=no default value]`

The depth (in Pixel coordinates), as int

`double` **x** `[INPUT, MANDATORY, default=no default value]`

The first world coordinate, as double

`double` **y** `[INPUT, MANDATORY, default=no default value]`

The second world coordinate, as double

Return

double

The intensity

setIntensity (int depth, int row, int column, Number value)

Sets the intensity of the image.

Sets the intensity of the image at a given point.

Arguments

`int` **depth** `[INPUT, MANDATORY, default=no default value]`

The depth, as int

`int` **row** `[INPUT, MANDATORY, default=no default value]`

Row index, as int

`int` **column** `[INPUT, MANDATORY, default=no default value]`

Column index, as int

`Number` **value** `[INPUT, MANDATORY, default=no default value]`

The value the pixel should get, as Number

Number getPixel (int depth, int row, int column)

Returns 1 pixel value.

Returns 1 pixel value of the image.

Arguments

`int` **depth** `[INPUT, MANDATORY, default=no default value]`

The depth, as int

`int` **row** `[INPUT, MANDATORY, default=no default value]`

Row index, as int

`int` **column** `[INPUT, MANDATORY, default=no default value]`

Column index, as int

Return

Number

1 pixel value of the image.

int getDepth

Returns the number of layers.

int getDepth

Returns the depth of this SimpleCube. This is the first axis (slowly varying axis).

Return

int

Returns the depth of this SimpleCube.

Cube getPreview (float res)

Returns a preview of the cube.

Returns a new SimpleCube, but with a lower resolution if res is smaller than 1. A higher resolution is returned, if res is larger than 1. The spatial regridding is based on linear interpolation which will, in general, not be suitable for scientific purposes.

Argument

`float` **res** `[INPUT, MANDATORY, default=no default value]`

The factor to multiply the resolution, as float

Return

Cube

The preview of the cube

AbstractOrdered3dData getError

Returns the error of the cube as a Numeric3d.

Returns the error of the cube as a Numeric3d containing the error of every pixel.

Return

AbstractOrdered3dData

The error

AbstractOrdered3dData getExposure

Returns the exposure of the cube as a Numeric3d.

Returns the exposure of the cube as a Numeric3d containing the exposure of every pixel.

Return

AbstractOrdered3dData

The exposure

AbstractOrdered3dData getCoverage

Returns the coverage of the cube as an AbstractOrdered3dData.

Returns the coverage of the cube as an AbstractOrdered3dData containing the coverage of every pixel.

Return

AbstractOrdered3dData

The coverage

Flag getFlag

Returns the flag of the cube.

Flag getFlag

Returns the flag of the cube.

Return

Flag

The flag

int getHeight

Returns the height.

Returns the height of this SimpleCube.

Return

int

Returns the height of this SimpleCube.

AbstractOrdered3dData getImage

Returns the cube as a Numeric3d.

Returns the cube as a Numeric3d containing the data of the cube.

Return

AbstractOrdered3dData

The cube as a Numeric3d

Unit getUnit

Returns the unit.

Returns the unit of the cube. The unit of the errors of this cube is the same as the unit of the cube.

Return

Unit

The unit

Unit getUnit (String identifier)

Returns the unit.

Returns the unit of the cube, the coverage or the exposure. The unit of the errors of this cube is the same as the unit of the cube.

Argument

String **identifier** [INPUT, MANDATORY, default=no default value]

A String : IMAGE, COVERAGE or EXPOSURE

Return

Unit

The unit

int getWidth

Returns the width.

Returns the width of this SimpleImage.

***int* getWidth**

Return

int

Returns the width of this SimpleImage.

***boolean* hasError**

Checks whether the cube has an error.

Returns true if the cube has an error.

Return

boolean

True if the error is set.

***boolean* hasExposure**

Checks whether the cube has an exposure.

Returns true if the cube has an exposure.

Return

boolean

True if the exposure is set.

***boolean* hasCoverage**

Checks whether the cube has a coverage.

Returns true if the cube has a coverage.

Return

boolean

True if the coverage is set.

***boolean* hasFlag**

Checks when the cube has a flag.

Returns true if the cube has a flag.

Return

boolean

True if the cube has a flag.

setUnit (Unit<?> unit)

Sets the unit.

Sets the unit of the cube. Adapting the unit of the cube will also adapt the unit of the errors on the cube.

Argument

Unit<?> **unit** [INPUT, MANDATORY, default=no default value]

The unit of the image/cube

setUnit (String unit)

Sets the unit.

Sets the unit of the image. Adapting the unit of the image will also adapt the unit of the errors on the image.

Argument

`String` **unit** `[INPUT, MANDATORY, default=no default value]`

The unit of the image/cube

setUnit (String identifier, Unit<?> unit)

Sets the unit.

Sets the unit of the cube, coverage or exposure. Adapting the unit of the cube will also adapt the unit of the errors on the cube.

Arguments

`String` **identifier** `[INPUT, MANDATORY, default=no default value]`

The identifier : IMAGE, COVERAGE or EXPOSURE

`Unit&lt;?&gt;` **unit** `[INPUT, MANDATORY, default=no default value]`

The unit of the cube

setUnit (String identifier, String unit)

Sets the unit.

Sets the unit of the image, coverage or exposure. Adapting the unit of the image will also adapt the unit of the errors on the image.

Arguments

`String` **identifier** `[INPUT, MANDATORY, default=no default value]`

The identifier : IMAGE, COVERAGE or EXPOSURE

`String` **unit** `[INPUT, MANDATORY, default=no default value]`

The unit of the image/cube

removeError

Removes the error.

Removes the error, if an error exists.

removeExposure

Removes the exposure.

Removes the exposure, if an exposure exists.

removeCoverage

Removes the coverage.

Removes the coverage, if a coverage exists.

removeFlag

Removes the flag.

Removes the flag, if a flag exists.

setImage (AbstractOrdered3dData cube)

Sets the cube.

Sets the cube. If the new cube has other dimensions than the original cube, the errors, exposure and flag are removed. If the new cube has the same dimensions, then the old values for the errors, exposure and flag are kept.

Argument

`AbstractOrdered3dData` **`cube`** `[INPUT, MANDATORY, default=no default value]`

An AbstractOrdered3dData describing the cube.

setError (AbstractOrdered3dData error)

Sets the error.

Sets the errors of the cube. The Numeric3d containing the errors should have the same dimensions as the cube. If this is not the case, the errors will not be adapted.

Argument

`AbstractOrdered3dData` **`error`** `[INPUT, MANDATORY, default=no default value]`

An AbstractOrdered3dData describing the errors of the cube.

setError (AbstractOrdered3dData error, String description)

Sets the error.

Sets the errors of the cube. The Numeric3d containing the errors should have the same dimensions as the cube. If this is not the case, the errors will not be adapted. The errors are also described.

Arguments

`AbstractOrdered3dData` **`error`** `[INPUT, MANDATORY, default=no default value]`

An AbstractOrdered3dData describing the errors of the cube.

`String` **`description`** `[INPUT, MANDATORY, default=no default value]`

The description of the errors

setFlag (Flag flag, String description)

Sets the flag.

Sets the flag of every pixel of the cube. The Flag should have the same dimensions as the cube. If this is not the case, the Flag will not be adapted. The flag is also described.

Arguments

`Flag` **`flag`** `[INPUT, MANDATORY, default=no default value]`

The Flag.

`String` **`description`** `[INPUT, MANDATORY, default=no default value]`

The description of the flag.

setFlag (Flag flag)

Sets the flag.

Sets the flag of every pixel of the cube. The Flag should have the same dimensions as the cube. If this is not the case, the Flag will not be adapted.

Argument

setFlag (Flag flag)

`Flag` **`flag`** `[INPUT, MANDATORY, default=no default value]`

The Flag.

setExposure (AbstractOrdered3dData exposure, String description, Unit<?> unit)

Sets the exposure.

Sets the exposure of every pixel of the cube. The Numeric3d containing the exposure should have the same dimensions as the cube. If this is not the case, the exposure will not be adapted.

Arguments

`AbstractOrdered3dData` **`exposure`** `[INPUT, MANDATORY, default=no default value]`

An AbstractOrdered3dData describing the exposure of the cube.

`String` **`description`** `[INPUT, MANDATORY, default=no default value]`

The exposure of every pixel of the cube.

`Unit&lt;?&gt;` **`unit`** `[INPUT, MANDATORY, default=no default value]`

The unit for the exposure

setExposure (AbstractOrdered3dData exposure, String description)

Sets the exposure.

Sets the exposure of every pixel of the cube. The Numeric3d containing the exposure should have the same dimensions as the cube. If this is not the case, the exposure will not be adapted.

Arguments

`AbstractOrdered3dData` **`exposure`** `[INPUT, MANDATORY, default=no default value]`

An AbstractOrdered3dData describing the exposure of the cube.

`String` **`description`** `[INPUT, MANDATORY, default=no default value]`

The exposure of every pixel of the cube.

setExposure (AbstractOrdered3dData exposure)

Sets the exposure.

Sets the exposure of every pixel of the cube. The Numeric3d containing the exposure should have the same dimensions as the cube. If this is not the case, the exposure will not be adapted.

Argument

`AbstractOrdered3dData` **`exposure`** `[INPUT, MANDATORY, default=no default value]`

An AbstractOrdered3dData describing the exposure of the Cube.

setCoverage (AbstractOrdered3dData coverage, String description, Unit<?> unit)

Sets the coverage.

Sets the coverage of every pixel of the cube. The AbstractOrdered3dData containing the coverage should have the same dimensions as the cube. If this is not the case, the coverage will not be adapted.

Arguments

setCoverage (AbstractOrdered3dData coverage, String description, Unit<?> unit)

`AbstractOrdered3dData` **`coverage`** `[INPUT, MANDATORY, default=no default value]`

An AbstractOrdered3dData describing the coverage of the cube.

`String` **`description`** `[INPUT, MANDATORY, default=no default value]`

The coverage of every pixel of the cube.

`Unit&lt;?&gt;` **`unit`** `[INPUT, MANDATORY, default=no default value]`

The unit of the coverage

setCoverage (AbstractOrdered3dData coverage, String description)

Sets the coverage.

Sets the coverage of every pixel of the cube. The AbstractOrdered3dData containing the coverage should have the same dimensions as the cube. If this is not the case, the coverage will not be adapted.

Arguments

`AbstractOrdered3dData` **`coverage`** `[INPUT, MANDATORY, default=no default value]`

An AbstractOrdered3dData describing the coverage of the cube.

`String` **`description`** `[INPUT, MANDATORY, default=no default value]`

The coverage of every pixel of the cube.

setCoverage (AbstractOrdered3dData coverage)

Sets the coverage.

Sets the coverage of every pixel of the cube. The AbstractOrdered3dData containing the coverage should have the same dimensions as the cube. If this is not the case, the coverage will not be adapted.

Argument

`AbstractOrdered3dData` **`coverage`** `[INPUT, MANDATORY, default=no default value]`

An AbstractOrdered3dData describing the coverage of the cube.

int[] **getDimensions**

Returns the dimension.

Returns the dimension (depth, width, height) of this SimpleCube.

Return

int[]

Returns the dimension (depth, width, height) of this SimpleCube.

See also

- Developers Manual: herschel.ia.dataset.image.SimpleCube

1.382. simpleFitsReader

Full Name:	herschel.ia.toolbox.util.SimpleFitsReaderTask
Alias:	simpleFitsReader
Type:	Java Task -
Import:	from herschel.ia.toolbox.util import SimpleFitsReaderTask
Category	Input-output

Description

Task to read HCSS products (and simple external FITS).

We recommend that you use `fitsReader` instead of this task. `simpleFitsReader` creates a product or a dataset from a FITS file. This tasks reads non-HCSS (external) fits files as simple HCSS products. If you need to import products (images, cubes, spectra), use FitsReader. fitsReader has a superset of the functionality of this task.

To save products into FITS files use `simpleFitsWriter`.

Examples

Example 1: Read as Herschel if possible else as a standard product

```
filepath = "path_to_file/filename"
product = simpleFitsReader(file=filepath)
```

Example 2: Read a standard product (non Herschel), with readerType argument

```
filepath = "path_to_file/filename"
readertype = SimpleFitsReaderTask.ReaderType.STANDARD
product=simpleFitsReader(file=filepath, reader=readertype)
```

Example 3: Read a standard product (non Herschel), with a String argument

```
filepath = "path_to_file/filename"
product=simpleFitsReader(file=filepath, reader="standard")
```

API Summary

Jython Syntax

```
product = simpleFitsReader(&lt;file&gt; [, &lt;reader&gt;="Herschel then standard"])
```

Properties

String **file** [INPUT, MANDATORY, default=no default value]
Object **reader** [INPUT, OPTIONAL, default="Herschel then standard"]
Annotatable **product** [OUTPUT, MANDATORY, default=no default value]

Limitations

- SimpleFitsReader cannot import complex external FITS files. Use `FitsReader` instead.

- SimpleFitsReader does not support zipped fits (only gzipped). Use `decompress` on those zipped files.

API details

Properties

`String` file [INPUT, MANDATORY, default=no default value]

The path of the FITS file to be read. Can be gzipped.

Object reader [INPUT, OPTIONAL, default="Herschel then standard"]

The strategy used to parse the contents. Allows SimpleFitsReaderTask.ReaderType or String HCSS has priority over STANDARD and RAW as any FITS file can be read as STANDARD but only some of them are also HCSS FITS files. Possible values (you can use strings too, HCSS and Herschel are considered synonyms):

- SimpleFitsReaderTask.ReaderType.HCSS_THEN_STANDARD (same as default) or "Herschel then standard": try to read the FITS Archive as an HCSS FITS file, if it fails then try to read it as an standard FITS file.
- SimpleFitsReaderTask.ReaderType.HCSS or "HCSS": read it as an HCSS FITS file.
- SimpleFitsReaderTask.ReaderType.STANDARD or "Standard": read it as an STANDARD FITS file.
- SimpleFitsReaderTask.ReaderType.RAW: read it as an STANDARD FITS file, keeping the original metadata.

Annotatable product [OUTPUT, MANDATORY, default=no default value]

The Product or Dataset read from the FITS file.

See also

- fitsReader
- decompress
- simpleFitsWriter
- Developers Manual: herschel.ia.toolbox.util.SimpleFitsReaderTask

History

- `2008-07-09 - JCS`: first release
- `2008-08-18 - JDS`: added optional parameter reader (type) to allow transparent failover reading
- `2011-01-24 - JDS`: limitations

1.383. simpleFitsWriter

Full Name:	herschel.ia.toolbox.util.SimpleFitsWriterTask
Alias:	simpleFitsWriter
Type:	Java Task -
Import:	from herschel.ia.toolbox.util import SimpleFitsWriterTask
Category	Input-output

Description

Saves a product in a FITS file.

SimpleFitsWriterTask flow:

- if no filename is given, flow is finished with RuntimeException "File is required."
- if the file already exists and warn is checked: user is asked to confirm overwriting
- if the user does not confirm overwriting (or mode is not interactive), flow is finished with RuntimeException "Execution cancelled."
- if the user asked for compression, the output will be compressed
- Otherwise the product or dataset (wrapped) is saved in HCSS' FITS format in the chosen file

To read FITS files use `fitsReader` (`SimpleFitsReader` is quite limited).

Examples

Example 1: Saving a product into a file

```
p = Product()
f = "path_to_file/filename.fits"
simpleFitsWriter(product=p,file=f)
```

Example 2: Saving a product into a file asking before overwriting

```
p = Product()
f = "path_to_file/filename.fits"
simpleFitsWriter(product=p,file=f,warn=True)
```

Example 3: Saving a product into a zipped file

```
p = Product()
f = "path_to_file/filename.zip"
simpleFitsWriter(product=p,file=f,compression="ZIP")
```

API Summary

Jython Syntax

```
simpleFitsWriter(&lt;product&gt;, &lt;file&gt; [,
&lt;warn&gt;=False, &lt;compression&gt;="NONE"])
```

Properties

Annotatable **product** [INPUT, MANDATORY, default=null]

Properties
String **file** [INPUT, MANDATORY, default=null]
Boolean **warn** [INPUT, OPTIONAL, default=False]
String **compression** [INPUT, OPTIONAL, default="NONE"]

Limitations

SimpleFitsReader does not support zipped fits. SimpleFitsWriter allows it for compatibility with external tools.

API details

Properties

Annotatable product [INPUT, MANDATORY, default=null]
Product or Dataset to be saved. Will not accept Contexts.

String file [INPUT, MANDATORY, default=null]
FITS filename to save the product into.

Boolean warn [INPUT, OPTIONAL, default=False]
If true, asks confirmation before overwriting.

String compression [INPUT, OPTIONAL, default="NONE"]
The type of compression in the output file. Valid values are "NONE", "ZIP" and "GZIP"

See also

- fitsReader
- decompress
- simpleFitsReader
- Developers Manual: herschel.ia.toolbox.util.SimpleFitsWriterTask

History

- 2008-07-08 - JCS: first release
- 2008-08-19 - JDS: SCR 4573: asking before overwriting
- 2009-01-07 - JDS: SCR-8864: support compression
- 2010-06-10 - JDS: reject Contexts

1.384. SimpleImage

Full Name:	herschel.ia.dataset.image.SimpleImage
Type:	Java Class -
Import:	from herschel.ia.dataset.image import SimpleImage
Category	Images

Description

A Product describing images.

SimpleImage is a Product describing images.

Example

Example 1: Creating a SimpleImage

```
s = SimpleImage()
s.setImage(myDouble2dImage)
```

API Summary

Constructors
SimpleImage The standard constructor.
SimpleImage (SimpleImage copy) Copy constructor.
SimpleImage (String description) Constructor with a description.

Methods
copy Returns a copy.
setImage (AbstractOrdered2dData image, Unit unit, String description) Sets the image.
setWcs (Wcs wcs) Sets the world coordinates system.
Wcs **getWcs** Returns the World Coordinates System.
setImage (AbstractOrdered2dData image, Unit unit) Sets the image.
double **getIntensity** (double row, double column) Returns the intensity.
double **getIntensityWorldCoordinates** (double x, double y) Returns the intensity of the image.
setIntensity (int row, int column, Number value) Sets the intensity of the image.

Methods
Number **getPixel** (int row, int column) Returns 1 pixel value.
Image **getPreview** (float res) Returns a preview of the image.
AbstractOrdered2dData **getError** Returns the error of the image as a Numeric2d.
AbstractOrdered2dData **getExposure** Returns the exposure of the image as a Numeric2d.
AbstractOrdered2dData **getCoverage** Returns the coverage of the image as an AbstractOrdered2dData.
Flag **getFlag** Returns the flag of the image.
int **getHeight** Returns the height.
AbstractOrdered2dData **getImage** Returns the image as a Numeric2d.
Unit **getUnit** Returns the unit.
Unit **getUnit** (String identifier) Returns the unit.
int **getWidth** Returns the width.
double **getWavelength** Returns the reference wavelength.
double **getWavelength** (Length unit) Returns the reference wavelength.
double **getWavelength** (Frequency unit) Returns the reference wavelength.
boolean **hasError** Checks whether the image has an error.
boolean **hasExposure** Checks whether the image has an exposure.
boolean **hasCoverage** Checks whether the image has a coverage.
boolean **hasFlag** Checks whether the image has a flag.
setUnit (Unit<?> unit) Sets the unit.
setUnit (String unit) Sets the unit.
setUnit (String identifier, Unit<?> unit)

Methods
Sets the unit.
setUnit (String identifier, String unit)
Sets the unit.
setWavelength (double wavelength)
Sets the reference wavelength.
setWavelength (double wavelength, Length unit)
Sets the reference wavelength.
removeError
Removes the error.
removeExposure
Removes the exposure.
removeCoverage
Removes the coverage.
removeFlag
Removes the flag.
setImage (AbstractOrdered2dData image)
Sets the image.
setError (AbstractOrdered2dData error)
Sets the error.
setError (AbstractOrdered2dData error, String description)
Sets the error.
setFlag (Flag flag, String description)
Sets the flag.
setFlag (Flag flag)
Sets the flag.
setExposure (AbstractOrdered2dData exposure, String description, Unit<?> unit)
Sets the exposure.
setExposure (AbstractOrdered2dData exposure, String description)
Sets the exposure.
setExposure (AbstractOrdered2dData exposure)
Sets the exposure.
setCoverage (AbstractOrdered2dData coverage, String description, Unit<?> unit)
Sets the coverage.
setCoverage (AbstractOrdered2dData coverage, String description)
Sets the coverage.
setCoverage (AbstractOrdered2dData coverage)
Sets the coverage.
double **getFrequency**
Returns the reference frequency.
double **getFrequency** (Frequency freq)

Methods
Returns the reference frequency.
setFrequency (double frequency)
Sets the reference frequency.
setFrequency (double frequency, Frequency unit)
Sets the reference frequency.
int[] **getDimensions**
Returns the dimension.
Double2d **getImageData**
Returns the Image data.

API Details

Constructors

SimpleImage

The standard constructor.

A constructor which creates a standard SimpleImage. The standard SimpleImage consists of a Numeric2d for the image, no error and no integration time. The dimension of the standard SimpleImage is 0x0. The reference wavelength is not set.

Example

Typical example on how to create an SimpleImage.

```
im = Double2d(10, 15)
er = Double2d(10, 15)
flag = Flag(10, 15)
wcs = Wcs()
image=SimpleImage(description="ngc 6992", image=im, flag=flag, error=er,
 wcs=wcs)
```

SimpleImage (SimpleImage copy)

Copy constructor.

Constructor which makes a copy from an existent SimpleImage.

Argument

SimpleImage **copy** [INPUT, MANDATORY, default=no default value]

The original SimpleImage, as SimpleImage

SimpleImage (String description)

Constructor with a description.

Creates a SimpleImage with a given description.

Argument

String **description** [INPUT, MANDATORY, default=no default value]

The descriprion of the SimpleImage, as String

Methods

copy

Returns a copy.

copy

Returns a copy from this SimpleImage.

setImage (AbstractOrdered2dData image, Unit unit, String description)

Sets the image.

Sets the image. You can give a description to the image and give a unit. The given unit will automatically also be used for the errors on this image. If the new image has other dimensions than the original image, the errors, exposure and the flag are removed. If the new image has the same dimensions, then the old values for the errors, exposure and flag are kept (but the unit of the errors is not).

Arguments

`AbstractOrdered2dData` **`image`** `[INPUT, MANDATORY, default=no default value]`

A Numeric2d describing the image, as AbstractOrdered2dData

`Unit` **`unit`** `[INPUT, MANDATORY, default=no default value]`

The unit of the image, as Unit

`String` **`description`** `[INPUT, MANDATORY, default=no default value]`

A string describing the image, as String

setWcs (Wcs wcs)

Sets the world coordinates system.

Sets the world coordinates system of the image.

Argument

`Wcs` **`wcs`** `[INPUT, MANDATORY, default=no default value]`

The Wcs object, as Wcs

Wcs **getWcs**

Returns the World Coordinates System.

Returns the World Coordinates System of the image.

Return

Wcs

The World Coordinates System of the image.

setImage (AbstractOrdered2dData image, Unit unit)

Sets the image.

Sets the image. You can give a unit. The given unit will automatically also be used for the errors on this image. If the new image has other dimensions than the original image, the errors, the exposure and the flag are removed. If the new image has the same dimensions, then the old values for the errors, exposure and the flag are kept (but the unit of the errors is not).

Arguments

`AbstractOrdered2dData` **`image`** `[INPUT, MANDATORY, default=no default value]`

A Numeric2d describing the image, as AbstractOrdered2dData

`Unit` **`unit`** `[INPUT, MANDATORY, default=no default value]`

setImage (AbstractOrdered2dData image, Unit unit)

The unit of the image, as Unit

***double* getIntensity (double row, double column)**

Returns the intensity.

Returns the intensity of the SimpleImage at a given point. If the pixel is flagged out, NaN is given back.

Arguments

`double` **row** `[INPUT, MANDATORY, default=no default value]`

The row of the image, as double

`double` **column** `[INPUT, MANDATORY, default=no default value]`

The column of the image, as double

Return

double

The intensity of the image at the given point.

***double* getIntensityWorldCoordinates (double x, double y)**

Returns the intensity of the image.

Returns the intensity of the image at a given point in world coordinates. If the pixel is flagged out, NaN is given back.

Arguments

`double` **x** `[INPUT, MANDATORY, default=no default value]`

The first world coordinate, as double

`double` **y** `[INPUT, MANDATORY, default=no default value]`

The second world coordinate, as double

Return

double

The intensity

setIntensity (int row, int column, Number value)

Sets the intensity of the image.

Sets the intensity of the image at a given point.

Arguments

`int` **row** `[INPUT, MANDATORY, default=no default value]`

Row index, as int

`int` **column** `[INPUT, MANDATORY, default=no default value]`

Column index, as int

`Number` **value** `[INPUT, MANDATORY, default=no default value]`

The value the pixel should get, as Number

***Number* getPixel (int row, int column)**

Returns 1 pixel value.

Returns 1 pixel value of the image.

Number getPixel (int row, int column)

Arguments

`int` **`row`** `[INPUT, MANDATORY, default=no default value]`

Row index, as int

`int` **`column`** `[INPUT, MANDATORY, default=no default value]`

Column index, as int

Return

Number

1 pixel value of the image.

Image getPreview (float res)

Returns a preview of the image.

Returns a new SimpleImage, but with a lower resolution if res is smaller than 1. A higher resolution is returned, if res is larger than 1. The spatial regridding is based on linear interpolation which will, in general, not be suitable for scientific purposes.

Argument

`float` **`res`** `[INPUT, MANDATORY, default=no default value]`

The factor to multiply the resolution, as float

Return

Image

The preview of the image

AbstractOrdered2dData getError

Returns the error of the image as a Numeric2d.

Returns the error of the image as a Numeric2d containing the error of every pixel.

Return

AbstractOrdered2dData

The error

AbstractOrdered2dData getExposure

Returns the exposure of the image as a Numeric2d.

Returns the exposure of the image as a Numeric2d containing the exposure of every pixel.

Return

AbstractOrdered2dData

The exposure

AbstractOrdered2dData getCoverage

Returns the coverage of the image as an AbstractOrdered2dData.

Returns the coverage of the image as an AbstractOrdered2dData containing the coverage of every pixel.

Return

AbstractOrdered2dData

***AbstractOrdered2dData* getCoverage**

The coverage

***Flag* getFlag**

Returns the flag of the image.

Returns the flag of the image.

Return

Flag

The flag

***int* getHeight**

Returns the height.

Returns the height of this SimpleImage.

Return

int

The height of this SimpleImage.

***AbstractOrdered2dData* getImage**

Returns the image as a Numeric2d.

Returns the image as a Numeric2d containing the data of the image.

Return

AbstractOrdered2dData

The image as a Numeric2d

***Unit* getUnit**

Returns the unit.

Returns the unit of the image. The unit of the errors of this image is the same as the unit of the image.

Return

Unit

The unit

***Unit* getUnit (String identifier)**

Returns the unit.

Returns the unit of the image, the coverage or the exposure. The unit of the errors of this image is the same as the unit of the image.

Argument

String **identifier** [INPUT, MANDATORY, default=no default value]

IMAGE, COVERAGE or EXPOSURE

Return

Unit

The unit

int getWidth

Returns the width.

Returns the width of this SimpleImage.

Return

int

The width of this SimpleImage.

double getWavelength

Returns the reference wavelength.

Returns the reference wavelength of the image.

Return

double

The reference wavelength

double getWavelength (Length unit)

Returns the reference wavelength.

Returns the reference wavelength of the image.

Argument

`Length` **`unit`** `[INPUT, MANDATORY, default=no default value]`

Length unit in which the wavelength should be returned.

Return

double

The reference wavelength

double getWavelength (Frequency unit)

Returns the reference wavelength.

Returns the reference wavelength of the image.

Argument

`Frequency` **`unit`** `[INPUT, MANDATORY, default=no default value]`

Frequency unit in which the wavelength should be returned.

Return

double

The reference wavelength

boolean hasError

Checks whether the image has an error.

Returns true if the image has an error.

Return

boolean

True if the error is set.

boolean hasExposure

Checks whether the image has an exposure.

Returns true if the image has an exposure.

Return

boolean

True if the exposure is set.

boolean hasCoverage

Checks whether the image has a coverage.

Returns true if the image has a coverage.

Return

boolean

True if the coverage is set.

boolean hasFlag

Checks whether the image has a flag.

Returns true if the image has a flag.

Return

boolean

True if the image has a flag.

setUnit (Unit<?> unit)

Sets the unit.

Sets the unit of the image. Adapting the unit of the image will also adapt the unit of the errors on the image.

Argument

`Unit&lt;?&gt;` **unit** [INPUT, MANDATORY, default=no default value]

The unit of the image/cube

setUnit (String unit)

Sets the unit.

Sets the unit of the image. Adapting the unit of the image will also adapt the unit of the errors on the image.

Argument

`String` **unit** [INPUT, MANDATORY, default=no default value]

The unit of the image/cube

setUnit (String identifier, Unit<?> unit)

Sets the unit.

Sets the unit of the image, coverage or exposure. Adapting the unit of the image will also adapt the unit of the errors on the image.

setUnit (String identifier, Unit<?> unit)

Arguments

`String` **identifier** [INPUT, MANDATORY, default=no default value]

The identifier : IMAGE, COVERAGE or EXPOSURE

`Unit&lt;?&gt;` **unit** [INPUT, MANDATORY, default=no default value]

The unit of the image/cube

setUnit (String identifier, String unit)

Sets the unit.

Sets the unit of the image, coverage or exposure. Adapting the unit of the image will also adapt the unit of the errors on the image.

Arguments

`String` **identifier** [INPUT, MANDATORY, default=no default value]

The unit of the image/cube

`String` **unit** [INPUT, MANDATORY, default=no default value]

The identifier : IMAGE, COVERAGE or EXPOSURE

setWavelength (double wavelength)

Sets the reference wavelength.

Set the reference wavelength of the image in microns.

Argument

`double` **wavelength** [INPUT, MANDATORY, default=no default value]

A double describing the wavelength in microns

setWavelength (double wavelength, Length unit)

Sets the reference wavelength.

Set the reference wavelength of the image.

Arguments

`double` **wavelength** [INPUT, MANDATORY, default=no default value]

The wavelength

`Length` **unit** [INPUT, MANDATORY, default=no default value]

The unit of the wavelength

removeError

Removes the error.

Removes the error, if an error exists.

removeExposure

Removes the exposure.

Removes the exposure, if an exposure exists.

removeCoverage

Removes the coverage.

removeCoverage

Removes the coverage, if a coverage exists.

removeFlag

Removes the flag.

Removes the flag, if a flag exists.

setImage (AbstractOrdered2dData image)

Sets the image.

Sets the image. If the new image has other dimensions than the original image, the errors, exposure and flag are removed. If the new image has the same dimensions, then the old values for the errors, exposure and flag are kept.

Argument

`AbstractOrdered2dData` **`image`** `[INPUT, MANDATORY, default=no default value]`

An AbstractOrdered2dData describing the image.

setError (AbstractOrdered2dData error)

Sets the error.

Sets the errors of the image. The Numeric2d containing the errors should have the same dimensions as the image. If this is not the case, the errors will not be adapted.

Argument

`AbstractOrdered2dData` **`error`** `[INPUT, MANDATORY, default=no default value]`

An AbstractOrdered2dData describing the errors of the image.

setError (AbstractOrdered2dData error, String description)

Sets the error.

Sets the errors of the image. The AbstractOrdered2dData containing the errors should have the same dimensions as the image. If this is not the case, the errors will not be adapted. The errors are also described.

Arguments

`AbstractOrdered2dData` **`error`** `[INPUT, MANDATORY, default=no default value]`

An AbstractOrdered2dData describing the errors of the image.

`String` **`description`** `[INPUT, MANDATORY, default=no default value]`

The description of the errors

setFlag (Flag flag, String description)

Sets the flag.

Sets the flag of every pixel of the image. The Flag should have the same dimensions as the image. If this is not the case, the Flag will not be adapted. The flag is also described.

Arguments

`Flag` **`flag`** `[INPUT, MANDATORY, default=no default value]`

The Flag.

setFlag (Flag flag, String description)

`String` **description** [INPUT, MANDATORY, default=no default value]

The description of the Flag.

setFlag (Flag flag)

Sets the flag.

Sets the flag of every pixel of the image. The Flag should have the same dimensions as the image. If this is not the case, the Flag will not be adapted.

Argument

`Flag` **flag** [INPUT, MANDATORY, default=no default value]

The Flag.

setExposure (AbstractOrdered2dData exposure, String description, Unit<?> unit)

Sets the exposure.

Sets the exposure of every pixel of the image. The Numeric2d containing the exposure should have the same dimensions as the image. If this is not the case, the exposure will not be adapted.

Arguments

`AbstractOrdered2dData` **exposure** [INPUT, MANDATORY, default=no default value]

An AbstractOrdered2dData describing the exposure of the image.

`String` **description** [INPUT, MANDATORY, default=no default value]

The exposure of every pixel of the image.

`Unit&lt;?&gt;` **unit** [INPUT, MANDATORY, default=no default value]

The unit for the exposure

setExposure (AbstractOrdered2dData exposure, String description)

Sets the exposure.

Sets the exposure of every pixel of the image. The Numeric2d containing the exposure should have the same dimensions as the image. If this is not the case, the exposure will not be adapted.

Arguments

`AbstractOrdered2dData` **exposure** [INPUT, MANDATORY, default=no default value]

An AbstractOrdered2dData describing the exposure of the image.

`String` **description** [INPUT, MANDATORY, default=no default value]

The exposure of every pixel of the image.

setExposure (AbstractOrdered2dData exposure)

Sets the exposure.

Sets the exposure of every pixel of the image. The Numeric2d containing the exposure should have the same dimensions as the image. If this is not the case, the exposure will not be adapted.

Argument

`AbstractOrdered2dData` **exposure** [INPUT, MANDATORY, default=no default value]

setExposure (AbstractOrdered2dData exposure)

An AbstractOrdered2dData describing the exposure of the image.

setCoverage (AbstractOrdered2dData coverage, String description, Unit<?> unit)

Sets the coverage.

Sets the coverage of every pixel of the image. The AbstractOrdered2dData containing the coverage should have the same dimensions as the image. If this is not the case, the coverage will not be adapted.

Arguments

`AbstractOrdered2dData` **`coverage`** `[INPUT, MANDATORY, default=no default value]`

An AbstractOrdered2dData describing the coverage of the image.

`String` **`description`** `[INPUT, MANDATORY, default=no default value]`

The coverage of every pixel of the image.

`Unit&lt;?&gt;` **`unit`** `[INPUT, MANDATORY, default=no default value]`

The unit of the coverage

setCoverage (AbstractOrdered2dData coverage, String description)

Sets the coverage.

Sets the coverage of every pixel of the image. The AbstractOrdered2dData containing the coverage should have the same dimensions as the image. If this is not the case, the coverage will not be adapted.

Arguments

`AbstractOrdered2dData` **`coverage`** `[INPUT, MANDATORY, default=no default value]`

An AbstractOrdered2dData describing the coverage of the image.

`String` **`description`** `[INPUT, MANDATORY, default=no default value]`

The coverage of every pixel of the image.

setCoverage (AbstractOrdered2dData coverage)

Sets the coverage.

Sets the coverage of every pixel of the image. The AbstractOrdered2dData containing the coverage should have the same dimensions as the image. If this is not the case, the coverage will not be adapted.

Argument

`AbstractOrdered2dData` **`coverage`** `[INPUT, MANDATORY, default=no default value]`

An AbstractOrdered2dData describing the coverage of the image.

double **getFrequency**

Returns the reference frequency.

Returns the reference frequency of the image.

Return

double

double getFrequency

The reference frequency of the image.

double getFrequency (Frequency freq)

Returns the reference frequency.

Returns the reference frequency of the image.

Argument

`Frequency` **`freq`** `[INPUT, MANDATORY, default=no default value]`

The frequency.

Return

double

The reference frequency of the image.

setFrequency (double frequency)

Sets the reference frequency.

Set the reference frequency of the image in gigahertz.

Argument

`double` **`frequency`** `[INPUT, MANDATORY, default=no default value]`

The reference frequency of the image, described as a double in GigaHertz

setFrequency (double frequency, Frequency unit)

Sets the reference frequency.

Set the reference frequency of the image.

Arguments

`double` **`frequency`** `[INPUT, MANDATORY, default=no default value]`

The reference frequency of the image, described as a Frequency

`Frequency` **`unit`** `[INPUT, MANDATORY, default=no default value]`

The unit of the frequency

int[] getDimensions

Returns the dimension.

Returns the dimension (width, height) of this SimpleImage.

Return

int[]

The dimension (width, height) of this SimpleImage.

Double2d getImageData

Returns the Image data.

Returns the image data as a Double2d.

Return

Double2d

The image data as Double2d

See also

- Developers Manual: herschel.ia.dataset.image.SimpleImage

1.385. SimpleSpectrum

Full Name:	herschel.ia.dataset.spectrum.SimpleSpectrum
Type:	Java Class -
Import:	from herschel.ia.dataset.spectrum import SimpleSpectrum
Category	Spectra

Description

SimpleSpectrum is a simple wrapper to transform a Spectrum1d into a Product.

The SimpleSpectrum copies all MetaData of the Spectrum1d to the MetaData of the Product and subsequently makes the MetaData of the Spectrum1d identical to that of the Product.

See also

- Developers Manual: herschel.ia.dataset.spectrum.SimpleSpectrum

History

- `2010-08-23 - DK:.`

1.386. SincGaussModel

Full Name:	herschel.ia.numeric.toolbox.fit.SincGaussModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import SincGaussModel
Category	Mathematics/Fitting

Description

Sinc-Gauss Model.

Convolution of a Sinc function with a Gaussian function.

$$f(x:p) = p_0 * e^{-b^2} * (erf(a - ib) + erf(a + ib)) / 2 / erf(a)$$

where $a = (p_3 / \sqrt{}(2) / p_2)$; $b = ((x - p_1) / \sqrt{}(2) / p_3)$

and where p_0 = amplitude, p_1 = offset, p_2 = width of sinc function (=Distance between first zero-crossings divided by 2π) and p_3 = width of Gaussian function (σ).

As always x = input.

The parameters are initialized at {1.0, 0.0, 1.0, 1.0}.

Parameters 2 and 3 (width of sinc and Gaussian functions) are always kept positive (>=0).

The function cannot be evaluated far away from the peak. Where this is the case, the return value is zero.

The function can be derived as follows:

- Take the Fourier transforms of the sinc and Gaussian functions (setting $\mu = 0$ for the Gaussian). For the sinc function (width = c), this is $\pi c e^{-2\pi i \mu \xi}$ rect($\pi c \xi$), while for the Gaussian this is $\sqrt{}(2)\pi\sigma e^{-2(\pi\sigma\xi)^2}$.
- Multiply these together and take the inverse Fourier transform (convolution theorem). Split the (Fourier transform) exponential term into odd and even components ((f(x) + f(-x)) / 2 + (f(x) - f(-x)) / 2). The odd terms cancel (since the rest of the integrand is even). What is left is an even integrand, so (with the rect function) the limits of the integral can be taken to be between 0 and $1/2c\pi$. Combine the exponential terms into two. By completing the square, the e^{-b^2} term can be taken outside the integral. The integral then evaluates to the two error functions.
- Re-normalise with a new normalisation parameter, such that the peak is 1.0 if $p_0 = 0$. This gives the erf(a) term.

Example

Example 1: SincGaussModel

```
# Model parameters
a = 1.567
mu = 1.12
c = 0.3
sig = 0.566
params = Double1d([a, mu, c, sig])
# For comparison, convolve a Sinc function with a Gaussian function
x = Double1d.range(1501) / 50 - 15.
# Sinc function
```

Example 1: SincGaussModel

```
sincx = a * SIN((x - mu) / c) / ((x - mu) / c)
p = PlotXY()
p.addLayer(LayerXY(x[500:1001], sincx[500:1001], name="sinc"))
p.legend.visible = True
# Gaussian function
gaussx = a * EXP(-(x ** 2) / 2 / sig ** 2)
p.addLayer(LayerXY(x[500:1001], gaussx[500:1001], name="gauss"))
# Convolution of Sinc and Gaussian functions
sgConv = Convolution(gaussx)(sincx)
sgConv /= MAX(sgConv)
sgConv *= a
p.addLayer(LayerXY(x[500:1001], sgConv[500:1001], name="sinc-gauss
 (convolution)", stroke=4))
# Use SincGaussModel
x = Double1d.range(501) / 50 - 5.
sgModel = SincGaussModel(parameters=params)(x)
p.addLayer(LayerXY(x, sgModel, name="sinc-gauss (model)", stroke=2))
# Use values from the convolution to perform a fit and recover the parameters
myModel = SincGaussModel()
myFitter = LevenbergMarquardtFitter(x, myModel)
fitParams = myFitter.fit(sgConv[500:1001])
print "Params expected:", " ".join(["%.5f" % s for s in params])
print "Params fit     :", " ".join(["%.5f" % s for s in fitParams])
sgFit = SincGaussModel(parameters=fitParams)(x)
p.addLayer(LayerXY(x, sgFit, name="sinc-gauss (fit)", stroke=4,
 line=Style.DASHED))
```

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.SincGaussModel

1.387. SINC

Full Name:	herschel.ia.numeric.toolbox.basic.Sinc
Alias:	SINC
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Sinc
Category	Mathematics/Trigonometry

Description

Returns the sinc of a number or array.

The sinc is calculated as follows: y = sin(x) / x for values close to zero (1.0e-9): y = 1.0 - (x^2 / 6.0) + (x^4 / 120.0) + (x^6 / 5040.0) If the input is an array, the output is an array of the same type and size, with sinc values instead of the original elements.

Example

Example 1: Applying SINC to a Float1d

```
x = Float1d([0,0.5,1.0e-10])
print  SINC(x) # [1.0,0.9588511,1.0]
```

API Summary

Jython Syntax

```
&lt;y&gt; = SINC(&lt;x&gt;)
```

Properties

Number or array **x** [INPUT, MANDATORY, default=no default value]

Number or array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Number or array x [INPUT, MANDATORY, default=no default value]

The angle or angles of which to compute the sine, in radians.

Number or array y [OUTPUT, MANDATORY, default=no default value]

The sinc value or values.

See also

- SIN
- SINH
- ARCSIN
- Developers Manual: herschel.ia.numeric.toolbox.basic.Sinc

1.388. SincModel

Full Name:	herschel.ia.numeric.toolbox.fit.SincModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import SincModel
Category	Mathematics/Fitting

Description

Sinc Model.

Also known as Cardinal Sine.

$$f(x:p) = p_0 * \sin((x - p_1) / p_2) / ((x - p_1) / p_2)$$

where p_0 = amplitude, p_1 = offset and p_2 = width (=Distance between first zero-crossings divided by 2 Pi.)

As always x = input.

The parameters are initialized at {1.0, 0.0, 1.0}. Parameter 2 (width) is always kept positive (>=0).

Example

Example 1: SincModel

```
sinc = SincModel()
print sinc.getNumberOfParameters()        # 3
print sinc( Double1d.range(15)-7 )        # sinc function between [-7,+7]
```

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.SincModel

1.389. SineAmpModel

Full Name:	herschel.ia.numeric.toolbox.fit.SineAmpModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import SineAmpModel
Category	Mathematics/Fitting

Description

Find amplitudes/phases for sinusoidal of a given frequency.

$$f(x:p) = p_0 \cos(2 \pi \omega x) + p_1 \sin(2 \pi \omega x)$$

It is a linear model.

Example

Example 1: SineAmpModel

```
sine = SineAmpModel( 150 )        # fixed frequency of 150 Hz
# Use a (linear)Fitter
```

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.SineAmpModel

1.390. SineModel

Full Name:	herschel.ia.numeric.toolbox.fit.SineModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import SineModel
Category	Mathematics/Fitting

Description

Sinusoidal Model.

$$f(x:p) = p_1 * \cos(2 * \pi * p_0 * x) + p_2 * \sin(2 * \pi * p_0 * x)$$

where p_0 = frequency, p_1 = amplitude cosine and p_2 = amplitude sine. As always x = input.

The parameters are initialized at {1.0, 1.0, 1.0}. It is a non-linear model.

See example

Example

Example 1: SineModel

```
sine = SineModel()
print sine.getNumberOfParameters()        # 3
pars = Double1d( [0.1,0,1] )
print sine.setParameters( pars )
print sine( Double1d.range(11) )     # One sine period
pars = Double1d( [0.1,1,0] )
print sine.setParameters( pars )
print sine( Double1d.range(11) )     # One cosine period
```

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.SineModel

1.391. SingularValueDecompositionFitter

Full Name:	herschel.ia.numeric.toolbox.fit.SingularValueDecompositionFitter
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import SingularValueDecompositionFitter
Category	Mathematics/Fitting

Description

SingularValueDecompositionFitter implements a linear fitter based on Singular Value Decomposition (SVD).

SVDFitter is much more robust in case of (nearly) degenerated models. Of course at the cost of more CPU use.

See Numerical Recipes for more information.

Example

Example 1: SingularValueDecompositionFitter (for linear models.)

```
x = Double1d.range(100)
y = Double1d( Int1d.range(100).divide( 4 ) )     # digitization noise
poly = PolynomialModel( 1 )             # line
svdfit = SingularValueDecompositionFitter( x, poly )
svdfit.setThreshold( 1e-8 )             # for detecting degeneracy
param = svdfit.fit( y )
print svdfit.hasDegeneracy()
stdev = svdfit.getStandardDeviation()    # stdevs on the parameters
chisq = svdfit.getChiSquared()
svd     = svdfit.getSingularValueDecomposition()            # the SVD
 decomposition
```

Limitations

The SVDFitter does **not** work with limits.

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.SingularValueDecompositionFitter

1.392. SingularValueDecomposition

Full Name:	herschel.ia.numeric.toolbox.matrix.SingularValueDecomposition
Alias:	SingularValueDecomposition
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.matrix import SingularValueDecomposition
Category	Mathematics/Matrices

Description

SingularValueDecompositon

Wraps the Jama class for DP. For an m-by-n matrix A with m >= n, the singular value decomposition is an m-by-n orthogonal matrix U, an n-by-n diagonal matrix S, and an n-by-n orthogonal matrix V so that A = U*S*V'. The singular values, sigma[k] = S[k][k], are ordered so that sigma[0] >= sigma[1] >= ... >= sigma[n-1]. The singular value decomposition always exists, so the constructor will never fail. The matrix condition number and the effective numerical rank can be computed from this decomposition.

Example

Example 1: SingularValueDecomposition

```
A = Double2d([[ 2.0,  1.0,  1.0],[ 4.0, -6.0,  0.0],[-2.0,  7.0,  2.0]])
svd=A.apply(SingularValueDecomposition())
leftSingularValues = svd.u
print leftSingularValues
# [
# [-0.02280776366786053,-0.7589359216500687,-0.6507657587378364],
# [0.6882378213114239,-0.4840524707277569,0.5403905133316635],
# [-0.7251265456683931,-0.4355565088613817,0.5333685595866234]
# ]
rightSingularValues = svd.v
print rightSingularValues
# [
# [0.4064876334980915,-0.810943043763452,-0.42087906052070156],
# [-0.9022337235002555,-0.28366605734965333,-0.32481975938444235],
# [-0.1440212206307982,-0.5117664972224835,0.8469669062771251]
# ]
singularValues = svd.singularValues
print singularValues
# array([10.228082004532315, 3.185141950908502, 0.49113059477592597], double)
diagonalMatrixOfSingularValues = svd.s
print diagonalMatrixOfSingularValues
# [
# [10.228082004532315,0.0,0.0],
# [0.0,3.185141950908502,0.0],
# [0.0,0.0,0.49113059477592597]
# ]
norm2 = svd.norm2
print norm2
# 10.228082004532315
cond = svd.cond
print cond
# 20.825585115907487
rank = svd.rank
print rank
# 3.0
```

API Summary

Jython Syntax
A=Double2d() svd=B.apply(SingularValueDecomposition()) leftSingularValues = svd.u rightSingularValues = svd.v singularValues = svd.singularValues diagonalMatrixOfSingularValues = svd.s norm2 = svd.norm2 cont = cvs.cond rank = svd.rank

Property
Double2d **A** [INPUT, MANDATORY, default=no default value]

API details

Property

Double2d A [INPUT, MANDATORY, default=no default value]
Input must be a Double2d or Float2d array.

See also

- Developers Manual: herschel.ia.numeric.toolbox.matrix.SingularValueDecomposition

1.393. SIN

Full Name:	herschel.ia.numeric.toolbox.basic.Sin
Alias:	SIN
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Sin
Category	Mathematics/Trigonometry

Description

Returns the sine of a number or array.

If the input is an array, the output is an array of the same type and size, with sine values instead of the original elements. For complex numbers, the sine is computed for the real and imaginary part.

Example

Example 1: Applying SIN to a Float1d

```
x = Float1d([0,0.5])
print  SIN(x) # [0.0,0.47942555]
```

API Summary

Jython Syntax
<y> = SIN(<x>)

Properties
Number or array **x** [INPUT, MANDATORY, default=no default value]
Number or array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Number or array x [INPUT, MANDATORY, default=no default value]

The angle or angles of which to compute the sine, in radians.

Number or array y [OUTPUT, MANDATORY, default=no default value]

The sine value or values.

See also

- ARCSIN
- COS
- SINH
- TAN

- Developers Manual: herschel.ia.numeric.toolbox.basic.Sin

1.394. SINH

Full Name:	herschel.ia.numeric.toolbox.basic.SinH
Alias:	SINH
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import SinH
Category	Mathematics/Trigonometry

Description

Returns the hyperbolic sine of a number or array.

If the input is an array, the output is an array of the same type and size, with hyperbolic sine values instead of the original elements. For complex numbers, the hyperbolic sine is computed for the real and imaginary part.

API Summary

Jython Syntax
`&lt;y&gt; = SINH(&lt;x&gt;)`

Properties
`Number or array x [INPUT, MANDATORY, default=no default value]`
`Number or array y [OUTPUT, MANDATORY, default=no default value]`

API details

Properties

`Number or array x [INPUT, MANDATORY, default=no default value]`
The angle or angles of which to compute the hyperbolic sine, in radians.

`Number or array y [OUTPUT, MANDATORY, default=no default value]`
The hyperbolic sine value or values.

See also

- SIN
- ARCSIN
- Developers Manual: herschel.ia.numeric.toolbox.basic.SinH

1.395. SkewGaussModel

Full Name:	herschel.ia.numeric.toolbox.fit.SkewGaussModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import SkewGaussModel
Category	Mathematics/Fitting

Description

A class that defines a skew gaussian model.

A skew gaussian model is defined as

$f(x:p) = p_0 * \exp(-0.5 * ((x - p_1) / p_2)^2) * (1 + \mathrm{erf}(p_3 * ((x - p_1) / p_2) / \mathrm{SQRT}(2)))$

with p_0 = amplitude

p_1 = x-shift

p_2 = sigma

p_3 = scale or skewness

API Summary

Constructor
SkewGaussModel
The construction of a new SkewGaussModel.

Method
double **result** (double input, Double1d param)
Returns the result of this SkewGaussModel.

API Details

Constructor

SkewGaussModel

The construction of a new SkewGaussModel.

A new SkewGaussModel is contructed with its parameters set to [1.0, 0.0, 1.0, 0.0].

Method

***double* result (double input, Double1d param)**

Returns the result of this SkewGaussModel.

Returns the result of this SkewGaussModel for the given input and the given parameters.

Arguments

double **input** [INPUT, MANDATORY, default=no default value]

The input value.

double **result (double input, Double1d param)**

Double1d **param** [INPUT, MANDATORY, default=no default value]

The parameters for this SkewGaussModel.

Return

double

Returns the result of this SkewGaussModel for the given input and fit parameters.

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.SkewGaussModel

1.396. SKEWNESS

Full Name:	herschel.ia.numeric.toolbox.basic.Skewness
Alias:	SKEWNESS
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Skewness
Category	Mathematics/Statistics

Description

Returns the skewness of an array.

Skewness measures the degree of asymmetry of a distribution, and is defined as the third central moment divided by the standard deviation raised to the third power. For more information on skewness see this website.

Returns NaN (Not a Number) if any of the array elements is NaN.

Example

Example 1: Applying SKEWNEES to a Float1d

```
x = Float1d([1,3,2,3,4])
print SKEWNESS(x) # -0.19430208
x = Float1d([1,Double.NaN,2,3,4])
print SKEWNESS(x) # NaN
# To remove NaN values (the original array is not modified):
print SKEWNESS(NAN_FILTER(x)) # 0.0 (SKEWNESS is applied to [1,2,3,4])
```

API Summary

Jython Syntax

```
&lt;y&gt; = SKEWNESS(&lt;x&gt;)
```

Properties

Array **x** [INPUT, MANDATORY, default=no default value]

Double **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Array x [INPUT, MANDATORY, default=no default value]

The array of which to compute the skewness. Complex arrays are not allowed.

Double y [OUTPUT, MANDATORY, default=no default value]

The skewness of the input array.

See also

- KURTOSIS

- MEAN
- MEDIAN
- STDDEV
- VARIANCE
- Developers Manual: herschel.ia.numeric.toolbox.basic.Skewness

1.397. SkyMaskToolbox

Full Name:	herschel.ia.gui.image.gui.SkyMaskToolbox
Type:	Java Class -
Import:	from herschel.ia.gui.image.gui import SkyMaskToolbox
Category	Images/Display

Description

SkyMaskToolbox construct a toolbox where the user can annotate an image and create a SkyMask from the annotations.

API Summary

Constructors
SkyMaskToolbox (Display d) Constructor for the SkyMask toolbox.
SkyMaskToolbox (Display d, boolean useAsComponent) Constructs the SkyMask toolbox.

Methods
getComponent Returns the component that contains the SkyMask toolbox.
close Closes the SkyMask toolbox.

API Details

Constructors

SkyMaskToolbox (Display d)

Constructor for the SkyMask toolbox.

Argument

`Display` **`d`** `[INPUT, MANDATORY, default=no default value]`

The Display for which the toolbox should be made.

SkyMaskToolbox (Display d, boolean useAsComponent)

Constructs the SkyMask toolbox.

If useAsComponent is true, the SkyMask toolbox is component based.

Arguments

`Display` **`d`** `[INPUT, MANDATORY, default=no default value]`

The Display for which the toolbox should be made.

`boolean` **`useAsComponent`** `[INPUT, MANDATORY, default=no default value]`

true if the SkyMaskToolbox should be component based, false otherwise.

Methods

getComponent
Returns the component that contains the SkyMask toolbox.

close
Closes the SkyMask toolbox.

See also

- Developers Manual: herschel.ia.gui.image.gui.SkyMaskToolbox

1.398. smoothBaseline

Full Name:	herschel.ia.toolbox.spectrum.standingwaves.SmoothBaselineTask
Alias:	smoothBaseline
Type:	Java Task -
Import:	from herschel.ia.toolbox.spectrum.standingwaves import SmoothBaselineTask
Category	Spectra/Analysis

Description

A task that generates a smooth and clipped version of the observation flux.

First define a user-supplied part of mask: mask lines start from 'suppressBegin' and end at 'suppressEnd'. If user did not supply anything, mask is all 1. Now, the task makes more than one pass on the data. The first iteration a Median filter followed by smoothing to eliminate strong spectral features which will be masked is done. On the second iteration smooth the masked spectrum twice and then again define mask.

Example

Example 1: Description

```
from herschel.ia.toolbox.spectrum.standingwaves import FitFringeData
myFreq = Double1d(freq_column.data)
myFlux = Double1d(flux_column.data)
ffData = FitFringeData(myFreq, myFlux)
baseline = smoothBaseline(data=ffData, midcycle=1700000.0)
mask = smoothBaseline.mask
u = ([(598.0,598.1), (599.0,599.1), (600.0,600.1)])
baseline2 = smoothBaseline(data=ffData, midcycle=1700000.0, box=200,
                           usermask=u, plot=False)
mask2 = smoothBaseline.mask
```

API Summary

Properties
FitFringeData **data** [INPUT, MANDATORY, default=no default value]
Double **midcycle** [INPUT, OPTIONAL, default="1700000.0"]
Integer **box** [INPUT, OPTIONAL, default=no default value]
Object **usermask** [INPUT, OPTIONAL, default=no default value]
Boolean **plot** [INPUT, OPTIONAL, default="true"]
Boolean **mhz** [INPUT, OPTIONAL, default="false"]
Boolean **automask** [INPUT, OPTIONAL, default="true"]
FitFringeData **baseline** [OUTPUT, MANDATORY, default=no default value]
Double1d **mask** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

FitFringeData data [INPUT, MANDATORY, default=no default value]

Input FitFringeData which contains: - four 1d-array of wavelength, flux, flag(optional), weight(optional).

Double midcycle [INPUT, OPTIONAL, default="1700000.0"]

- Input mid of fringe search range (per inverse wavenumber in micron).

Integer box [INPUT, OPTIONAL, default=no default value]

- Input smooth box.

Object usermask [INPUT, OPTIONAL, default=no default value]

- Input frequencies of a set of wavelengths to disregard during fringe fitting (eg. at emission lines).

Boolean plot [INPUT, OPTIONAL, default="true"]

- Plot option

Boolean mhz [INPUT, OPTIONAL, default="false"]

- Unit in mhz.

Boolean automask [INPUT, OPTIONAL, default="true"]

- Automatically mask data points. DEFAULT: TRUE. The automatically defined mask is added to any user-defined mask. - If set to FALSE, only the user-defined mask is used.

FitFringeData baseline [OUTPUT, MANDATORY, default=no default value]

- Smoothed background.

Double1d mask [OUTPUT, MANDATORY, default=no default value]

- Output mask.

See also

- Developers Manual: herschel.ia.toolbox.spectrum.standingwaves.SmoothBaselineTask

1.399. smooth

Full Name:	herschel.ia.toolbox.spectrum.SmoothSpectrumTask
Alias:	smooth
Type:	Java Task -
Import:	from herschel.ia.toolbox.spectrum import SmoothSpectrumTask
Category	Spectra/Analysis

Description

Task for smoothing spectra using a suitable smoothing filter (kernel).

The smoothing is processed by convolving with a suitable kernel function - "Box" and "Gaussian" are available at the moment. The `width` parameterizes these kernel functions: In case there is no `unit` (or `unit="pixels"`), it is just specified the width is rounded to the nearest integer and taken as the width of the box (for the Box smoothing) or as the standard deviation of the Gaussian kernel function for Gaussian smoothing. Without `unit`-parameter set the width is assumed to be expressed in number of bins (channels, wavescale pixels). In case a width unit (e.g. `unit="MHz"`) is specified the width parameter is translated into a number of bins quantity by making reference to the wavescale of the first spectrum found in the spectrum container. The spectra in the spectrum container may consist of separate so-called segments (e.g. the subbands in a HIFI spectrum). In this case, a separate width in bins (pixels, channels) is computed for each segment. This translation of the width 'in frequency units' to a width in 'number of bins' is only approximate if the frequency scale is not perfectly linear (the width is divided by the average pixel width).

Remark: For Gaussian smoothing the width parameter defines the sigma (standard deviation) of the kernel function. This function is truncuated at `+/- 3.717*width` (where the value of the function becomes less than 1/1000 of the peak height).

The spectra passed to the task may also contain pixel dependent weights and flags. These are also processed by the smoothing operation: For the weigths, the same smoothing operation is applied as for the flux values - the flags are propagated to the per pixel flags in the result by applying a bitwise-OR logic. Note that for PACS and SPIRE, we rather have error and mask instead of weights and flags, respectively. Typically, the PACS and SPIRE-specific implementations of the spectrum data containers map the error to weights (e.g. by defining the weights to be inverse squared error). Furthermore, the mask is just mapped 'as is' to flags.

The smoothing task can be configured to ignore flagged bins. This can be achieved by setting the `variant`-parameter to `"flux-flag"` or `"flux-flag-weight"`. The impact of that is that for each bin in the output spectrum only unflagged bins in the input spectrum contribute. In case no unflagged bins are available for a given 'output bin', all bins are used but the flag is propagated - otherwise only the unflagged bins are used and the flag for the result bin set to 0.

Since UR 5.0 the flag values to consider are configurable: Use the `flagToIgnore` (see below). When setting `variant="flag"` no weights will be included in the result spectra.

Similarly, the when including in the variant parameter the 'weight' ("flux-weight" and "flux-flag-weight") a weighted average is computed for the smoothed values.

The other attributes that characterize the spectra in a `SpectrumContainer` (examples for such attributes: integration time, observation time, position in sky, many other instrument-specific parameters) are not processed but just copied to the result.

The smooth task also allows to configure the behavior at the edges of the spectra (i.e. the boundary conditions). This can be configured using the `edge`-parameter:

- `edge="ZEROES"` : The spectra are extrapolated with zero values.

- `edge="CIRCULAR"` : The spectra are extrapolated by using circular boundary conditions which corresponds to glueing the beginning and the end of each spectrum to a circle.
- `edge="REPEAT"` : The spectra are extrapolated by repeating the value found at the edge.
- `edge="CUT"` : The spectra are cut once the kernel window exceeds the spectral range. Half the window width on both sides of the spectra are cut off. Note that this edge behavior is available if `overwrite=False`.

Example

Example 1: Some examples using the smooth task

```
global spectra # defined elsewhere
# no unit for width specified --&gt; width is expressed in pixels
y = smooth(ds=spectra, filter="box", width=10)
y = smooth(ds=spectra, filter="Gaussian", width=20)
# Using units:
y = smooth(ds=spectra, filter="box", width=10.0, unit="pixel")
# Configuring the edge behavior:
y = smooth(ds=spectra, filter="box", width=10, edge="CUT")
# Ignoring flagged pixels:
y = smooth(ds=spectra, filter="box", width=10, variant="flag-weight")
y = smooth(ds=spectra, filter="box", width=10, variant="flag-weight",
 flagToIgnore=21)
# Ignoring NaN flux values:
y = smooth(ds=spectra, filter="box", width=10, variant="flag-weight",
 ignoreNaNs=True)
# Overwrite the input spectra
spectra = smooth(ds=spectra, filter="box", width=10, overwrite=True)
```

API Summary

Properties
SpectrumContainer **ds** [INPUT, MANDATORY, default=no default value.]
SpectrumContainer **result** [OUTPUT, MANDATORY, default=no default value.]
String **filter** [INPUT, OPTIONAL, default=no default value.]
Double **width** [INPUT, MANDATORY, default=no default value.]
String **unit** [INPUT, OPTIONAL, default="pixels".]
String **variant** [INPUT, OPTIONAL, default=no default value.]
Integer **flagToIgnore** [INPUT, OPTIONAL, default=no default value.]
Boolean **ignoreNaNs** [INPUT, OPTIONAL, default=False.]
String **edge** [INPUT, OPTIONAL, default="REPEAT".]
Boolean **overwrite** [INPUT, OPTIONAL, default=False.]

API details

Properties

SpectrumContainer ds [INPUT, MANDATORY, default=no default value.]

Input container with the spectra to be smoothed. Examples of `SpectrumContainer` are `Spectrum1d, Spectrum2d, SpectralSimpleCube`.

SpectrumContainer result [OUTPUT, MANDATORY, default=no default value.]

Output container with the smoothed spectra.

String filter [INPUT, OPTIONAL, default=no default value.]

Filter (convolution) to be applied. Available are: "Box" and "Gaussian".

Double width [INPUT, MANDATORY, default=no default value.]

Smoothing width parameter to configure the filter/convolution kernel. The width is expressed in in units as specified as the `unit` parameter.

String unit [INPUT, OPTIONAL, default="pixels".]

Specify the unit the width is expressed in. The default is pixels, i.e. specify the width in number of pixels. Other values should be compatible with the unit specified for the wave scale (eg width in GHz and wave scale unit given in MHz is ok but width in "cm" not).

String variant [INPUT, OPTIONAL, default=no default value.]

Specify the variant of processing mode you would like to run (including flags / weights / ...). HCSS defaults include: "flux" / "flux-weight" / "flux-flag-weight"

Integer flagToIgnore [INPUT, OPTIONAL, default=no default value.]

Applies only if a 'variant' with the substring 'flag' has been specified. By setting this parameter you can configure which flags should be ignored and which should be propagated as informative flags. For example, if you want to ignore channels that have the flag 1,4 or 16 set you specify a `flagToIgnore=21`. By default, the flagToIgnore is set to 0 which means that any flagged channel is ignored.

Boolean ignoreNaNs [INPUT, OPTIONAL, default=False.]

Once set to true, the task checks the spectra for NaN flux values and ignores them while, accordingly, renormalizing the result for the missing contributions from the kernel funtion.

String edge [INPUT, OPTIONAL, default="REPEAT".]

Parameter to configure the behavior at the edges: "ZEROES", "CIRCULAR", "REPEAT", "CUT", "SHRINK-WIDTH".

Boolean overwrite [INPUT, OPTIONAL, default=False.]

Specify whether the input data container can be reused - the values found therein are overwritten.

See also

- SpectrumContainer
- Developers Manual: herschel.ia.toolbox.spectrum.SmoothSpectrumTask

History

- `2011-08-08 - melchior`: renamed from SmoothSpectrum
- `2012-01-10 - melchior`: add ignoreNaNs-parameter (HCSS-13903)
- `2013-03-13 - melchior`: center parameter deprecated (HCSS-16162)

1.400. SORT

Full Name:	herschel.ia.numeric.toolbox.basic.Sort
Alias:	SORT
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Sort
Category	Arrays and datasets/Manipulation

Description

Sorts the specified array into ascending natural order.

This function does not work on native Jython arrays. Use Numeric arrays instead, such as Int1d.

String1d arrays are sorted in alphabetical order, but with all upper case letters coming before any lower case letter. For example, the array ["string", "Integer", "double"] will be sorted as ["Integer","double","string"].

Two accessory functions provide additional features:

- SORT.IS_SORTED returns *True* if a given array is sorted, *False* otherwise.
- SORT.BY_INDEX returns an array with the *indices* of the sorted array elements, rather than their values. This is useful if, for example, you want to reorder a second array based on the sorted order of the first. Note that SORT.BY_INDEX returns an `Int1d` array object, which you must convert to a `Selection` object to actually sort an array. See the second example for the correct syntax.

Examples

Example 1: Apply SORT to a Double1d

```
x = Double1d([-1,4,2,7,2,-6,-8,3,2])
print SORT(x)    # [-8.0,-6.0,-1.0,2.0,2.0,2.0,3.0,4.0,7.0]
```

Example 2: Use SORT.BY_INDEX

```
x = Double1d([7,1,9])
y = Double1d([3,0,4])
idx = SORT.BY_INDEX(x)
xsorted = x[Selection(idx)]
ysorted = y[Selection(idx)]
print xsorted    # [1.0,7.0,9.0]
print ysorted    # [0.0,3.0,4.0]
```

Example 3: Use SORT.IS_SORTED

```
x=Double1d([-1,4,2,7,2,-6,-8,3,2])
print SORT.IS_SORTED(x)    # 0 (false)
```

API Summary

Jython Syntax

```
&lt;y&gt;=SORT(&lt;x&gt;)
```

Properties
1-D array **x** [INPUT, MANDATORY, default=no default value]
1-D array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

1-D array x [INPUT, MANDATORY, default=no default value]
The array to be sorted.

1-D array y [OUTPUT, MANDATORY, default=no default value]
The sorted array.

See also

- Developers Manual: herschel.ia.numeric.toolbox.basic.Sort

1.401. sortTable

Full Name:	herschel.ia.toolbox.util.SortTableTask
Alias:	sortTable
Type:	Java Task -
Import:	from herschel.ia.toolbox.util import SortTableTask
Category	Arrays and datasets/Manipulation

Description

Returns a new table with rows sorted.

Allows to reorder the rows in a TableDataset

Examples

Example 1: Sorting by specifying the column name

```
t = herschel.ia.dataset.TableDataset(description = "table to sort")
t["one"] = herschel.ia.dataset.Column(herschel.ia.numeric.String1d(["b", "d",
 "a"]))
t["two"] = herschel.ia.dataset.Column(herschel.ia.numeric.Int1d([1, 3, 2]))
sortedTable = sortTable(t, "two")
```

Example 2: Sorting in reverse order by specifying the column index, overwriting original table

```
t = herschel.ia.dataset.TableDataset(description = "table to sort")
t["one"] = herschel.ia.dataset.Column(herschel.ia.numeric.String1d(["b", "d",
 "a"]))
t["two"] = herschel.ia.dataset.Column(herschel.ia.numeric.Int1d([1, 3, 2]))
t = sortTable(t, "1", reverse=True) # Column index is 0 based, parameter is
 string
```

API Summary

Jython Syntax

```
sortedTable = sortTable(&lt;table&gt;, [&lt;columnId&gt;= "0",
&lt;reverse&gt;=False])
```

Properties

TableDataset **table** [INPUT, MANDATORY, default=no default value]
String **columnId** [INPUT, OPTIONAL, default="0" (first column)]
Boolean **reverse** [INPUT, OPTIONAL, default=False]
tabelDataset **sortedTable** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

TableDataset table [INPUT, MANDATORY, default=no default value]

The (rectangular) table to sort (will not be modified)

String columnId [INPUT, OPTIONAL, default="0" (first column)]

The ID of the column to sort the table by. If it can be interpreted as a valid index it will, otherwise it will be a column name.

Boolean reverse [INPUT, OPTIONAL, default=False]

If true sort in reverse order

tabelDataset sortedTable [OUTPUT, MANDATORY, default=no default value]

The sorted table (a new table)

See also

- asciiTableReader
- asciiTableWriter
- Developers Manual: herschel.ia.toolbox.util.SortTableTask

History

- 2012-09-25 - JDS: first release

1.402. sourceExtractorDaophot

Full Name:	herschel.ia.toolbox.srcext.SourceExtractorDaophotTask
Alias:	sourceExtractorDaophot
Type:	Java Task -
Import:	from herschel.ia.toolbox.srcext import SourceExtractorDaophotTask
Category	Astronomical utilities/Source extraction

Description

This task extracts point sources from an HCSS SimpleImage image using DAOPHOT.

The task returns a SourceListProduct listing the sources found in the input image.

The task implements the DAOPHOT (classic) algorithm, following the "FIND" and "APER" procedures in the IDL Astronomy User's Library. The image is smoothed with the DAOPHOT convolution kernel, derived from the point response function, and the resulting smoothed image is searched for peaks, which are taken to be the positions of the point sources. Then the source flux densities are estimated using aperture photometry, as in the aperturePhotometry task.

The DAOPHOT algorithm is described in:

- DAOPHOT: Stetson (1987) *"DAOPHOT - A computer program for crowded-field stellar photometry"*

Note that the image must have units specified, and these must be < mJy, Jy or MJy > / < pixel, sr or beam >. Use `print myImage.getUnit()` to view the image units, and `myImage.setUnit("MJy/sr")` to set the units of the image to MJy/sr (Jy/beam, Jy/pixel, mJy/pixel etc. also work).

If the image does not have a valid WCS, only the pixel coordinates will be calculated, and the input parameters usually in arcsec (e.g., fwhm) will be assumed to be in pixels.

A WCS is valid for the task if it defines square pixels, that is, the difference between the value of cdelt1 and cdelt2 is less than 0.1 %.

Detailed description of the algorithm, with return values in bold

1. The DAOPHOT H filter is computed and applied to the image. The DAOPHOT filter is created first by scaling the PRF such that the maximum value is 1.0, and then by subtracting a constant value from the modified PRF such that the sum is zero. The **filtered image** (optional return value) is then a spatial convolution of the input image and the DAOPHOT filter (with pixels set to NaN if the original image was NaN at that position, or if the pixel in the original image was an "edge" pixel: in a connected region with NaN pixels at the edge of the image). (See details below for how the point response function is determined.)

2. The thresholds are computed. Sigma for the filtered image is estimated using 1.4826 x MAD (median absolute deviation), as a robust estimate of the standard deviation. This makes it possible to convert between the threshold in DAOPHOT H units (the filtered image is known as the H-image), and the threshold in signal-to-noise, one of which will be the detThreshold, depending on whether useSignalToNoise was set (true by default).

3. With the threshold estimated, the filtered image is searched for local maxima (within the region of interest, if specified). A local maximum is a pixel which is higher than all of its neighbours within a pixel distance defined by pixelRegion. Pixels close to the edge of the image are ignored. The value of the threshold image at the local maximum is used to define the **quality** of the source:

if useSignalToNoise is set to False, the value of the filtered image at this point is the **quality**; if useSignalToNoise is set to True, the value of the filtered image is scaled using the value of sigma from the previous step to give the signal-to-noise of the source as the **quality**.

4. The position is refined by fitting a quadratic function to certain pixels in the threshold image. First, using the source pixel and the pixels immediately above and below, and then again using the source pixel and the pixels immediately to the left and right. This gives the source position to a better accuracy than simply the centre of a pixel.

5. **Errors in the position** are estimated as 0.6 * FWHM / signal-to-noise, with signal-to-noise estimated as in the previous step.

6. The sources are filtered on sharpness. The **sharpness** at each source position is defined as "delta" / H-image at that pixel, where "delta" is calculated from the input image, as the mean of the values of the surrounding pixels subtracted from the value of the pixel itself. Any sources with sharpness < sharpnessMin or sharpness > sharpnessMax are removed.

7. The sources are filtered on roundness. At each source location, the input image is convolved with 1-D horizontal and vertical versions of the DAOPHOT H filter to give hX and hY. **Roundness** is computed as (hY-hX)/(hY+hX). Any sources with roundness < roundnessMin or roundness > roundnessMax are removed.

8. For the background estimation (within an annulus), the pixels in the input map are divided in to source pixels and background pixels. Pixels will be marked as "background" if the distance from the nearest source is greater than innerArcsec.

9. For each source, the annulus between innerArcsec and outerArcsec is searched for background pixels. If no background pixels are found, the source is removed (see the log messages). If there are valid background pixels, the **background** for the source is set to the median value, and the **plus and minus errors for the background** are computed using the 95% confidence interval of the values in the annulus. Units of the background are the units of the input image.

10. Aperture photometry is performed to get the source fluxes. The source **flux** is the sum of the pixels in the aperture (defined by radiusArcsec), with the background value subtracted from each pixel. The **flux error** is set to be the flux divided by the signal-to-noise (from above). Units of the flux are mJy.

More information is provided through the links below under "See also", with some SPIRE-specific documentation on source extraction in the SPIRE Data Reduction Guide, Section 5.7.

Examples

Example 1: Extract sources from a SimpleImage using DAOPHOT

```
# myImage is a SimpleImage
disp = Display(myImage)
#
# Extract a list of sources from the image
sourceList = sourceExtractorDaophot( \
    image = myImage,        # Image name \
    detThreshold = 10,      # Threshold in signal-to-noise ratio (or log
 evidence) \
    fwhm = 18.0,            # FWHM of PRF (arcsec) \
    )
#
# How many sources found?
print "Found", len(sourceList), "sources."
#
# Display each source on the image (or drag and drop the sourceList onto the
 image)
disp.addPositionList(sourceList)
#
```

Example 1: Extract sources from a SimpleImage using DAOPHOT

```
# Close display
disp.close()
```

Example 2: Find sources in a rectangular region using DAOPHOT

```
# Define a region of interest (RoI)
roi = SkyMaskRectangle(180, 181, 0, 1) # raMin, raMax, decMin, decMax
# Extract a list of sources from the image
sourceList = sourceExtractorDaophot( \
    image = myImage,        # Image name \
    detThreshold = 10,      # Threshold in signal-to-noise ratio (or log
 evidence) \
    fwhm = 18.0,            # FWHM of PRF (arcsec) \
    roi = roi,              # Region of interest \
    )
```

API Summary

Properties
SimpleImage **image** [INPUT, MANDATORY, default=no default value]
Double **detThreshold** [INPUT, MANDATORY, default=no default value]
Double **fwhm** [INPUT, MANDATORY, default=no default value]
Object **prf** [INPUT, OPTIONAL, default=no default value]
Double **beamArea** [INPUT, OPTIONAL, default=no default value]
Double **pixelRegion** [INPUT, MANDATORY, default=default value: 1.5]
Object **inputSourceList** [INPUT, OPTIONAL, default=no default value]
Boolean **useSignalToNoise** [INPUT, OPTIONAL, default=default value: True]
Boolean **doApertureCorrection** [INPUT, OPTIONAL, default=default value: True]
Double **radiusArcsec** [INPUT, OPTIONAL, default=default value: 1*FWHM]
Double **innerArcsec** [INPUT, OPTIONAL, default=default value: 1.25*FWHM]
Double **outerArcsec** [INPUT, OPTIONAL, default=default value: 3*FWHM]
Double **roundnessMin** [INPUT, OPTIONAL, default=default value: -1.0 (no limit if inputSourceList)]
Double **roundnessMax** [INPUT, OPTIONAL, default=default value: 1.0 (no limit if inputSourceList)]
Double **sharpnessMin** [INPUT, OPTIONAL, default=default value: 0.2 (no limit if inputSourceList)]
Double **sharpnessMax** [INPUT, OPTIONAL, default=default value: 1.0 (no limit if inputSourceList)]
Object **roi** [INPUT, OPTIONAL, default=no default value]
Boolean **getFilteredMap** [INPUT, OPTIONAL, default=default value: False]
Boolean **getPrf** [INPUT, OPTIONAL, default=default value: False]

Properties
SourceListProduct **sourceList** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

SimpleImage image [INPUT, MANDATORY, default=no default value]

Image map to search for point sources

Double detThreshold [INPUT, MANDATORY, default=no default value]

Threshold at which a local maximum is detected. The meaning of this depends on the useSignalToNoise parameter and may be set to either signal-to-noise or DAOPHOT H-units for DAOPHOT extractor.

Double fwhm [INPUT, MANDATORY, default=no default value]

Full-width half maximum, in arcsec, of a default Gaussian point response function (PRF).

Object prf [INPUT, OPTIONAL, default=no default value]

Custom point response function (PRF). Either an Image (SimpleImage) or a Prf (PrfGaussian or PrfImage). If defined, this will be used in preference to the default Gaussian. If a SimpleImage is used, it should have odd-valued length and width, with the peak in the centre, and is interpreted as the response of the central pixels of a 1 Jy source in a map with units of the input map. However, if the SimpleImage has units or a Wcs, these will be taken into account if the image for extraction has different units or resolution. Also, if the image for extraction and the image for the PRF both has a "posAngle" parameter in the meta data, the PRF will be rotated to the same position angle as the image for extraction. If the PRF is a SimpleImage, or if the PRF is not specified (so the default Gaussian is used), a message will be given in the log to give the equivalent PrfImage or PrfGaussian, which will have odd dimensions and width at least 3 x sigma (calculated from FWHM).

Double beamArea [INPUT, OPTIONAL, default=no default value]

Solid angle of the beam in square arcsec. This parameter has two functions. (1) For DAOPHOT only, if the map has units of .../beam, the beamArea will be used to convert to .../pixel units for the aperture photometry. (However, if doApertureCorrection=True, which is the default, the beamArea will make no difference to the fluxes, but only to the aperture correction.) (2) For both DAOPHOT, SUSSEXtractor and sourceExtractorSimultaneous, if no PRF is supplied, and if the image has any units other than .../beam, then the default Gaussian beam will integrate to (Gaussian beam area) / (beamArea), which will typically be less than 1.0 Jy for a 1.0 Jy point source.

Double pixelRegion [INPUT, MANDATORY, default=default value: 1.5]

Radius around each pixel to consider when searching for local maxima. Units: pixels.

Object inputSourceList [INPUT, OPTIONAL, default=no default value]

SourceListProduct containing a list of "known" source positions, at which the fluxes will be extracted. Or [ra, dec], or [[ra1, ra2, ...], [dec1, dec2, ...]], or [[ra1, dec1], [ra2, dec2], ...]. Will fail if ambiguous.

Boolean useSignalToNoise [INPUT, OPTIONAL, default=default value: True]

Threshold to use. If set to False, the threshold will be DAOPHOT H units. This will be returned as the 'quality' in the SourceListDataset.

Boolean doApertureCorrection [INPUT, OPTIONAL, default=default value: True]

If set to True, the PRF will be used to determine the aperture correction, which will be displayed in the log and included in the meta data

Double radiusArcsec [INPUT, OPTIONAL, default=default value: 1*FWHM]

Source aperture radius in arcsec (default value: 1*FWHM)

Double innerArcsec [INPUT, OPTIONAL, default=default value: 1.25*FWHM]

Inner radius of sky aperture annulus in arcsec (default value: 1.25*FWHM)

Double outerArcsec [INPUT, OPTIONAL, default=default value: 3*FWHM]

Outer radius of sky aperture annulus in arcsec (default value: 3*FWHM)

Double roundnessMin [INPUT, OPTIONAL, default=default value: -1.0 (no limit if inputSourceList)]

Minimum value for DAOPHOT roundness filter. Default value: -1.0 (no limit if inputSourceList)

Double roundnessMax [INPUT, OPTIONAL, default=default value: 1.0 (no limit if inputSourceList)]

Maximum value for DAOPHOT roundness filter. Default value: 1.0 (no limit if inputSourceList)

Double sharpnessMin [INPUT, OPTIONAL, default=default value: 0.2 (no limit if inputSourceList)]

Minimum value for DAOPHOT sharpness filter. Default value: 0.2 (no limit if inputSourceList)

Double sharpnessMax [INPUT, OPTIONAL, default=default value: 1.0 (no limit if inputSourceList)]

Maximum value for DAOPHOT sharpness filter. Default value: 1.0 (no limit if inputSourceList)

Object roi [INPUT, OPTIONAL, default=no default value]

Region of interest. Either a SkyMask or a Bool2d (with the same dimensions as the image). Only sources lying within the region of interest will be included in the source extraction. Example: `roi=coverage>10` will restrict extraction to those pixels in which `coverage` has a value greater than 10.

Boolean getFilteredMap [INPUT, OPTIONAL, default=default value: False]

Return the input map filtered by the DAOPHOT kernel. If this is set to True, the return value for the task will be a tuple: (sourceList, [optional return values in alphabetical order])

Boolean getPrf [INPUT, OPTIONAL, default=default value: False]

Return the PRF used by the DAOPHOT extractor. If this is set to True, the return value for the task will be a tuple: (sourceList, [optional return values in alphabetical order])

SourceListProduct sourceList [OUTPUT, MANDATORY, default=no default value]

A SourceListProduct listing the sources found in the input image

Change log: 2013-01-14 AJS checkParameters -> preamble 2013-01-15 AJS [HCSS-17414] Tooltips for *Arcsec and *Min/Max: include details about default values 2013-04-16 AJS [HCSS-17406] Changed default PRF width from 3 x sigma to 9 x sigma for Sussextractor and Simultaneous 2013-04-30 AJS [HCSS-17999] Example for description of roi parameter: roi=coverage>10 2013-04-30 AJS [HCSS-17925] Sonar violations: Line length 2013-05-15 AJS [HCSS-17925] Sonar violations: pass SourceExtractorData in call to getSourceExtractor 2013-05-16 AJS [HCSS-18061] Minor changes due to changes to other classes

See also

- Section 4.19 in *Data Analysis Guide*
- PrfGaussian
- PrfImage
- http://idlastro.gsfc.nasa.gov/contents.html#C2
- http://adsabs.harvard.edu/abs/1987PASP...99..191S
- Developers Manual: herschel.ia.toolbox.srcext.SourceExtractorDaophotTask

1.403. sourceExtractor

Full Name:	herschel.ia.toolbox.srcext.SourceExtractorTask
Alias:	sourceExtractor
Type:	Java Task -
Import:	from herschel.ia.toolbox.srcext import SourceExtractorTask

Description

Extracts point sources from a SimpleImage image.

(A thin wrapper on sourceExtractorSussextractor and sourceExtractorDaophot).

This task should be used via the custom GUI. From the HIPE command line, either sourceExtractor-Sussextractor or sourceExtractorDaophot should be used instead. For details of the parameters in this task, see the documentation for sourceExtractorSussextractor and sourceExtractorDaophot.

See also

- sourceExtractorDaophot
- sourceExtractorSussextractor
- Developers Manual: herschel.ia.toolbox.srcext.SourceExtractorTask

1.404. sourceExtractorSimultaneous

Full Name:	herschel.ia.toolbox.srcext.SourceExtractorSimultaneousTask
Alias:	sourceExtractorSimultaneous
Type:	Java Task -
Import:	from herschel.ia.toolbox.srcext import SourceExtractorSimultaneousTask
Category	Astronomical utilities/Source extraction

Description

This task measures point source photometry of multiple sources simultaneously using linear inversion method.

Typically this task would be used in one of two ways.

1. Blind source extraction is performed on a shorter-wavelength image, and those positions are then used as the inputSourceList for this task.

2. Ancilliary data are used (e.g., from mid-infrared or radio observations) and those positions are used as the inputSourceList in this task.

The data (i.e., the image) are considered to originate entirely from a set of sources of known positions, plus Gaussian random noise. Representing the data (image) as a vector, **d**, the flux densities of the sources as a vector, **f**, the pointing matrix, projecting the sources onto the image, as a matrix, **A**, and the noise as a vector, δ, the model is:

$\mathbf{d} = \mathbf{A}\mathbf{f} + \delta$.

The maximum likelihood value for the flux densities is given by

$\mathbf{f} = (\mathbf{A}^T\mathbf{N}_d^{-1}\mathbf{A})^{-1}\mathbf{A}^T\mathbf{N}_d^{-1}\mathbf{d}$

where $\mathbf{N}_d$ is the covariance matrix of the data, assumed to be a diagonal matrix derived from the error map.

The task calculates this matrix equation for the flux densities of the sources in the inputSourceList that lie within the bounds of the image.

If fitBackground is set to True (not the default), then an additional "source" is included in that matrix equation, but the corresponding elements of the pointing matrix project that "source" equally onto all pixels of the image, rather than onto a particular location. This makes it possible for the task to estimate the value of the background, which is assigned to each of the sources.

Uncertainties in the flux densities are given by the covariance matrix of flux densities, $\mathbf{N}_f$, which is estimated by $(\mathbf{A}^T\mathbf{N}_d^{-1}\mathbf{A})^{-1}$. (Strictly, this is a lower limit to the covariance matrix, but it is computationally very challenging to improve on this estimate.) The values given as errors in the flux density of the sources are the diagonal elements of this matrix.

Note that the task relies on being able to store a double-precision matrix of size N_{sources} x N_{sources} in memory, so if the number of sources is very large (more than 10,000, approximately), it may be necessary to process different regions of the image separately.

More information is provided through the links below under "See also", with some SPIRE-specific documentation on source extraction in the SPIRE Data Reduction Guide.

Example

Example 1: Measure photometry of sources simultaneously

```
sourceList = sourceExtractorSimultaneous( \
    image = myImage,           # Image name \
    inputSourceList = sources, # Known source positions \
    fwhm = 18.0                # FWHM of PRF (arcsec) \
    )
```

API Summary

Properties
SimpleImage **image** [INPUT, MANDATORY, default=no default value]
Object **inputSourceList** [INPUT, MANDATORY, default=no default value]
Double **fwhm** [INPUT, MANDATORY, default=no default value]
Object **prf** [INPUT, OPTIONAL, default=no default value]
Double **beamArea** [INPUT, OPTIONAL, default=no default value]
Boolean **fitBackground** [INPUT, OPTIONAL, default=default value: True]
Object **roi** [INPUT, OPTIONAL, default=no default value]
Boolean **getPrf** [INPUT, OPTIONAL, default=default value: False]
SourceListProduct **sourceList** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

SimpleImage image [INPUT, MANDATORY, default=no default value]

Image map to search for point sources

Object inputSourceList [INPUT, MANDATORY, default=no default value]

SourceListProduct containing a list of "known" source positions, at which the fluxes will be extracted. Or [ra, dec], or [[ra1, ra2, ...], [dec1, dec2, ...]], or [[ra1, dec1], [ra2, dec2], ...]. Will fail if ambiguous.

Double fwhm [INPUT, MANDATORY, default=no default value]

Full-width half maximum, in arcsec, of a default Gaussian point response function (PRF).

Object prf [INPUT, OPTIONAL, default=no default value]

Custom point response function (PRF). Either an Image (SimpleImage) or a Prf (PrfGaussian or PrfImage). If defined, this will be used in preference to the default Gaussian. If a SimpleImage is used, it should have odd-valued length and width, with the peak in the centre, and is interpreted as the response of the central pixels of a 1 Jy source in a map with units of the input map. However, if the SimpleImage has units or a Wcs, these will be taken into account if the image for extraction has different units or resolution. Also, if the image for extraction and the image for the PRF both has a "posAngle" parameter in the meta data, the PRF will be rotated to the same position angle as the image for extraction. If the PRF is a SimpleImage, or if the PRF

Object prf [INPUT, OPTIONAL, default=no default value]

is not specified (so the default Gaussian is used), a message will be given in the log to give the equivalent PrfImage or PrfGaussian, which will have odd dimensions and width at least 9 x sigma (calculated from FWHM).

Double beamArea [INPUT, OPTIONAL, default=no default value]

Solid angle of the beam in square arcsec. This parameter has two functions. (1) For DAOPHOT only, if the map has units of .../beam, the beamArea will be used to convert to .../pixel units for the aperture photometry. (However, if doApertureCorrection=True, which is the default, the beamArea will make no difference to the fluxes, but only to the aperture correction.) (2) For both DAOPHOT, SUSSEXtractor and sourceExtractorSimultaneous, if no PRF is supplied, and if the image has any units other than .../beam, then the default Gaussian beam will integrate to (Gaussian beam area) / (beamArea), which will typically be less than 1.0 Jy for a 1.0 Jy point source.

Boolean fitBackground [INPUT, OPTIONAL, default=default value: True]

Flag to fit background as a free parameter

Object roi [INPUT, OPTIONAL, default=no default value]

Region of interest. Either a SkyMask or a Bool2d (with the same dimensions as the image). Only sources lying within the region of interest will be included in the source extraction. Example: `roi=coverage>10` will restrict extraction to those pixels in which `coverage` has a value greater than 10.

Boolean getPrf [INPUT, OPTIONAL, default=default value: False]

Return the PRF used by the extractor. If this is set to True, the return value for the task will be a tuple: (sourceList, [optional return values in alphabetical order])

SourceListProduct sourceList [OUTPUT, MANDATORY, default=no default value]

A SourceListProduct listing the sources found in the input image

Change log: 2013-01-14 AJS checkParameters -> preamble 2013-01-14 AJS [HCSS-17497] Added roi parameter 2013-04-16 AJS [HCSS-17406] Changed default PRF width from 3 x sigma to 9 x sigma for Sussextractor and Simultaneous 2013-04-30 AJS [HCSS-17999] Example for description of roi parameter: roi=coverage>10 2013-04-30 AJS [HCSS-17925] Sonar violations: Line length 2013-05-16 AJS [HCSS-18061] Moved adding of optional parameter switch task parameters & inputSourceList to superclass

See also

- Section 4.19 in *Data Analysis Guide*
- PrfGaussian
- PrfImage
- http://adsabs.harvard.edu/abs/2010MNRAS.409...48R
- Developers Manual: herschel.ia.toolbox.srcext.SourceExtractorSimultaneousTask

1.405. sourceExtractorSussextractor

Full Name:	herschel.ia.toolbox.srcext.SourceExtractorSussextractorTask
Alias:	sourceExtractorSussextractor
Type:	Java Task -
Import:	from herschel.ia.toolbox.srcext import SourceExtractorSussextractorTask
Category	Astronomical utilities/Source extraction

Description

This task extracts point sources from an HCSS SimpleImage image using SUSSEXtractor.

The task returns a SourceListProduct listing the sources found in the input image

The task implements the SUSSEXtractor algorithm, as described by Savage & Oliver (2007), ApJ, 661, 1339 and in more detail here (PDF) or here (LaTeX). First the image is smoothed with the PRF and searched for peaks, which are taken to be at the positions of the point sources. The optional returned **filtered image** is this smoothed image. The flux densities (and background) of these sources are estimated by a similar but more complex procedure, as described below.

Note that the image must have units specified, and these must be < mJy, Jy or MJy > / < pixel, sr or beam >. Use `print myImage.getUnit()` to view the image units, and `myImage.setUnit("MJy/sr")` to set the units of the image to MJy/sr (Jy/beam, Jy/pixel, mJy/pixel etc. also work).

If the image does not have a valid WCS, only the pixel coordinates will be calculated, and the input parameters usually in arcsec (e.g., fwhm) will be assumed to be in pixels.

Detailed description of the algorithm, with return values in bold

1. A PSF-filtered image is created, smoothing with the PSF. This is the same as the calculations for gamma / alpha in the detailed description, except that the weight in each pixel is assumed to be 1.0. (See details below for how the point response function is determined.)

2. This PSF-filtered is searched for local maxima (within the region of interest, if specified). A local maximum is a pixel which is higher than all of its neighbours within a pixel distance defined by pixelRegion. Pixels close to the edge of the image are ignored.

3. The position is refined by fitting a quadratic function to certain pixels in the PSF-filtered image. First, using the source pixel and the pixels immediately above and below, and then again using the source pixel and the pixels immediately to the left and right. This gives the source position to a better accuracy than simply the centre of a pixel.

4. Using the PRF, quantities are calculated at each source location pixel of the image: alpha, beta, gamma, delta and epsilon (see Savage & Oliver, 2007, equations 9 and 10). If background fitting is not being performed, then beta, delta and epsilon are not needed. Pixels are skipped if it is an "edge" pixel: in a connected region with NaN pixels at the edge of the image. Other NaN pixels are not skipped, but NaN values are ignored in the summation over the pixels in the PRF.

5. After the parameters have been calculated (see the attached PDF file), the threshold image is selected for detecting the sources.

   - If useSignalToNoise is true, the threshold image is the flux estimate divided by the flux error, at each pixel.

   - If useSignalToNoise is false (default), the threshold image is the difference in log evidence between the source model and the background model: the log(Bayes factor).

6. With the threshold calculated at each source position, only those sources with a threshold above the specified detection threshold are accepted as detections. The value of the threshold at the source position is the **quality** of the source.

7. **Errors in the position** are estimated as 0.6 * FWHM / signal-to-noise, up to a maximum of 1.0 pixels.

8. If priors in flux or background-with-source have been imposed, then the corresponding plus- or minus-errors in those quantities are set to zero. For example, if the flux has been increased to the prior minimum value, then fluxMinusError is set to be zero.

More information is provided through the links below under "See also", with some SPIRE-specific documentation on source extraction in the SPIRE Data Reduction Guide, Section 5.7.

Examples

Example 1: Extract sources from a SimpleImage using SUSSEXtractor

```
# myImage is a SimpleImage
disp = Display(myImage)
#
# Extract a list of sources from the image
sourceList = sourceExtractorSussextractor( \
    image = myImage,        # Image name \
    detThreshold = 10,      # Threshold in signal-to-noise ratio (or log
 evidence) \
    fwhm = 18.0,            # FWHM of PRF (arcsec) \
    )
#
# How many sources found?
print "Found", len(sourceList), "sources."
#
# Display each source on the image (or drag and drop the sourceList onto the
 image)
disp.addPositionList(sourceList)
#
# Close display
disp.close()
```

Example 2: Find sources in a rectangular region using SUSSEXtractor

```
# Define a region of interest (RoI)
roi = SkyMaskRectangle(180, 181, 0, 1) # raMin, raMax, decMin, decMax
# Extract a list of sources from the image
sourceList = sourceExtractorSussextractor( \
    image = myImage,        # Image name \
    detThreshold = 10,      # Threshold in signal-to-noise ratio (or log
 evidence) \
    fwhm = 18.0,            # FWHM of PRF (arcsec) \
    roi = roi,              # Region of interest \
    )
```

API Summary

Properties
SimpleImage **image** [INPUT, MANDATORY, default=no default value]
Double **detThreshold** [INPUT, MANDATORY, default=no default value]
Double **fwhm** [INPUT, MANDATORY, default=no default value]
Object **prf** [INPUT, OPTIONAL, default=no default value]
Double **beamArea** [INPUT, OPTIONAL, default=no default value]

Properties
Double **fluxPriorsLambda** [INPUT, OPTIONAL, default=default value: 0.0]
Boolean **fitBackground** [INPUT, OPTIONAL, default=default value: True]
Double **pixelRegion** [INPUT, MANDATORY, default=default value: 1.5]
Object **inputSourceList** [INPUT, OPTIONAL, default=no default value]
Boolean **useSignalToNoise** [INPUT, OPTIONAL, default=default value: False]
Double **fluxPriorsMin** [INPUT, OPTIONAL, default=default value: 1.0e-4]
Double **fluxPriorsMax** [INPUT, OPTIONAL, default=default value: 1.0e8]
Double **backgroundPriorsMin** [INPUT, OPTIONAL, default=no default value]
Double **backgroundPriorsMax** [INPUT, OPTIONAL, default=no default value]
Object **roi** [INPUT, OPTIONAL, default=no default value]
Boolean **getFilteredMap** [INPUT, OPTIONAL, default=default value: False]
Boolean **getPrf** [INPUT, OPTIONAL, default=default value: False]
SourceListProduct **sourceList** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

SimpleImage image [INPUT, MANDATORY, default=no default value]

Image map to search for point sources

Double detThreshold [INPUT, MANDATORY, default=no default value]

Threshold at which a local maximum is detected. The meaning of this depends on the useSignalToNoise parameter and may be set to either signal-to-noise or log(Bayes factor). For log(Bayes factor), 1.0 is weak evidence for a source, 2.5 is moderate and 5.0 is strong.

Double fwhm [INPUT, MANDATORY, default=no default value]

Full-width half maximum, in arcsec, of a default Gaussian point response function (PRF).

Object prf [INPUT, OPTIONAL, default=no default value]

Custom point response function (PRF). Either an Image (SimpleImage) or a Prf (PrfGaussian or PrfImage). If defined, this will be used in preference to the default Gaussian. If a SimpleImage is used, it should have odd-valued length and width, with the peak in the centre, and is interpreted as the response of the central pixels of a 1 Jy source in a map with units of the input map. However, if the SimpleImage has units or a Wcs, these will be taken into account if the image for extraction has different units or resolution. Also, if the image for extraction and the image for the PRF both has a "posAngle" parameter in the meta data, the PRF will be rotated to the same position angle as the image for extraction. If the PRF is a SimpleImage, or if the PRF

Object prf [INPUT, OPTIONAL, default=no default value]

is not specified (so the default Gaussian is used), a message will be given in the log to give the equivalent PrfImage or PrfGaussian, which will have odd dimensions and width at least 9 x sigma (calculated from FWHM).

Double beamArea [INPUT, OPTIONAL, default=no default value]

Solid angle of the beam in square arcsec. This parameter has two functions. (1) For DAOPHOT only, if the map has units of .../beam, the beamArea will be used to convert to .../pixel units for the aperture photometry. (However, if doApertureCorrection=True, which is the default, the beamArea will make no difference to the fluxes, but only to the aperture correction.) (2) For both DAOPHOT, SUSSEXtractor and sourceExtractorSimultaneous, if no PRF is supplied, and if the image has any units other than .../beam, then the default Gaussian beam will integrate to (Gaussian beam area) / (beamArea), which will typically be less than 1.0 Jy for a 1.0 Jy point source.

Double fluxPriorsLambda [INPUT, OPTIONAL, default=default value: 0.0]

Lambda value for SUSSEXtractor flux priors: 0 = tophat, 1 = Jeffreys, >1 = deboost

Boolean fitBackground [INPUT, OPTIONAL, default=default value: True]

SUSSEXtractor flag to fit background as a free parameter

Double pixelRegion [INPUT, MANDATORY, default=default value: 1.5]

Radius around each pixel to consider when searching for local maxima. Units: pixels.

Object inputSourceList [INPUT, OPTIONAL, default=no default value]

SourceListProduct containing a list of "known" source positions, at which the fluxes will be extracted. Or [ra, dec], or [[ra1, ra2, ...], [dec1, dec2, ...]], or [[ra1, dec1], [ra2, dec2], ...]. Will fail if ambiguous.

Boolean useSignalToNoise [INPUT, OPTIONAL, default=default value: False]

Threshold to use. If set to False, the threshold will be log(Bayes factor). This will be returned as the 'quality' in the SourceListDataset. For log(Bayes factor), 1.0 is weak evidence for a source, 2.5 is moderate and 5.0 is strong

Double fluxPriorsMin [INPUT, OPTIONAL, default=default value: 1.0e-4]

Prior on minimum flux for SUSSEXtractor, in mJy

Double fluxPriorsMax [INPUT, OPTIONAL, default=default value: 1.0e8]

Prior on maximum flux for SUSSEXtractor, in mJy

Double backgroundPriorsMin [INPUT, OPTIONAL, default=no default value]

Prior on minimum background level for SUSSEXtractor, in units of the input map (only used if fitBackground is True). Default is the minimum value of the map, minus (max-min) of map.

Double backgroundPriorsMax [INPUT, OPTIONAL, default=no default value]

Prior on maximum background level for SUSSEXtractor, in units of the input map (only used if fitBackground is True). Default is the maximum value of the map, plus (max-min) of map.

Object roi [INPUT, OPTIONAL, default=no default value]

Region of interest. Either a SkyMask or a Bool2d (with the same dimensions as the image). Only sources lying within the region of interest will be included in the source extraction. Example: `roi=coverage>10` will restrict extraction to those pixels in which `coverage` has a value greater than 10.

Boolean getFilteredMap [INPUT, OPTIONAL, default=default value: False]

Return the input map filtered by SUSSEXtractor, in units of mJy. If this is set to True, the return value for the task will be a tuple: (sourceList, [optional return values in alphabetical order]). The filtered map is the source flux estimate, assuming each pixel is weighted as 1.0.

Boolean getPrf [INPUT, OPTIONAL, default=default value: False]

Return the PRF used by the SUSSEXtractor extractor. If this is set to True, the return value for the task will be a tuple: (sourceList, [optional return values in alphabetical order])

SourceListProduct sourceList [OUTPUT, MANDATORY, default=no default value]

A SourceListProduct listing the sources found in the input image

Change log: 2013-01-14 AJS checkParameters -> preamble 2013-04-16 AJS [HCSS-17406] Changed default PRF width from 3 x sigma to 9 x sigma for Sussextractor and Simultaneous 2013-04-30 AJS [HCSS-17999] Example for description of roi parameter: roi=coverage>10 2013-04-30 AJS [HCSS-17925] Sonar violations: Line length 2013-05-15 AJS [HCSS-17925] Sonar violations: pass SourceExtractorData in call to getSourceExtractor 2013-05-16 AJS [HCSS-18061] Minor changes due to changes to other classes

See also

- Section 4.19 in *Data Analysis Guide*
- PrfGaussian
- PrfImage
- http://adsabs.harvard.edu/abs/2007ApJ...661.1339S
- Developers Manual: herschel.ia.toolbox.srcext.SourceExtractorSussextractorTask

1.406. sourceFit

Full Name:	herschel.ia.toolbox.fit.SourceFitTask
Alias:	sourceFit
Type:	Java Task -
Import:	from herschel.ia.toolbox.fit import SourceFitTask
Category	Mathematics/Fitting

Description

Task to easily fit Source models (Gauss, Lorentz, Voigt etcetera).

Task shell around the package herschel.ia.numeric.toolbox.fit. Both a command line and a GUI interface are provided. With the GUI interface there is more of the Fitter package accessable than on this tasks command line. However not everything possible within the fitter package is accessable via this task, GUI or otherwise. Most text fields in the GUI accept either numerics or HIPE variables or HIPE expressions.

Example

Example 1: SourceFitTask

```
# Assume that `tt' and/or `data' are Double1d's
# usage on command line:
tt = Double1d.range(11) - 5
data = Double1d(11)
data[5] = 1.0
data[6] = 4.0
data[7] = 2.0
# one-liner using default GaussModel and LevenbergMarquardtFitter
par1 = sourceFit( x=tt, y=data )
print par1
# more extended
ft = sourceFit
ft.x = tt
ft.y = data
ft.autoinit = "high"           # put initial par values at highest point
ft.modelname = "LorentzModel"
ft.fittername = "AmoebaFitter"
par2 = ft()              # performs the execute method
print par2             # parameters of the fit
print ft.stdev          # standard deviations
print ft.fitter          # name of the fitter
# Etcetera, etc.
# use the GUI.
ft = sourceFit
ft.gui = 1         # start the GUI
# from the GUI select the data, model, fitter, properties etc and run.
# Etcetera as before.
```

API Summary

Jython Syntax

```
see example below
```

Properties

NumericData **x** [INPUT, OPTIONAL, default=null]

Properties
DoubleArray **y** [INPUT, MANDATORY, default=no]
Boolean **indexview** [INPUT, OPTIONAL, default=true]
DoubleArray **weights** [INPUT, OPTIONAL, default=null]
AbstractModel **model** [INOUT, OPTIONAL, default=null]
String **modelname** [INPUT, OPTIONAL, default=no]
Array1dData **modelarg** [INPUT, OPTIONAL, default=null]
Fitter **fitter** [INOUT, OPTIONAL, default=null]
String **fittername** [INPUT, OPTIONAL, default=no]
String **autoinit** [INPUT, OPTIONAL, default=""]
Integer **smooth** [INPUT, OPTIONAL, default=1]
Double1d **result** [OUTPUT, OPTIONAL, default=n/a]
Double1d **parameters** [OUTPUT, OPTIONAL, default=n/a]
Double **chisq** [OUTPUT, OPTIONAL, default=n/a]
Double1d **prior** [INPUT, OPTIONAL, default=null]
Double **scale** [OUTPUT, OPTIONAL, default=n/a]
IterationPlotable **plotter** [INPUT, OPTIONAL, default=null]
Integer **plotfreq** [INPUT, OPTIONAL, default=10]
Integer **printfreq** [INPUT, OPTIONAL, default=0]
Boolean **auto** [INPUT, OPTIONAL, default=false]
Double **fixed** [INPUT, OPTIONAL, default=1.0]
Double **mixed** [INPUT, OPTIONAL, default=0.0]
Double **tolerance** [INPUT, OPTIONAL, default=0.01]
Integer **iterations** [INPUT, OPTIONAL, default=10000]
Double **temperature** [INPUT, OPTIONAL, default=0.0]
Double **cooling** [INPUT, OPTIONAL, default=0.95]
Integer **tempsteps** [INPUT, OPTIONAL, default=100]
Double1d **initialpars** [INPUT, OPTIONAL, default=from model]
Double1d **highlimits** [INPUT, OPTIONAL, default=null]
Double1d **lowlimits** [INPUT, OPTIONAL, default=null]
Int1d **keepfixed** [INPUT, OPTIONAL, default=null]
Double1d **fixedvalues** [INPUT, OPTIONAL, default=null]
Boolean **gui** [INPUT, OPTIONAL, default=false]

Limitations

Not everything which is possible using the package directly is accessable via this task.

Miscellaneous

In case of problems look at the trouble shooting section.

API details

Properties

NumericData x [INPUT, OPTIONAL, default=null]

The independent variable(s) of the fit problem. If not provided, it will use indices ranging from 0 to len(y). If y is more-dimensional the proper 2-d indices are provided.

DoubleArray y [INPUT, MANDATORY, default=no]

The data to be fitted. It can be more-dimensional. E.g. a 2-dimensional map.

Boolean indexview [INPUT, OPTIONAL, default=true]

Applicable parameters appear in the same order as in the y DoubleArray This parameter is only active when y has 2 dimensions or more.

DoubleArray weights [INPUT, OPTIONAL, default=null]

The weights to be used in the fit. Can be more-dimensional. When provided it should be the same size as parameter y.

AbstractModel model [INOUT, OPTIONAL, default=null]

The model to be fitted. Default: GaussModel (for 1d data) or Gauss2DModel (for 2d data)

String modelname [INPUT, OPTIONAL, default=no]

The model to be fitted. Default: GaussModel (for 1d data) or Gauss2DModel (for 2d data)

Array1dData modelarg [INPUT, OPTIONAL, default=null]

Arguments, if any, needed in the Constructor of the model.

Fitter fitter [INOUT, OPTIONAL, default=null]

Fitter to be used. Default: LevenbergMarquardtFitter

String fittername [INPUT, OPTIONAL, default=no]

Fitter to be used. Default: LevenbergMarquardtFitter

String autoinit [INPUT, OPTIONAL, default=""]

Automated search for initial params. options: "high", "low", "center"

Integer smooth [INPUT, OPTIONAL, default=1]

Smooth data with BoxCarFilter before auto-search. smooth > 1.

Double1d result [OUTPUT, OPTIONAL, default=n/a]

The parameters of the fit.

Double1d parameters [OUTPUT, OPTIONAL, default=n/a]

The parameters of the fit. (same as result)

Double chisq [OUTPUT, OPTIONAL, default=n/a]

Chi-squared of the fit.

Double1d prior [INPUT, OPTIONAL, default=null]

Prior ranges for the parameters, needed when the evidence is requested.

Double scale [OUTPUT, OPTIONAL, default=n/a]

Noise scale of the data (when auto or mixed is selected)

IterationPlotable plotter [INPUT, OPTIONAL, default=null]

make a plot of the fit at each "plotfreq" iteration. A plotter is provided at herschel.ia.toolbox.fit.IterationPlotter

Integer plotfreq [INPUT, OPTIONAL, default=10]

to be used in conjunction with plotter.

Integer printfreq [INPUT, OPTIONAL, default=0]

Report about the present parameter settings every printfreq-th iteration.

Boolean auto [INPUT, OPTIONAL, default=false]

Select automatic noise scaling.

Double fixed [INPUT, OPTIONAL, default=1.0]

Fixed noise scale.

Double mixed [INPUT, OPTIONAL, default=0.0]

Automatic noise scaling with a minimum.

Double tolerance [INPUT, OPTIONAL, default=0.01]

Stopping criterion for iterative fitting.

Integer iterations [INPUT, OPTIONAL, default=10000]

Maximum number of iterations.

Double temperature [INPUT, OPTIONAL, default=0.0]

Starting temperature in annealing amoeba fitting.

Double cooling [INPUT, OPTIONAL, default=0.95]

Cooling step in annealing amoeba fitting.

Integer tempsteps [INPUT, OPTIONAL, default=100]

Number of exploratory steps at each temperature.

Double1d initialpars [INPUT, OPTIONAL, default=from model]

Initial parameters in iterative fitters.

Double1d highlimits [INPUT, OPTIONAL, default=null]

Upper limits of the parameters (only in AmoebaFitter)

Double1d lowlimits [INPUT, OPTIONAL, default=null]

Lower limits of the parameters (only in AmoebaFitter)

Int1d keepfixed [INPUT, OPTIONAL, default=null]

Keep these parameters fixed. (Parameter numbering starts at 0)

Double1d fixedvalues [INPUT, OPTIONAL, default=null]

Values at which the parameters should be kept.

Boolean gui [INPUT, OPTIONAL, default=false]

Allows to handle this task using a gui.

See also

- Developers Manual: herschel.ia.toolbox.fit.SourceFitTask

History

- 2006-10-15 - DK: Creation

1.407. sourceFitting

Full Name:	herschel.ia.toolbox.image.SourceFittingTask
Alias:	sourceFitting
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import SourceFittingTask
Category	Astronomical utilities/Source extraction

Description

This is a task to fit a two-dimensional gaussian to a source in a

specified rectangular region on an image. The user can decide whether to use an elongated or a circular gaussian, and whether or not there is a slope in the background. This is a task to fit a two-dimensional gaussian to a source in a specified rectangular region on an image. The user can decide whether to use an elongated or a circular gaussian, and whether or not there is a slope in the background. For circular sources :

- model : Gauss2DModel + PolySurfaceModel(degree = 0)
- initial parameters :
  - the maximum value in the cropped region yields an inital value for the center of the gaussian and its peak value
  - the background is estimated via the Daophot algorithm (as for aperture photometry) on the whole rectangle
- fitter : LevenbergMarquardtFitter

For elongated sources :

- model : Gauss2DRotModel + PolySurfaceModel(degree = 0)
- initial parameters :
  - the fit that would be done if the source was circular yields an initial value for the center and the peak value of the gaussian and for the background
  - the sigma's and the rotation angle of the gaussian are estimated via profile plotting : for all straight lines through the estimated center with an angle between 0 and 180 degrees w.r.t. the x-axis (in steps of 15 degrees), a gaussian combined with a constant function is fitted (deriving the initial parameters from the circular 2D-fit) and the largest sigma is taken to be the initial value for the width of the gaussian in the x-direction and the correponding angle as initial value for the rotation angle of the gaussian; the sigma in the y-direction is estimated via the fit of the line perpendicular to the previous one.
- fitter : LevenBergMarquardtFitter

In order to improve the convergence of the fitter, upscaling of the data has been provided. If the intensity peak in the cropped image is below 1000.0, the image is multiply such that the new peak is at 1000.0.

Example

Example 1: This is an example of how to use the sourceFitting :

```
parameters = sourceFitting(elongated = True, slope =
```

Example 1: This is an example of how to use the sourceFitting :

```
False,image = myImage, minX = 74.19, minY = 116.65,
width=29.67, height = 26.60)
```

API Summary

Properties
Image **image** [INPUT, MANDATORY, default=None This is input image.]
Double **minX** [INPUT, MANDATORY, default=NaN This is the]
Double **minY** [INPUT, OPTIONAL, default=NaN This is the y-pixel-co-ordinate]
Double **width** [INPUT, OPTIONAL, default=0.0 This is the width of the]
Double **height** [INPUT, OPTIONAL, default=0.0 This is the height of the]
Boolean **elongated** [INPUT, OPTIONAL, default=true This indicates whether]
Boolean **slope** [INPUT, OPTIONAL, default=0.0 This indicates whether the]
SourceFittingProduct **parameters** [OUTPUT, MANDATORY, default=None These]

API details

Properties

Image image [INPUT, MANDATORY, default=None This is input image.]

Double minX [INPUT, MANDATORY, default=NaN This is the]

x-pixel-coordinate of the reference corner of the rectangle. The reference corner of the rectangle is defined as the corner with the smallest x- and y-pixel-coordinate.

Double minY [INPUT, OPTIONAL, default=NaN This is the y-pixel-co-ordinate]

of the reference corner of the rectangle. The reference corner of the rectangle is defined as the corner with the smallest x- and y-pixel-coordinate.

Double width [INPUT, OPTIONAL, default=0.0 This is the width of the]

rectangle in pixels.

Double height [INPUT, OPTIONAL, default=0.0 This is the height of the]

rectangle in pixels.

Boolean elongated [INPUT, OPTIONAL, default=true This indicates whether]

the source is elongated (or circular).

Boolean slope [INPUT, OPTIONAL, default=0.0 This indicates whether the]
bakcground has a slope (or is constant).

SourceFittingProduct parameters [OUTPUT, MANDATORY, default=None These]
are the fitting parameters.

See also

- Developers Manual: herschel.ia.toolbox.image.SourceFittingTask

1.408. SourceFittingProduct

Full Name:	herschel.ia.dataset.image.SourceFittingProduct
Type:	Java Class -
Import:	from herschel.ia.dataset.image import SourceFittingProduct
Category	Astronomical utilities/Photometry

Description

A class to handle the result of the SourceFittingTask.

API Summary

Constructors
SourceFittingProduct (boolean Indicates whether or not the 2D isElongated, boolean Indicates whether the background hasSlope, Wcs Wcs of the image for which the fit was made wcs, Unit The unit of the image for which the fit was unit) Construction of a new SourceFittingProduct. Construction of a new
SourceFittingProduct (double The peak intensity peak, double The x-pixel-coordinate of the center centerX, double The y-pixel-coordinate of the center centerY, double The width in pixels sigmaPixels, double The background background, Wcs The Wcs of the image for which the fit was wcs, Unit The unit of the image for which the fit was unit) Construction of a new SourceFittingProduct. Construction of a new
SourceFittingProduct (double The peak intensity peak, double The x-pixel-coordinate of the center centerX, double The y-pixel-coordinate of the center centerY, double The sigmaXPixels sigmaXPixels, double The sigmaYPixels sigmaYPixels, double The rotationAngle rotationAngle, double The background background, Wcs The Wcs of the image for which the fit was wcs, Unit The unit of the image for which the fit was unit) Construction of a new SourceFittingProduct. Construction of a new

Methods
Image **evaluate** (Image The image image) Evaluation of this SourceFittingProduct. Evaluates this
double **getPeak** Returns the peak intensity for this SourceFittingProduct. Returns
double **getPeakError** Returns the error on the peak intensity for this
double **getCenterX** Returns the x-pixel-coordinate for the center for this
double **getCenterXError** Returns the error on the x-pixel-coordinate for the center for
double **getCenterY** Returns the y-pixel-coordinate for the center for this

Methods
double **getCenterYError** Returns the error on the y-pixel-coordinate for the center for
double **getCenterRA** Returns the right ascension for the center for this
double **getCenterDec** Returns the declination for the center for this
double **getSigmaPixels** Returns the width for this SourceFittingProduct in pixels. Returns
double **getSigmaPixelsError** Returns the error on the width for this SourceFittingProduct in
double **getSigmaXPixels** Returns the width in the x-direction for this SourceFittingProduct
double **getSigmaXPixelsError** Returns the error on width in the x-direction for this
double **getSigmaYPixels** Returns the width in the y-direction for this SourceFittingProduct
double **getSigmaYPixelsError** Returns the error on width in the y-direction for this
double **getSigmaArcsec** Returns the width for this SourceFittingProduct in arcsec. Returns
double **getSigmaArcsecError** Returns the error on the width for this SourceFittingProduct in
double **getSigmaXArcsec** Returns the width in the x-direction for this SourceFittingProduct
double **getSigmaXArcsecError** Returns the error on width in the x-direction for this
double **getSigmaYArcsec** Returns the width in the y-direction for this SourceFittingProduct
double **getSigmaYArcsecError** Returns the error on width in the y-direction for this
double **getRotationAngle** Returns the rotation angle for this SourceFittingProduct in
double **getRotationAngleError** Returns the error on the rotation angle for this
double **getRotationAngleDegrees** Returns the rotation angle for this SourceFittingProduct in
double **getRotationAngleDegreesError** Returns the error on the rotation angle for this
double **getRotationAngleRadians** Returns the rotation angle for this SourceFittingProduct in
double **getRotationAngleRadiansError**

Methods
Returns the error on the rotation angle for this
double **getPositionAngleDegrees** Returns the position angle for this SourceFittingProduct in
double **getPositionAngleDegreesError** Returns the error on the position angle for this
double **getPositionAngleRadians** Returns the position angle for this SourceFittingProduct in
double **getPositionAngleRadiansError** Returns the error on the position angle for this
double **getBackground** Returns the background intensity for this SourceFittingProduct.
double **getBackgroundError** Returns the error on the background intensity for this
double **getSlopeX** Returns the slope of the background along the x-axis for this
double **getSlopeXError** Returns the error on the slope of the background along the x-axis
double **getSlopeY** Returns the slope of the background along the y-axis for this
double **getSlopeYError** Returns the error on the slope of the background along the y-axis
double **getPixel2Arcsec** Converts the given length in pixels to a length in arcsec.
Wcs **getWcs** Returns the Wcs of the image for which the fit was made. Returns
String **getUnit** Returns the unit of the image for which the fit was made. Returns
boolean **hasErrors** Checks whether errors are available. Checks whether errors are
setPeak (double Peak value value, double Error on the peak value error) Setting the peak intensity. Sets the peak intensity and the
setCenter (double The x-pixel-coordinate of the centerXValue, double Error on the x-pixel-coordinate centerXError, double The y-pixel-coordinate of the centerYValue, double Error on the y-pixel-coordinate centerYError) Setting the centroid of the fitted 2D gaussian. Sets the centroid
setSigma (double Standard deviation of the symmetric 2D value, double Error on the standard deviation of the error) Setting the standard deviation of the fitted 2D gaussian. Sets the
setSigma (double Value of the standard deviation in sigmaXValue, doule Error on the standard deviation in sigmaXError, double Value of the standard deviation in sigmaYValue, double Error on the standard deviation in sigmaYError)

Methods
Setting the standard deviation in the x- and y-direction of the
setRotationAngle (double Rotation angle of the asymmetric 2D value, double Error on the rotation angle of the error)
Setting the rotation angle of the fitted 2D gaussian. Sets the
setBackground (double Value of the constant background value, double Error on the constant background error)
Setting the background. Sets the (constant) background and the
setBackground (double Value of the background backgroundValue, doule Error on the background backgroundError, double slopeXValue, double Error on the slope of the slopeXError, double slopeYValue, double Error on the slope of the slopeYError)
Setting the background. Sets the value of the background with a

API Details

Constructors

SourceFittingProduct (boolean Indicates whether or not the 2D isElongated, boolean Indicates whether the background hasSlope, Wcs Wcs of the image for which the fit was made wcs, Unit The unit of the image for which the fit was unit)

Construction of a new SourceFittingProduct. Construction of a new

SourceFittingProduct.

Arguments

`boolean Indicates whether or not the 2D` **`isElongated`** `[INPUT, MANDATORY, default=no default value]`

gaussian is elongated (i.e. asymmetric) or not.

`boolean Indicates whether the background` **`hasSlope`** `[INPUT, MANDATORY, default=no default value]`

shows a linear slope (in the x- and y-direction) or is constant.

`Wcs Wcs of the image for which the fit was made` **`wcs`** `[INPUT, MANDATORY, default=no default value]`

`Unit The unit of the image for which the fit was` **`unit`** `[INPUT, MANDATORY, default=no default value]`

made, as Unit

SourceFittingProduct (double The peak intensity peak, double The x-pixel-coordinate of the center centerX, double The y-pixel-coordinate of the center centerY, double The width in pixels sigmaPixels, double The background background, Wcs The Wcs of the image for which the fit was wcs, Unit The unit of the image for which the fit was unit)

Construction of a new SourceFittingProduct. Construction of a new

SourceFittingProduct with the given peak intensity, pixel coordinates of the center, width and background.

Arguments

SourceFittingProduct (double The peak intensity peak, double The x-pixel-coordinate of the center centerX, double The y-pixel-coordinate of the center centerY, double The width in pixels sigmaPixels, double The background background, Wcs The Wcs of the image for which the fit was wcs, Unit The unit of the image for which the fit was unit)

double The peak intensity **peak** [INPUT, MANDATORY, default=no default value]

double The x-pixel-coordinate of the center **centerX** [INPUT, MANDATORY, default=no default value]

as double

double The y-pixel-coordinate of the center **centerY** [INPUT, MANDATORY, default=no default value]

as double

double The width in pixels **sigmaPixels** [INPUT, MANDATORY, default=no default value]

double The background **background** [INPUT, MANDATORY, default=no default value]

Wcs The Wcs of the image for which the fit was **wcs** [INPUT, MANDATORY, default=no default value]

made, as Wcs

Unit The unit of the image for which the fit was **unit** [INPUT, MANDATORY, default=no default value]

made, as Unit

SourceFittingProduct (double The peak intensity peak, double The x-pixel-coordinate of the center centerX, double The y-pixel-coordinate of the center centerY, double The sigmaXPixels sigmaXPixels, double The sigmaYPixels sigmaYPixels, double The rotationAngle rotationAngle, double The background background, Wcs The Wcs of the image for which the fit was wcs, Unit The unit of the image for which the fit was unit)

Construction of a new SourceFittingProduct. Construction of a new

SourceFittingProduct with the given peak intensity, pixel coordinates of the center, rotation angle, widths and background.

Arguments

double The peak intensity **peak** [INPUT, MANDATORY, default=no default value]

double The x-pixel-coordinate of the center **centerX** [INPUT, MANDATORY, default=no default value]

as double

double The y-pixel-coordinate of the center **centerY** [INPUT, MANDATORY, default=no default value]

as double

double The sigmaXPixels **sigmaXPixels** [INPUT, MANDATORY, default=no default value]

double The sigmaYPixels **sigmaYPixels** [INPUT, MANDATORY, default=no default value]

SourceFittingProduct (double The peak intensity peak, double The x-pixel-coordinate of the center centerX, double The y-pixel-coordinate of the center centerY, double The sigmaXPixels sigmaXPixels, double The sigmaYPixels sigmaYPixels, double The rotationAngle rotationAngle, double The background background, Wcs The Wcs of the image for which the fit was wcs, Unit The unit of the image for which the fit was unit)

double The rotationAngle **rotationAngle** [INPUT, MANDATORY, default=no default value]

double The background **background** [INPUT, MANDATORY, default=no default value]

Wcs The Wcs of the image for which the fit was **wcs** [INPUT, MANDATORY, default=no default value]

made, as Wcs

Unit The unit of the image for which the fit was **unit** [INPUT, MANDATORY, default=no default value]

made, as Unit

Methods

***Image* evaluate (Image The image image)**

Evaluation of this SourceFittingProduct. Evaluates this

SourceFittingProduct for the given image.

Argument

Image The image **image** [INPUT, MANDATORY, default=no default value]

Return

Image

An image representing the evaluation of this SourceFittingProduct for the given image.

***double* getPeak**

Returns the peak intensity for this SourceFittingProduct. Returns

the value for the peak parameter.

Return

double

Returns the peak intensity for this SourceFittingProduct.

***double* getPeakError**

Returns the error on the peak intensity for this

SourceFittingProduct. Returns the error on the peak parameter.

Return

double

Returns the error on the peak intensity for this SourceFittingProduct.

double getCenterX

Returns the x-pixel-coordinate for the center for this

SourceFittingProduct. Returns the value for the centerX parameter.

Return

double

Returns the x-pixel-coordinate for the center for this SourceFittingProduct.

double getCenterXError

Returns the error on the x-pixel-coordinate for the center for

this SourceFittingProduct. Returns the error on the centerX parameter.

Return

double

Returns the error on the x-pixel-coordinate for the center for this SourceFittingProduct.

double getCenterY

Returns the y-pixel-coordinate for the center for this

SourceFittingProduct. Returns the value of the centerY parameter.

Return

double

Returns the y-pixel-coordinate for the center for this SourceFittingProduct.

double getCenterYError

Returns the error on the y-pixel-coordinate for the center for

this SourceFittingProduct. Returns the error on the centerY parameter.

Return

double

Returns the error on the y-pixel-coordinate for the center for this SourceFittingProduct.

double getCenterRA

Returns the right ascension for the center for this

SourceFittingProduct. Returns the value of the centerRA parameter.

Return

double

Returns the right ascension for the center for this SourceFittingProduct.

double getCenterDec

Returns the declination for the center for this

SourceFittingProduct. Returns the value of the centerDec parameter.

Return

double

Returns the declination for the center for this SourceFittingProduct.

double getSigmaPixels

Returns the width for this SourceFittingProduct in pixels. Returns

the value of the sigmaPixels parameter.

Return

double

Returns the width for this SourceFittingProduct in pixels.

double getSigmaPixelsError

Returns the error on the width for this SourceFittingProduct in

pixels. Returns the error on the sigmaPixels parameter.

Return

double

Returns the error the width for this SourceFittingProduct in pixels.

double getSigmaXPixels

Returns the width in the x-direction for this SourceFittingProduct

in pixels. Returns the value of the sigmaXPixels parameter.

Return

double

Returns the width in the x-direction for this SourceFittingProduct in pixels.

double getSigmaXPixelsError

Returns the error on width in the x-direction for this

SourceFittingProduct in pixels. Returns the error on the sigmaYPixels parameter.

Return

double

Returns the error on the width in the x-direction for this SourceFittingProduct in pixels.

double getSigmaYPixels

Returns the width in the y-direction for this SourceFittingProduct

in pixels. Returns the value of the sigmaYPixels parameter.

Return

double

Returns the width in the y-direction for this SourceFittingProduct in pixels.

double getSigmaYPixelsError

Returns the error on width in the y-direction for this

SourceFittingProduct in pixels. Returns the error on the sigmaYPixels parameter.

Return

double

Returns the error on the width in the y-direction for this SourceFittingProduct in pixels.

double getSigmaArcsec

Returns the width for this SourceFittingProduct in arcsec. Returns

the value of the sigmaArcsec parameter.

Return

double

Returns the width for this SourceFittingProduct in arcsec.

double getSigmaArcsecError

Returns the error on the width for this SourceFittingProduct in

arcsec. Returns the error on the sigmaArcsec parameter.

Return

double

Returns the error the width for this SourceFittingProduct in arcsec.

double getSigmaXArcsec

Returns the width in the x-direction for this SourceFittingProduct

in arcsec. Returns the value of the sigmaXArcsec parameter.

Return

double

Returns the width in the x-direction for this SourceFittingProduct in arcsec.

double getSigmaXArcsecError

Returns the error on width in the x-direction for this

SourceFittingProduct in pixels. Returns the error on the sigmaXArcsec parameter.

Return

double

Returns the error on the width in the x-direction for this SourceFittingProduct in arcsec.

double getSigmaYArcsec

Returns the width in the y-direction for this SourceFittingProduct

in arcsec. Returns the value of the sigmaYArcsec parameter.

Return

double

Returns the width in the y-direction for this SourceFittingProduct in arcsec.

double getSigmaYArcsecError

Returns the error on width in the y-direction for this

SourceFittingProduct in pixels. Returns the error on the sigmaYArcsec parameter.

Return

double

Returns the error on the width in the y-direction for this SourceFittingProduct in arcsec.

***double* getRotationAngle**

Returns the rotation angle for this SourceFittingProduct in

radians. Returns the value for the "rotationAngle" parameter.

Return

double

Returns the rotation angle for this SourceFittingProduct in radians.

***double* getRotationAngleError**

Returns the error on the rotation angle for this

SourceFittingProduct in radians. Returns the error on the "rotationAngle" parameter.

Return

double

Returns the error on the rotation angle for this SourceFittingProduct in radians.

***double* getRotationAngleDegrees**

Returns the rotation angle for this SourceFittingProduct in

degrees.

Return

double

Returns the rotation angle for this SourceFittingProduct in degrees.

***double* getRotationAngleDegreesError**

Returns the error on the rotation angle for this

SourceFittingProduct in degrees.

Return

double

Returns the error on the rotation angle for this SourceFittingProduct in degrees.

***double* getRotationAngleRadians**

Returns the rotation angle for this SourceFittingProduct in

radians.

Return

double

Returns the rotation angle for this SourceFittingProduct in radians.

***double* getRotationAngleRadiansError**

Returns the error on the rotation angle for this

SourceFittingProduct in radians.

Return

double

***double* getRotationAngleRadiansError**

Returns the error on the rotation angle for this SourceFittingProduct in radians.

***double* getPositionAngleDegrees**

Returns the position angle for this SourceFittingProduct in

degrees.

Return

double

Returns the position angle for this SourceFittingProduct in degrees.

***double* getPositionAngleDegreesError**

Returns the error on the position angle for this

SourceFittingProduct in degrees.

Return

double

Returns the error on the position angle for this SourceFittingProduct in degrees.

***double* getPositionAngleRadians**

Returns the position angle for this SourceFittingProduct in

radians.

Return

double

Returns the position angle for this SourceFittingProduct in radians.

***double* getPositionAngleRadiansError**

Returns the error on the position angle for this

SourceFittingProduct in radians.

Return

double

Returns the error on the position angle for this SourceFittingProduct in radians.

***double* getBackground**

Returns the background intensity for this SourceFittingProduct.

Returns the value of the background parameter.

Return

double

Returns the background intensity for this SourceFittingProduct.

***double* getBackgroundError**

Returns the error on the background intensity for this

SourceFittingProduct. Returns the error on the background parameter.

***double* getBackgroundError**

Return

double

Returns the error on the background intensity for this SourceFittingProduct.

***double* getSlopeX**

Returns the slope of the background along the x-axis for this

SourceFittingProduct. Returns the value of the slopeX parameter.

Return

double

Returns the slope of the background along the x-axis for this SourceFittingProduct.

***double* getSlopeXError**

Returns the error on the slope of the background along the x-axis

for this SourceFittingProduct. Returns the error on the slopeX parameter.

Return

double

Returns error on the slope of the background along the x-axis for this SourceFittingProduct.

***double* getSlopeY**

Returns the slope of the background along the y-axis for this

SourceFittingProduct. Returns the value of the slopeY parameter.

Return

double

Returns the slope of the background along the y-axis for this SourceFittingProduct.

***double* getSlopeYError**

Returns the error on the slope of the background along the y-axis

for this SourceFittingProduct. Returns the error on the slopeY parameter.

Return

double

Returns error on the slope of the background along the y-axis for this SourceFittingProduct.

***double* getPixel2Arcsec**

Converts the given length in pixels to a length in arcsec.

Converts the given length in pixels to a length in arcsec for the given Wcs.

Return

double

The length in arcsec.

***Wcs* getWcs**

Returns the Wcs of the image for which the fit was made. Returns

***Wcs* getWcs**

the Wcs of the image for which the fit was made for this SourceFittingProduct.

Return

Wcs

Returns the Wcs of the image for which the fit was made for this SourceFittingProduct.

***String* getUnit**

Returns the unit of the image for which the fit was made. Returns

the unit of the image for which the fit was made for this SourceFittingProduct.

Return

String

Returns the unit of the image for which the fit was made for this SourceFittingProduct.

***boolean* hasErrors**

Checks whether errors are available. Checks whether errors are

available for this SourceFittingProduct. This is necessary to make sure that the old products (without the errors) can still be visualised in the designated explorer.

Return

boolean

True if errors are available for this SourceFittingProduct; false otherwise.

setPeak (double Peak value value, double Error on the peak value error)

Setting the peak intensity. Sets the peak intensity and the

corresponding error for this SourceFittingProduct.

Arguments

`double Peak value` **`value`** `[INPUT, MANDATORY, default=no default value]`

`double Error on the peak value` **`error`** `[INPUT, MANDATORY, default=no default value]`

setCenter (double The x-pixel-coordinate of the centerXValue, double Error on the x-pixel-coordinate centerXError, double The y-pixel-coordinate of the centerYValue, double Error on the y-pixel-coordinate centerYError)

Setting the centroid of the fitted 2D gaussian. Sets the centroid

and corresponding error of the fitted 2D gaussian. If the given Wcs is valid, the corresponding sky coordinates are stored too (but without errors).

Arguments

`double The x-pixel-coordinate of the` **`centerXValue`** `[INPUT, MANDATORY, default=no default value]`

centroid of the fitted 2D gaussian.

`double Error on the x-pixel-coordinate` **`centerXError`** `[INPUT, MANDATORY, default=no default value]`

setCenter (double The x-pixel-coordinate of the centerXValue, double Error on the x-pixel-coordinate centerXError, double The y-pixel-coordinate of the centerYValue, double Error on the y-pixel-coordinate centerYError)

of the centroid of the fitted 2D gaussian.

`double The y-pixel-coordinate of the` **`centerYValue`** `[INPUT, MANDATORY, default=no default value]`

centroid of the fitted 2D gaussian.

`double Error on the y-pixel-coordinate` **`centerYError`** `[INPUT, MANDATORY, default=no default value]`

of the centroid of the fitted 2D gaussian.

setSigma (double Standard deviation of the symmetric 2D value, double Error on the standard deviation of the error)

Setting the standard deviation of the fitted 2D gaussian. Sets the

standard deviation and the corresponding error of the fitted 2D gaussian.

Arguments

`double Standard deviation of the symmetric 2D` **`value`** `[INPUT, MANDATORY, default=no default value]`

gaussian

`double Error on the standard deviation of the` **`error`** `[INPUT, MANDATORY, default=no default value]`

symmetric 2D gaussian.

setSigma (double Value of the standard deviation in sigmaXValue, doule Error on the standard deviation in sigmaXError, double Value of the standard deviation in sigmaYValue, double Error on the standard deviation in sigmaYError)

Setting the standard deviation in the x- and y-direction of the

fitted 2D gaussian. Sets the standard deviation in the x- and y-direction and the corresponding error of the fitted 2D gaussian.

Arguments

`double Value of the standard deviation in` **`sigmaXValue`** `[INPUT, MANDATORY, default=no default value]`

the x-direction of the asymmetric 2D gaussian [pixels]

`doule Error on the standard deviation in` **`sigmaXError`** `[INPUT, MANDATORY, default=no default value]`

the x-direction of the asymmetric 2D gaussian [pixels]

`double Value of the standard deviation in` **`sigmaYValue`** `[INPUT, MANDATORY, default=no default value]`

the y-direction of the asymmetric 2D gaussian [pixels]

`double Error on the standard deviation in` **`sigmaYError`** `[INPUT, MANDATORY, default=no default value]`

the y-direction of the asymmetric 2D gaussian [pixels]

setRotationAngle (double Rotation angle of the asymmetric 2D value, double Error on the rotation angle of the error)

Setting the rotation angle of the fitted 2D gaussian. Sets the

setRotationAngle (double Rotation angle of the asymmetric 2D value, double Error on the rotation angle of the error)

rotation angle and the corresponding error of the fitted 2D gaussian.

Arguments

double Rotation angle of the asymmetric 2D **value** [INPUT, MANDATORY, default=no default value]

gaussian [radians]

double Error on the rotation angle of the **error** [INPUT, MANDATORY, default=no default value]

asymmetric 2D gaussian [radians]

setBackground (double Value of the constant background value, double Error on the constant background error)

Setting the background. Sets the (constant) background and the

corresponding error.

Arguments

double Value of the constant background **value** [INPUT, MANDATORY, default=no default value]

double Error on the constant background **error** [INPUT, MANDATORY, default=no default value]

setBackground (double Value of the background backgroundValue, doule Error on the background backgroundError, double slopeXValue, double Error on the slope of the slopeXError, double slopeYValue, double Error on the slope of the slopeYError)

Setting the background. Sets the value of the background with a

linear slope in the x- and y-direction and the corresponding error.

Arguments

double Value of the background **backgroundValue** [INPUT, MANDATORY, default=no default value]

doule Error on the background **backgroundError** [INPUT, MANDATORY, default=no default value]

double **slopeXValue** [INPUT, MANDATORY]

double Error on the slope of the **slopeXError** [INPUT, MANDATORY, default=no default value]

background in the x-direction

double **slopeYValue** [INPUT, MANDATORY]

double Error on the slope of the **slopeYError** [INPUT, MANDATORY, default=no default value]

background in the y-direction

See also

- Developers Manual: herschel.ia.dataset.image.SourceFittingProduct

1.409. SpectralLineList

Full Name:	herschel.ia.dataset.spectrum.SpectralLineList
Type:	Java Class -
Import:	from herschel.ia.dataset.spectrum import SpectralLineList
Category	Spectra

Description

SpectralLineList is a simple Product which implements the Herschel LineList definition.

There are utility methods to set and get all of the Columns and MetaData defined in the Herschel LineList Definition document. The methods, setXyz() and getXyz() are defined to set and get the data from Column (or MetaData) "xyz". The method, hasXyz(), can be used to decide on the presence of a Column (or MetaData) "xyz".

None of the defined Columns is obligatory. When a Column is requested which is not present an IndexOutOfBoundException is raised. Use hasXyz() to avoid this exception.

It is not forbidden to include other Columns (or MetaData) than the predefined ones. There are no utility methods for them.

See also

- Developers Manual: herschel.ia.dataset.spectrum.SpectralLineList

History

- `2010-09-29 - DK:.`

1.410. SpectralSimpleCube

Full Name:	herschel.ia.dataset.spectrum.SpectralSimpleCube
Type:	Java Class -
Import:	from herschel.ia.dataset.spectrum import SpectralSimpleCube
Categories	Spectra Data cubes

Description

SpectralSimpleCube is a SimpleCube implementation where the 3rd axis represents a spectral axis.

It is made such that both the image-toolkits and the spectrum-toolkits work on it. Contract: There are 3 axis in the cube: x1, x2 and x3. To be more specific, let's assume that x1 is the RA-axis, x2 is the DEC-axis and x3 is the Spectral-axis. Within the SpectralSimpleCube can be 1-d ArrayDatasets, 2-d ArrayDatasets and 3-d ArrayDatasets. 3-d sets are of dimension [x3,x2,x1] and are interpreted as spectral stacks of images. 2-d sets are of dimension [x2,x1] and are interpreted image-wise. 1-d sets are of dimension [x3] and are interpreted spectrum-wise. Other combinations of axes are not allowed (no [x3,x1] planes e.g) under penalty of loosing performance of the image/spectral toolkits.

Some instantiations of SpectralSimpleCube might contain an error cube instead of a weight cube. In order to make the spectrum toolboxes (ia.toolbox.spectrum) work there are provisions to convert these errors into weights. These automatically kick in when getWeight is requested and no weight column could be found but only an error column. The toolbox operations would operate with the weights (possibly transforming them too) and then write back the (transformed) flux and the error (after being on-the-fly converted from the weight). See setWeight. This behaviour can be turned off by using setAutoConversion. The default of autoConversion is True.

Example

Example 1: for SpectralSimpleCube

```
flux = Double3d(10,100,100)         # cube of 10 images of size 100 * 100
ssc = SpectralSimpleCube()
ssc.setFlux( flux )
it = ssc.getSpectralSegmentIterator()
while it.hasNext() :
    sflux = it.next().getFlux()           # get flux for the 10000 segments
```

Limitations

SpectralSimpeCube still **is** a SimpleCube and can be addressed as such. If you do so, the special methods of SpectralSimpleCube are **not** guarandeed to work.

See also

- Developers Manual: herschel.ia.dataset.spectrum.SpectralSimpleCube

1.411. Spectrum1d

Full Name:	herschel.ia.dataset.spectrum.Spectrum1d
Type:	Java Class -
Import:	from herschel.ia.dataset.spectrum import Spectrum1d
Category	Spectra

Description

Spectrum1d is an extension of AbstractSpectrumDataset which in turn is an extension of (Strict)Table-Dataset.

It is a 1-dimensional incarnation of a SpectrumDataset.

Spectrum1d implements an iterator over spectral segmants when these are defined.

Within a Spectrum1d there are provisions for 4 (predefined) columns.

1. flux, a Double1d array. This flux column is more or less obligatory, otherwise there can't be much talk of a spectrum. However see below for a caveat.

2. weight. A Double1d array of the same dimensionality as flux.

   Weights can be filled with anything that can be used as a measure of relevance for the fluxes. They are necessarily >= 0.

   One obvious choice would be weight = 1/stdev^2, if such a value is available for the fluxes.

   If weight = 0, it means that the corresponding samples are irrelevant.

   Weights can be used in averaging, fitting etc.

3. flag, an Int1d array of the same size as flux. The definition of flags is of course problem dependent. See FixedMask. Whether a flag is taken into account, depends on the Task at hand and might be settable by that task.

   It is advised to use flag == 1 as BAD DATA, not to be used in any further processing. As such is would be equivalent to weight = 0.

   It is suggested to store the meaning of the flags in MetaData.

4. segment, an Int1d array of the same size as flux. The values within this array indicate to which segment the corresponding flux/weight/flag/wave belong. If this column is not present it is assumed the Spectrum1d contains only one SpectralSegment.

The other part that a spectrum needs is a frequency or wavelength scale. See AbstractSpectrumDataset.

None of the predefined columns really needs to exist. There can be reasons for any of them to be absent. If the column is not present a IndexOutOfBoundsException will be thrown. So unless you really know always use a try{}catch() or use the hasFlux(), hasWeight() etc methods.

Some instantiations of Spectrum1d might contain an error column in stead of a weight column. In order to make the spectrum toolboxes (ia.toolbox.spectrum) work there are provisions to convert these errors into weights. These automatically kick in when getWeight is requested and no weight column could be found but only an error column. The toolbox operations would operate with the weights (possibly transforming them too) and then write back the (transformed) flux and the error (after being on-the-fly converted from the weight). See setWeight. This behaviour can be turned off by using setAutoConversion. The default of autoConversion is True.

Example

Example 1: for Spectrum1d

```
flux = Double1d(100)
segment = Int1d.range(100) / 10 + 1     # define 10 segments
s1 = Spectrum1d()
s1.setFlux( flux )
s1.setSegment( segment )
it = s1.iterator()
seg = s1.getSpectralSegment()
while it.hasNext() :
    sflux = seg.getFlux()            # get flux for the 10 segments
    seg = it.next()
```

Limitations

Spectrum1d still **is** a TableDataset and can be addressed as such. If you do so, the special methods of Spectrum1d (and its predecessors, AbstractSpectrumDataset and StrictTableDataset) are **not** guaranteed to work.

See also

- Developers Manual: herschel.ia.dataset.spectrum.Spectrum1d

History

- 2006-03-06 - DK:.

1.412. Spectrum2d

Full Name:	herschel.ia.dataset.spectrum.Spectrum2d
Type:	Java Class -
Import:	from herschel.ia.dataset.spectrum import Spectrum2d
Category	Spectra

Description

Spectrum2d is an extension of AbstractSpectrumDataset which in turn is an extension of (Strict)TableDataset.

It is a 2-dimensional incarnation of a SpectrumDataset.

Within a Spectrum2d there are provisions for 3 (predefined) columns.

1. flux, a Double2d array, where the first axis runs over the spectral dimension and the second axis runs over e.g. time. In the following this time index is called sequential index or sequindex.

   There is no requirement that either axis projects monotonically or equidistantly on the frequency/sequindex scale. It is not even necessary that they are independent, ie. frequency can change over sequindex. This flux column is more or less obligatory, otherwise there can't be much talk of a spectrum.

2. weight. An Double2d array of the same dimensionality as flux.

   Weights can be filled with anything that can be used as a measure of relevance for the fluxes. They are necessarily >= 0.

   One obvious choice would be weight = 1/stdev^2, if such a value is available for the fluxes.

   If weight = 0, it means that the corresponding samples are irrelevant.

   Weights can be used in averaging, fitting etc.

3. flag, an Int2d array of the same size as flux. The definition of flags is of course problem dependent. See FixedMask. Whether a flag is taken into account, depends on the Task at hand and might be settable by that task.

   It is advised to use flag == 1 as BAD DATA, not to be used in any further processing. As such is would be equivalent to weight = 0.

   It is suggested to store the meaning of the flags in MetaData.

The other part that a spectrum needs is a frequency or wavelength scale. See AbstractSpectrumDataset.

Some instantiations of Spectrum2d might contain an error column in stead of a weight column. In order to make the spectrum toolboxes (ia.toolbox.spectrum) work there are provisions to convert these errors into weights. These automatically kick in when getWeight is requested and no weight column could be found but only an error column. The toolbox operations would operate with the weights (possibly transforming them too) and then write back the (transformed) flux and the error (after being on-the-fly converted from the weight). See setWeight. This behaviour can be turned off by using setAutoConversion. The default of autoConversion is True.

As a further extension of the Spectrum2d we introduce the concept of "subbands". Subbands are vertical splits in the flux (and weight, flag etc.) columns, equivalent to (functionality of) the segment column in Spectrum1d. The flux etc. columns are replaced with flux_1, flux_2, ... columns, depending on how many subbands were defined. The definition of the subbands is stored in small array MetaData

subbandstart and subbandlength. StrictTableDataset.setMeta(String name, Double1d data). Operations of split and join are exact inverses of each other as long as the metadata of subbandstart and subbandlength is kept consistent and provided that the subband ranges cover the complete width of flux. Otherwise what is lost will stay so.

The SpectralSegment interface is fully implemented on Spectrum2d, whether it has subbands or not. When there are no subbands, each time a next segment is requested, the next row is returned. When subbands are present, first a row from the next subband at the same sequential index is returned. When there are no more subbands, the row at the next sequindex in the first subband is returned. This behaviour can be overruled by setColumnFirst(boolean cf).

Example

Example 1: for Spectrum2d

```
flux = Double2d()
for i in range(5) : flux.append( Double1d( range(1000 ) ) + i, 0 )
flag = Int2d( 5, 1000 )
s2 = Spectrum2d( flux, None, flag )
F = DataFormatter()
print F.p( s2.getFlux( ), 6 )
s2.setSubbandStart( Int1d( [0,100,500] ) )
s2.setSubbandLength( Int1d( [100,200,400] ) )
s2.splitInSegments( ["flux","flag"] );
print F.p( s2.getFlux( 2 ) )
print s2.getSegmentCount()            # 15 = 5 * 3
it = s2.iterator()
seg = s2.getSpectralSegment()
while it.hasNext() :
    seg = it.next()
    print F.p( seg.getFlux() )
```

Limitations

Spectrum2d still **is** a TableDataset and can be addressed as such. If you do so, the special methods of Spectrum2d (and its predecessors, AbstractSpectrumDataset and StrictTableDataset) are **not** guaranteed to work.

See also

- Developers Manual: herschel.ia.dataset.spectrum.Spectrum2d

History

- 2006-03-07 - DK: .

1.413. SpectrumContainerBox

Full Name:	herschel.ia.dataset.spectrum.SpectrumContainerBox
Type:	Java Class -
Import:	from herschel.ia.dataset.spectrum import SpectrumContainerBox
Category	Spectra

Description

Defines the requirements for an object that contains a number of SpectrumContainers.

See also

- Developers Manual: herschel.ia.dataset.spectrum.SpectrumContainerBox

1.414. SpectrumContainer

Full Name:	herschel.ia.dataset.spectrum.SpectrumContainer
Type:	Java Class -
Import:	from herschel.ia.dataset.spectrum import SpectrumContainer
Category	Spectra

Description

Interface used to access spectrum information sitting in diverse data structures.

These can include table datasets that contain several individual spectra (such as obtained from observations with HIFI) or spectra organized into 3d cubes with two space dimensions and one spectral dimension. The implementation of spectrum tools (ia.toolbox.spectrum, ia.toolbox.spectrum.fit) or the spectrum explorer is based on accessing the spectra through this interface so that they become independent of data formats. For example, the same tools may be used to process or inspect spectrum information steming from different instruments - provided that the instrument-specific data structure provides access through this interface.

A SpectrumContainer contains several (0 ... n) PointSpectrum's. Each PointSpectrum is assumed to consist of SpectralSegment-information (frequency, wavelength or velocity array, flux array and, optionally, weights and flags) with the per pixel information. One PointSpectrum may contain one or several segments. In addition to the segment information, the PointSpectrum may be characterized by additional attributes that hold for all segments included in a PointSpectrum. Examples are integration time, observation time, position in sky or other instrument specific information (e.g. house-keeping data).

For the PointSpectrum's included in a SpectrumContainer several underlying assumptions hold:

- All the PointSpectrum's have the same number of segments with the same indices used to access them. These indices can be obtained by the method SpectrumContainer.getSegmentCount(). For a given segment index, the corresponding segment in each PointSpectrum has the same shape (i.e. length of frequency / flux array).
- All the PointSpectrum's have the same attributes - but each PointSpectrum with its specific attribute value. The names of these attributes can be obtained by the method SpectrumContainer.getAttributeNames().
- Unit of the flux and frequency/wavelength/velocity values found within the SpectralSegment's in the PointSpectrum are the same. These units can be obtained through SpectrumContainer.getFluxUnit(), SpectrumContainer.getWaveUnit(), respectively.

Note that the SpectrumContainer.include(PointSpectrum) and SpectrumContainer.include(PointSpectrum, int) methods will work only for and instance of PointSpectrum if this satisfies the above implicit rules of the SpectrumContainer.

As an extension of the Spectral-interface, it provides access to PointSpectrum and SpectralSegment-objects:

- Spectral.getSpectralSegmentIterator(), Spectral.getSpectralSegment(int)
- Spectral.getPointSpectrumIterator(), Spectral.getPointSpectrum(int)

Since the point spectra included in Spectral may contain several segments, some care is needed to distinguish

- the indices to enumerate the SpectralSegment within Spectral,

- the indices to enumerate the SpectralSegment within PointSpectrum: Only the indices as obtained by getSegmentIndices() are permissible.
- the indices to enumerate the PointSpectrum within Spectral: the indices range from 0 up to `n-1` where `n` is the number of point spectra in the container which can be looked up by getPointSpectrumCount(). For a given PointSpectrum the index associated with this object within the container can be obtained by PointSpectrum.getIndex().

See also

- Developers Manual: herschel.ia.dataset.spectrum.SpectrumContainer

1.415. SplinesModel

Full Name:	herschel.ia.numeric.toolbox.fit.SplinesModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import SplinesModel
Category	Mathematics/Fitting

Description

General splines model of arbitrary order and with arbitrary knot settings.

It is a linear model.

Table 1.9. Splines table.

order behaviour between knots	continuity at knots	
0	piecewise constant	not continuous at all
1	piecewise linear	lines are continuous
2	parabolic pieces	1st derivatives are also continuous
3	cubic pieces	2nd derivatives are also continuous
n > 3	n-th order polynomials	(n-1)-th derivs are also continuous

The user lays out a number (<< datapoints) of knots on the x-axis at arbitrary position, generally more knots where the curvature is higher. The knots need to be monotonously increasing in x. Alternatively one can ask this class to do the lay-out which is then equidistant in x ove the user-provided range. The knots need to be monotonously increasing in x. Through these knots a cubic splines function is obtained which best fits the datapoints. One needs at least 2 knots, one smaller and one larger than the x-values in the dataset.

If the end knots are put in between the x-values in the dataset, a kind of extrapoling spline is obtained. It still works more or less.

This model is NOT for (cubic) spline interpolation.

For interpolation see: herschel.ia.numeric.toolbox.interp.CubicSplineInterpolator

A more complete worked out example is found in the HCSS-DRM (developers reference manual).

Example

Example 1: to use some methods of SplinesModel

```
knots = Double1d.range( 17 ) * 10    # make equidistant knots from 0 to 160
csm = SplinesModel( knots, 2 )
# or alternatively:
npt = 160
x = Double1d.range( npt )        # x-values
csm = SplinesModel( 17, 2, x )    # automatic layout of knots
print csm.getNumberOfParameters()    # 18 (= #knots + order - 1)
# ... fitter etc. see Fitter
```

API Summary

Constructors
SplinesModel (Double1d knots, int order) Splines on a given set of knots and a given order.
SplinesModel (Double1d knots) Splines of order 3 on a given set of knots (=cubic splines).
SplinesModel (int nrknots, int order, Double1d input) Splines with a given number of knots and order, evenly placed on
SplinesModel (int nrknots, int order, double min, double max) Splines with a given number of knots and order, evenly placed on [min,max].

Limitations

Dont put the knots too closely so that there are no datapoints in between. Otherwise nothing special. In case of problems look at the trouble shooting section.

API Details

Constructors

SplinesModel (Double1d knots, int order)

Splines on a given set of knots and a given order.

The number of parameters is (length(knots) + order - 1)

Arguments

`Double1d` **`knots`** `[INPUT, MANDATORY, default=no default value]`

- An array of arbitrarily positioned knots

`int` **`order`** `[INPUT, MANDATORY, default=no default value]`

- The order of the spline

SplinesModel (Double1d knots)

Splines of order 3 on a given set of knots (=cubic splines).

The number of parameters is (length(knots) + order - 1)

Argument

`Double1d` **`knots`** `[INPUT, MANDATORY, default=no default value]`

An array of arbitrarily positioned knots

SplinesModel (int nrknots, int order, Double1d input)

Splines with a given number of knots and order, evenly placed on

the input range.

The number of parameters is (nrknots + order - 1)

Arguments

`int` **`nrknots`** `[INPUT, MANDATORY, default=no default value]`

Number of knots.

SplinesModel (int nrknots, int order, Double1d input)

`int` **order** `[INPUT, MANDATORY, default=no default value]`

Order of splines.

`Double1d` **input** `[INPUT, MANDATORY, default=no default value]`

Range over which the knots are placed; only the min and max of this array are used.

SplinesModel (int nrknots, int order, double min, double max)

Splines with a given number of knots and order, evenly placed on [min,max].

The number of parameters is (nrknots + order - 1)

Arguments

`int` **nrknots** `[INPUT, MANDATORY, default=no default value ]`

Number of knots.

`int` **order** `[INPUT, MANDATORY, default=no default value]`

Order of splines.

`double` **min** `[INPUT, MANDATORY, default=no default value]`

Lowest knot value.

`double` **max** `[INPUT, MANDATORY, default=no default value]`

Highest knot value.

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.SplinesModel

1.416. SQRT

Full Name:	herschel.ia.numeric.toolbox.basic.Sqrt
Alias:	SQRT
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Sqrt
Category	Mathematics/General functions

Description

Returns the square root of a number or array.

If the input is an array, the output is an array of the same size, with square root values instead of the original elements.

Example

Example 1: Applying SQRT to an Int1d

```
x = Int1d([0,5,7])
print SQRT(x) # [0.0,2.23606797749979,2.6457513110645907]
```

API Summary

Jython Syntax

```
&lt;y&gt; = SQRT(&lt;x&gt;)
```

Properties

Number or array **x** [INPUT, MANDATORY, default=no default value]

Number or array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Number or array x [INPUT, MANDATORY, default=no default value]

The value or values of which to compute the square root.

Number or array y [OUTPUT, MANDATORY, default=no default value]

The square root value or values.

See also

- SQUARE
- Pow
- Developers Manual: herschel.ia.numeric.toolbox.basic.Sqrt

1.417. SQUARE

Full Name:	herschel.ia.numeric.toolbox.basic.Square
Alias:	SQUARE
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Square
Category	Mathematics/General functions

Description

Returns the square of a number or array.

If the input is an array, the output is an array of the same type and size, with squared values instead of the original elements.

Example

Example 1: Applying SQUARE to an Int1d

```
x = Int1d([-1,0,2])
print SQUARE(x) # [1,0,4]
```

API Summary

Jython Syntax

```
&lt;y&gt; = SQUARE(&lt;x&gt;)
```

Properties

Number or array **x** [INPUT, MANDATORY, default=no default value]

Number or array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Number or array x [INPUT, MANDATORY, default=no default value]

The value or values of which to compute the square.

Number or array y [OUTPUT, MANDATORY, default=no default value]

The squared value or values.

See also

- SQRT
- Pow
- Developers Manual: herschel.ia.numeric.toolbox.basic.Square

1.418. Sso

Full Name:	herschel.ia.toolbox.pointing.Sso
Type:	Java Class -
Import:	from herschel.ia.toolbox.pointing import Sso
Category	Astronomical utilities

Description

Tools for SSO centred coordinate systems.

This just removes tracking so that it appears like a fixed observation. The origin of a object-centred reference frame is defined as the instantaneous ephemeris of the solar system object to reposition SKY(X,Y) as (OCX,OCY).

Example

Example 1: Example for steps times for the target of observation obs

```
obt0 = obs.auxiliary.pointing.getStartTime() + 1 # OBT
obt1 = obs.auxiliary.pointing.getEndTime() # OBT
steps = 10
obts = range(obt0, obt1, ((obt1 - obt0) / (steps - 1)))
obts.append(obt1) # Last point is closer if range step is not integer
tc=obs.auxiliary.timeCorrelation
tais = []
for obt in obts:
     tais.append(tc.toFineTime(obt))
dirs = Sso.getDirections(obs, tais)
```

API Summary

Methods
Direction[] **getDirections** (PointingProduct pt, FineTime[] ts, BasicHorizons h, int naifid, TimeCorrProduct t2t) Get the directions of the pointing in an SSO-centred reference frame at times ts.
Direction[] **getDirections** (ObservationContext oc, FineTime[] ts, BasicHorizons h) Get the directions of the pointing in an SSO-centred reference frame at times ts.
Direction[] **getDirections** (ObservationContext oc, FineTime[] ts) Get the directions of the pointing in an SSO-centred reference frame at times ts.

Limitations

- To obtain an Horizons object you need an Ephemeris object and a directory with SSO files
- To extract data from a Pointing product via FineTimes you need the TimeCorr product.
- Pointing product is indexed by OBT, see example to convert to TAI
- For Attitude:
  - ra Right ascension in radians [0,2*pi)

- dec Declination in radians [-pi/2,pi/2]
- pos Position angle in radians [0,2*pi)

For further information consult the following classes in the HCSS Developer's Reference Manual (Javadoc API):

- herschel.share.fltdyn.math.Direction
- herschel.share.fltdyn.time.FineTime
- herschel.ia.obs.ObservationContext
- herschel.ia.obs.auxiliary.pointing.PointingProduct
- herschel.share.fltdyn.ephem.horizons.Horizons
- herschel.ia.obs.auxiliary.timecorr.TimeCorrProduct
- herschel.share.fltdyn.math.Attitude

API Details

Methods

Direction[] getDirections (PointingProduct pt, FineTime[] ts, BasicHorizons h, int naifid, TimeCorrProduct t2t)

Get the directions of the pointing in an SSO-centred reference frame at times ts.

If some calculation fails the whole process is interrupted by an exception

Arguments

`PointingProduct` **`pt`** `[INPUT, MANDATORY, default=no default value]`

Pointing product from where pointing(ts) of the S/C can be obtained

`FineTime[]` **`ts`** `[INPUT, MANDATORY, default=no default value]`

The array of FineTimes to get the SSO centred pointing directions

`BasicHorizons` **`h`** `[INPUT, MANDATORY, default=no default value]`

The BasicHorizons object that has access to information about the NAIFID body

`int` **`naifid`** `[INPUT, MANDATORY, default=no default value]`

The NAIFID of the observed SSO

`TimeCorrProduct` **`t2t`** `[INPUT, MANDATORY, default=no default value]`

The time correlation product to access the pointing product

Return

Direction[]

A Direction[] with the SSO-centred data

Direction[] getDirections (ObservationContext oc, FineTime[] ts, BasicHorizons h)

Get the directions of the pointing in an SSO-centred reference frame at times ts.

If some calculation fails the whole process is interrupted by an exception. Use this version if you what to provide your own updated Horizons.

Arguments

***Direction[]* getDirections (ObservationContext oc, FineTime[] ts, BasicHorizons h)**

`ObservationContext` **`oc`** `[INPUT, MANDATORY, default=no default value]`

An observation context for the naifid SSO

`FineTime[]` **`ts`** `[INPUT, MANDATORY, default=no default value]`

The array of FineTimes to get the SSO centred pointing directions

`BasicHorizons` **`h`** `[INPUT, MANDATORY, default=no default value]`

An Horizons ephemerides containing info about the observed SSO (NAIFID)

Return

Direction[]

A Direction[] with the SSO-centred data

***Direction[]* getDirections (ObservationContext oc, FineTime[] ts)**

Get the directions of the pointing in an SSO-centred reference frame at times ts.

If some calculation fails the whole process is interrupted by an exception, then try to use the version with more parameters. Horizons information for the naifid will be obtained from the auxiliary Horizons product.

Arguments

`ObservationContext` **`oc`** `[INPUT, MANDATORY, default=no default value]`

An observation context for the naifid SSO

`FineTime[]` **`ts`** `[INPUT, MANDATORY, default=no default value]`

The array of FineTimes to get the SSO centred pointing directions

Return

Direction[]

A Direction[] with the SSO-centred data

See also

- Developers Manual: herschel.ia.toolbox.pointing.Sso

History

- `2010-03-29 - JDS`: Changed implementation to just remove tracking
- `2010-08-18 - JDS`: Fixed time conversions
- `2010-11-19 - JDS`: Removed deprecated code without TimeCorr

1.419. StationaryWavelet

Full Name:	herschel.ia.numeric.toolbox.wavelet.StationaryWavelet
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.wavelet import StationaryWavelet

Description

Stationary Wavelet

Performs a stationary wavelet transform using an internal wavelet and an internal algorithm. This method is a translation-invariant wavelet transform. Signal is convolved with two filters (high and low filters), then decimated (even indexes and odd indexes). Whatever the level decomposition, the signal keeps the same size. Zeros are inserting in the filters (2^{j-1} zeros between each sample). As the filter creates holes, this algorithm is also called ('algorithme à trous' in French).

Here are some few comments about the examples of this help:

First example: Gives an example of possible import statements when we use the Stationary wavelet transform algorithm

Second example: In this example, one dimensional signal (signal1d) is decomposed. By default the maximum of decomposition depth is performed and the border management selected is Symmetric.

Third example: In this example, two-dimensional signal is decomposed. By default the maximum of decomposition is performed and the border management selected is Symmetric

Fourth Example: Many border managements are available with the wavelet toolbox. This window shows us how to use a specific border during decomposition

- WBorder.ZERO pads signal outside of the support with zero
- WBorder.SYMMETRIC pads signal outside of the support with symmetric values, making the signal periodic.

The same border management will be used during synthesis.

Examples

Example 1: Import statements

```
from herschel.ia.numeric.toolbox.wavelet import StationaryWavelet
from herschel.ia.numeric.toolbox.wavelet.wutil import WBorder
```

Example 2: One-dimensional signal

```
from herschel.ia.numeric.toolbox.wavelet import StationaryWavelet
signal1d = Double1d.range(256)       # Get a Double1d signal to decompose
algo = StationaryWavelet("coif3")    # Define the algorithm and wavelet to use
coefs = algo.decompose(signal1d)     # Decompose the signal into a coefficients
 tree
print coefs.depth                    # Show the decomposition depth
detail = coefs.getDetailForLevel(1)  # Get the detail node for level 1
approx = coefs.getApproximation()    # Get the approximation node at the level
 equal to the decomposition depth
PlotXY(detail.data)                  # Plot the detail coefficients
PlotXY(approx.data)                  # Plot the approximation coefficients
res = algo.synthesis(coefs)          # Synthesize a new signal from the
 coefficients tree
```

Example 2: One-dimensional signal

```
res -= signal1d                     # Subtract the original signal to compare
 their similarity
print STDDEV(signal1d),STDDEV(res)  # Evaluate the deviation (the synthesized
 signal STDDEV should be close to 0.0)
coefs.reset(1)                      # Reset the level 1 coefficients, setting
 them to 0.0
coefs.multiply(1, 2.25)             # Multiply the level 1 coefficients by
 2.25
coefs.add(1, 3.50)                  # Add 3.50 to the level 1 coefficients
coefs.divide(1, 2.00)               # Divide the level 1 coefficients by 2.00
coefs.subtract(1,5.0)               # Subtract the value 5.0 from the
 coefficients at level 1
detail = coefs.getDetailForLevel(1) # Get the detail node for level 1
print detail.scale                  # Get the scale of the detail coefficients
data = detail.data                  # Get the detail coefficients
data[1] = 3.0                       # Set the coefficient at index 1 to the
 value 3.0
detail.data = data                  # Set the detail coefficients
```

Example 3: Two-dimensional signal

```
from herschel.ia.numeric.toolbox.wavelet import StationaryWavelet
# Generate an interesting signal to decompose
totalRows = 256
totalCols = 256
signal2d = Double2d(totalRows, totalCols)
for y in range(totalRows):
    for x in range(totalCols):
        signal = 0.0
        signal += SIN(x * 0.49 - 3.25) * SIN(y * 0.12 - 2.50)
        signal += COS(x * 0.31 - 1.75) * COS(y * 0.17 - 1.25)
        signal += SIN(x * 0.26 - 0.25) * SIN(y * 0.37 + 0.00)
        signal += COS(x * 0.17 + 1.25) * COS(y * 0.18 + 1.25)
        signal += SIN(x * 0.41 + 2.75) * SIN(y * 0.05 + 2.50)
        signal += COS(x * 0.07 + 4.25) * COS(y * 0.22 + 3.75)
        signal2d.set(y, x, signal)
algo = StationaryWavelet("db5")             # Defines the algorithm and wavelet
 to use
coefs = algo.decompose(signal2d)            # Decompose our two-dimensional
 signal into a coefficients tree
horizontal = coefs.getHorizontalForLevel(1) # Get the horizontal detail node
 for level 1
vertical = coefs.getVerticalForLevel(1)     # Get the vertical detail node for
 level 1
diagonal = coefs.getDiagonalForLevel(1)     # Get the diagonal detail node for
 level 1
approximation = coefs.getApproximation()    # Get approximation node for the
 final level
Display(horizontal.data)                    # Display the horizontal detail
 coefficients
Display(vertical.data)                      # Display the vertical detail
 coefficients
Display(diagonal.data)                      # Display the diagonal detail
 coefficients
Display(approximation.data)                 # Display the diagonal detail
 coefficients
res = algo.synthesis(coefs)                 # Synthesize a two-dimensional
 signal
diff = res - signal2d                       # Subtract for comparison
print STDDEV(signal2d), STDDEV(diff)        # Evaluate the deviation (should be
 minimal for the resulting signal)
print MIN(signal2d), MIN(diff)              # Evaluate the minimum (should be
 minimal for the resulting signal)
print MAX(signal2d), MAX(diff)              # Evaluate the maximum (should be
 minimal for the resulting signal)
# --
coefs = algo.decompose(signal2d)            # Decompose our two-dimensional
 signal into a two-dimensional tree
coefs.getVerticalForLevel(2).reset()        # Reset the vertical detail
 coefficients for level 1
```

Example 3: Two-dimensional signal

```
horizontal = coefs.getHorizontalForLevel(1) # Get the horizontal detail node
 for level 1
horizontal.multiply(3.2)                    # Multiply the value 3.2 and the
 coefficients at level 1.
horizontal.add(-5.0)                        # Subtract the value 5.0 and the
 coefficients at level 1.
print horizontal.scale                      # Get the scale of the horizontal
 detail node
diagonal = coefs.getDiagonalForLevel(1)     # Get the diagonal detail
 coefficients for level 1
data = diagonal.data                        # Get the horizontal detail
 coefficients.
data.set(2, 2, 4.0)                         # Set the value 4 at location 2 for
 the level 1.
diagonal.data = data * 0.5                  # Replace diagonal coefficients
 with new ones
# --
coefs.reset(1)                       # Reset coefficients at level 1 to 0.0 for
 all details (horizontal, vertical,
                                     #   diagonal).
coefs.add(1, 7.25)                   # Add the value 7.25 to the coefficients
 for level 1, this operation is
                                     #   done for all details.
coefs.multiply(1, 3.5)               # Multiply the value 3.5 and the
 coefficients at level 1, this operation is
                                     #   done for all details.
coefs.subtract(1, -5.75)             # Subtract the value 5.0 and the
 coefficients at level 1, this operation is
                                     #   done for all details.
coefs.divide(1, 2.0)                 # Subtract the value 5.0 and the
 coefficients at level 1, this operation is
                                     #   done for all details.
```

Example 4: Extra information

```
from herschel.ia.numeric.toolbox.wavelet import DiscreteWavelet
signal1d = Double1d(10000)
algo = DiscreteWavelet("db5")
coefs = algo.decompose(signal1d)
print coefs                          # Give information on the wavelet used
print coefs.wavelet                  # Give detail description of the wavelet
 used
print coefs.wavelet.family           # Give the wavelet family
print coefs.wavelet.symmetry         # Give the symmetry of the wavelet
print coefs.wavelet.length           # Give the length of the wavelet
print coefs.wavelet.orthogonality    # Give the orthogonality of the wavelet
print coefs.border                   # Border management use during
 decomposition
# See Wavelet for more information
```

Example 5: Border management

```
from herschel.ia.numeric.toolbox.wavelet import StationaryWavelet
from herschel.ia.numeric.toolbox.wavelet.wutil import WBorder
signal1d = Double1d.range(256)
signal2d = Double2d(512, 512, 1.0)
algo = StationaryWavelet("db5")
algo.decompose(signal2d, WBorder.SYMMETRIC)
algo.decompose(signal2d, WBorder.ZERO)
algo.decompose(signal1d, WBorder.SYMMETRIC)
# See WBorder for more information
```

See also

- Developers Manual: herschel.ia.numeric.toolbox.wavelet.StationaryWavelet

1.420. statistics

Full Name:	herschel.ia.toolbox.spectrum.SpectrumStatisticsTask
Alias:	statistics
Type:	Java Task -
Import:	from herschel.ia.toolbox.spectrum import SpectrumStatisticsTask
Category	Spectra/Analysis

Description

Task to compute statistical characteristics for the spectrum data.

Mean, RMS and median are always reported - percentiles (quantiles) can be given on demand. Two modes are available: Compute the statistics over a set of PointSpectra on a bin by bin basis ('per channel') or compute the statistics for a range of bins within a single point spectrum ('across channel'). Note that we here refer to bins as frequency or wavelength bins (or whatever the wavescale unit is), sometimes also called 'channel'.

Sigma clipping is also available - a way to flag bins that are distant from the mean by more than a given multiple of of sigma (the standard deviation). Clipping refers to the acrossChannel statistics.

Example

Example 1: global spectra # some spectrum container, defined elsewhere

```
# compute just the statistics in the across channel mode:
stats = statistics(ds=spectra) # stats is a dataset
stats = statistics(ds=spectra, mode="acrossChannels") # stats is dataset
# compute the statistics in all modes
# stats is a product that also includes a 'summary' with the across channel
 statistics
stats = statistics(ds=spectra, mode="all")
# compute the statistics just in the per channel mode
stats = statistics(ds=spectra, mode="perChannel") # stats is a product but
 without "summary" TableDataset
# compute just the statistics in the across channel mode, specify the
 percentiles you want to compute and
# the range you would like to draw the statistics from:
stats = statistics(ds=spectra, percentiles=[0.2,0.8], ranges = Range(500,1500))
# same as above - this time several ranges to draw the statistics from:
stats = statistics(ds=spectra, mode = "acrossChannels", percentiles=[0.2,0.8],
 ranges = [(4200,4500),(7300,7600)])
# same as above - this time restrict the computation on selected segments and
 point spectra
stats = statistics(ds=spectra, mode = "acrossChannels", percentiles=[0.2,0.8],
 \
                   ranges = [(4200,4500),(7300,7600)], segments=[0],
 selection=[0])
```

API Summary

Properties
SpectrumContainer **ds** [INPUT, MANDATORY, default=no default value.]
String **mode** [INPUT, OPTIONAL, default=default value 'acrossChannel'.]
Object **selection** [INPUT, OPTIONAL, default=None.]

Properties
Object **segments** [INPUT, OPTIONAL, default=no default value.]
Object **stats** [OUTPUT, OPTIONAL, default=no default value.]
Object **ranges** [INPUT, OPTIONAL, default=no default value.]
Object **exclude** [INPUT, OPTIONAL, default=no default value.]
double[] **percentiles** [INPUT, OPTIONAL, default=no default value.]
Boolean **integratedFlux** [INPUT, OPTIONAL, default=false.]
Boolean **ignoreNaNs** [INPUT, OPTIONAL, default=True.]
String **variant** [INPUT, OPTIONAL, default=no default value.]
Double **clipvalue** [INPUT, OPTIONAL, default=no default value.]
Integer **clipflag** [INPUT, OPTIONAL, default=no default value.]
Integer **flagToIgnore** [INPUT, OPTIONAL, default=no default value.]

API details

Properties

SpectrumContainer ds [INPUT, MANDATORY, default=no default value.]

The input container(s) to be considered. Examples of `SpectrumContainer` are `Spectrum1d, Spectrum2d, SpectralSimpleCube`.

String mode [INPUT, OPTIONAL, default=default value 'acrossChannel'.]

Contains the output with the statistics when the task is used in the normal mode. Possible values are 'acrossChannel', 'perChannel', 'all':

- 'acrossChannel'-mode: Here, the statistics are computed over a range of neighbouring bins (or channels) - separately for each spectrum and each segments. If no range is specified, all the bins within a point spectrum contribute to the statistical characteristics - if a range is specified, the statistics computations are restricted accordingly.
- 'perChannel'-mode: Here, the statistics are computed for all the spectra included in the container on a per bin (channel) basis. Note that this assumes that the bin (channel) is uniquely characterized by a frequency or wavelength (or whatever the wavescale unit is).
- 'all': Both, the per channel and the across channel statistics are computed.

Object selection [INPUT, OPTIONAL, default=None.]

Specification of what point spectra the average should be restricted to. Different ways to specify these selections are possible:

- Specify a list of indices (in jython) of the point spectra for which the average should be taken.
- Use a dictionary (in jython) to specify a selection model (lookup for attributes, filter attributes to be included in ranges or intervals).
- Pass a string representation of a jython list or a jython dictionary. The string will be parsed and the processing delegated to the two cases above.
- Pass any java instance that implements the SelectionModel interface (for the advanced user).

Object selection [INPUT, OPTIONAL, default=None.]

See the examples in the `SelectSpectrum`-task for how to specify selections.

Object segments [INPUT, OPTIONAL, default=no default value.]

Specify what segments the operation should be applied to. There are two options available:

- Either pass an instance of a SegmentSelection which gives the information on what segments for each point spectrum included in the container.
- Specify a PyList of segment indices.

Object stats [OUTPUT, OPTIONAL, default=no default value.]

If the task is used in the per channel mode, a product is created which contains for each statistical characteristics and each sub-segment one Spectrum1d. In case 'mode' is set to 'all', the product also contains the across channel statistics as a "summary". In case the task is used in the across channel mode, the statistics are written in a TableDataset (each point spectrum included in the input spectrum container corresponds a line in the table).

Object ranges [INPUT, OPTIONAL, default=no default value.]

Specify the ranges that should be considered within the spectra when computing the 'across channels'-statistics. You have various alternatives to do that:

- Specify a 'Range'-object: Then this range object is considered for all the sub-segments as the range of pixel numbers to select.
- Specify an array of 'Range'-objects: Here, each Range object in the array is associated with the Range of pixel number to select for each sub-segments. The number of ranges specified in the array must correspond to the number of segments.
- Specify a tuple with the lower and upper bound of a wavescale interval to restrict to (specify the interval in the units of the wavescale grid of the data).
- Specify a (py-)array of tuples with the lower and upper bounds of wavescale intervals to restrict the calculation of the statistics to (specify the intervals in the units of the wavescale grid of the data).

Object exclude [INPUT, OPTIONAL, default=no default value.]

Specify the ranges that should be excluded when computing the 'across channels'-statistics. You have various alternatives to do that:

- Specify a 'Range'-object: Then this range of pixels is excluded in all sub-segments.
- Specify an array of such 'Range'-objects: Here, each range corresponds to one range of pixels for one sub-segment. The number of ranges specified in the array must correspond to the number of segments.
- Specify a tuple with the lower and upper bound of a wavescale interval to exclude (specify the interval in the units of the wavescale grid of the data).
- Specify a (py-)array of tuples with the lower and upper bounds of wavescale intervals to exclude in the calculation of the statistics (specify the intervals in the units of the wavescale grid of the data).

double[] percentiles [INPUT, OPTIONAL, default=no default value.]

Specify what percentiles should be given in the output (a list of probabilities).

Boolean integratedFlux [INPUT, OPTIONAL, default=false.]

Specify whether you also want to computed the integrated (over the whole range and just over the selected range(s)). Note that its computation is only possible for spectra with strictly monotonous wavescale.

Boolean ignoreNaNs [INPUT, OPTIONAL, default=True.]

Flag to specify that NaN's found in the flux data should be ignored when computing the statistics.

String variant [INPUT, OPTIONAL, default=no default value.]

Specify the way how to compute the average and the standard deviation:

- default: None
- disregard flagged channels: "flag"
- including weights: "weight"
- including weights and disregard flagged channels: "flag-weight"

Use the `flagToIgnore`-parameter to specify which flag values to ignore. Note that this option to compute weighted means and standard deviation as well as excluding flagged values from the computations is only available for "across channels" statistics. In case no flags or no weights are available in the data but a variant including "flag" or "weight" is specified the statistics are computed without considering flags or weights.

Double clipvalue [INPUT, OPTIONAL, default=no default value.]

Once a positive value is specified data found in the individual spectra are clipped, using a threshold `clip * sigma` where sigma is the standard deviation (rms). Clipping means that all the values beyond the clipping threshold are flagged. In presence of ranges, the clipping is restricted to the specified ranges. Note that clipping modfies the input data.

Integer clipflag [INPUT, OPTIONAL, default=no default value.]

The flag value to be specified for the clipped pixels. If clipping is applied a clippingFlag needs to be specified. Otherwise a runtime exception is thrown.

Integer flagToIgnore [INPUT, OPTIONAL, default=no default value.]

Applies only if a 'variant' with the substring 'flag' has been specified. By setting this parameter you can configure which flags should be ignored and which should be propagated as informative flags. For example, if you want to ignore channels that have the flag 1,4 or 16 set you specify a `flagToIgnore=21`. By default, the flagToIgnore is set to 0 which means that any flagged channel is ignored.

See also

- SpectrumContainer
- Developers Manual: herschel.ia.toolbox.spectrum.SpectrumStatisticsTask

History

- `2011-08-08 - melchior`: renamed from SpectrumStatistics

1.421. StatWithNaN

Full Name:	herschel.ia.numeric.toolbox.basic.StatWithNaN
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import StatWithNaN
Categories	Arrays and datasets/Reduction Mathematics/Statistics

Description

Removes NaN variables from input arrays to other numeric functions.

You can use this class with functions of type `ArrayToNumber` and `ArrayReductor`, that is, functions that take an array as input and return a number or an array of lower dimensionality. Examples are SUM, MEAN, MEDIAN, STDDEV, QRMS, MIN, MAX, PRODUCT, SKEWNESS, KURTOSIS and VARIANCE. This class returns a NaN value if the input array contains NaN values only.

Example

Example 1: Apply to a Double2d

```
flux = Double2d([[2.3,6.9,Double.NaN,3.1],
                 [1.3,Double.NaN,Double.NaN,3.3],
                 [3.4,3.8,Double.NaN,2.9],
                 [2.0,3.5,Double.NaN,Double.NaN]])
# Entire array
print StatWithNaN(MEDIAN)(flux)
# 3.2
# Dim 0
print StatWithNaN(MEDIAN)(flux,0)
# [2.15,3.8,NaN,3.1]
# Dim 1
print StatWithNaN(MEDIAN)(flux,1)
# [3.1,2.3,3.4,2.75]
# Other functions
print StatWithNaN(SUM)(flux,0)
# [9.0,14.2,NaN,9.3]
print StatWithNaN(MEAN)(flux,0)
# [2.25,4.733333333333333,NaN,3.1]
print StatWithNaN(MEDIAN)(flux,0)
# [2.15,3.8,NaN,3.1]
print StatWithNaN(STDDEV)(flux,0)
# [0.8736894948054104,1.8823743871327334,NaN,0.19999999999999996]
print StatWithNaN(QRMS)(flux,0)
# [2.3738154940938436,4.976611966656298,NaN,3.104298095651683]
print StatWithNaN(MIN)(flux,0)
# [1.3,3.5,NaN,2.9]
print StatWithNaN(MAX)(flux,0)
# [3.4,6.9,NaN,3.3]
print StatWithNaN(PRODUCT)(flux,0)
# [20.331999999999997,91.77,NaN,29.667]
print StatWithNaN(SKEWNESS)(flux,0)
# [0.2429090374756943,0.3739308813767407,NaN,-2.2406844898033775E-15]
print StatWithNaN(KURTOSIS)(flux,0)
# [-1.8984370530691632,-2.333333333333333,NaN,-2.3333333333333335]
print StatWithNaN(VARIANCE)(flux,0)
# [0.7633333333333332,3.543333333333334,NaN,0.03999999999999998]
```

API Summary

Jython Syntax
`outArray = StatWithNaN (function)(inArray [, alongDimension])`

Properties
`ArrayReductor|ArrayToNumber` **`function`** `[INPUT, MANDATORY, default=no default value]`
`Float or double array` **`inArray`** `[INPUT, MANDATORY, default=no default value]`
`Integer` **`alongDimension`** `[INPUT, OPTIONAL, default=no default value]`
`Number or array` **`outArray`** `[OUTPUT, MANDATORY, default=no default value]`

API details

Properties

`ArrayReductor|ArrayToNumber function [INPUT, MANDATORY, default=no default value]`

The function to apply to the array.

`Float or double array inArray [INPUT, MANDATORY, default=no default value]`

The array from which to remove NaN values.

`Integer alongDimension [INPUT, OPTIONAL, default=no default value]`

The dimension along which to apply the function. Array dimensions are counted from zero. See the example for more details.

`Number or array outArray [OUTPUT, MANDATORY, default=no default value]`

The result of applying the function to the input array after removing NaN values.

See also

- Developers Manual: herschel.ia.numeric.toolbox.basic.StatWithNaN

1.422. STDDEV

Full Name:	herschel.ia.numeric.toolbox.basic.StdDev
Alias:	STDDEV
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import StdDev
Category	Mathematics/Statistics

Description

Returns the standard deviation of an array.

Equivalent to SQRT(VARIANCE(x)). See this website for a definition of standard deviation.

Returns NaN (Not a Number) if any of the array elements is NaN.

Examples

Example 1: Applying STDDEV to a Float1d

```
x = Float1d([1,3,2,3,4])
print STDDEV(x) # 1.140175425099138
x = Float1d([1,Double.NaN,2,3,4])
print STDDEV(x) # NaN
# To remove NaN values (the original array is not modified):
print STDDEV(NAN_FILTER(x)) # 1.2909944487358056 (STDDEV is applied to
 [1,2,3,4])
```

Example 2: Applying StdDev to a Float1d using a specified mean value

```
# Note the explicit import statement and the different
# name: StdDev instead of STDDEV
from herschel.ia.numeric.toolbox.basic import StdDev
x = Float1d([1,3,2,3,4])
mean = 2.6
print StdDev(mean)(x) # 1.140175425099138
```

API Summary

Jython Syntax

```
&lt;y&gt; = STDDEV(&lt;x&gt;)
```

Properties

Array **x** [INPUT, MANDATORY, default=no default value]

Double **mean** [INPUT, OPTIONAL, default=no default value]

Double **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Array x [INPUT, MANDATORY, default=no default value]

The array of which to compute the standard deviation. Complex arrays are not allowed.

Double mean [INPUT, OPTIONAL, default=no default value]
The mean value to use for computing the standard deviation. See the second example for details.

Double y [OUTPUT, MANDATORY, default=no default value]
The standard deviation of the input array.

See also

- MEAN
- MEDIAN
- VARIANCE
- SKEWNESS
- KURTOSIS
- Developers Manual: herschel.ia.numeric.toolbox.basic.StdDev

1.423. stitch

Full Name:	herschel.ia.toolbox.spectrum.StitchSpectrumTask
Alias:	stitch
Type:	Java Task -
Import:	from herschel.ia.toolbox.spectrum import StitchSpectrumTask
Category	Spectra/Analysis

Description

Task for stitching the overlapping parts of spectra.

The stitching is performed on a per point spectrum (scan) basis and glues the different segments per point spectrum - once these are overlapping. When using the variant 'stitchAll' all segments found in all spectra are combined by computing an average in the overlapping regions and glueing all connected regions in the end. Note that the task does not allow (yet) to stitch segments different spectrum containers.

As long as the 'variant'-parameter is not set to 'stitchAll' the stitch task is useful only when applied to data with more than one segment per point spectrum. At the time of writing, this seems to be true only with HIFI data. When setting 'variant'-parameter to 'stitchAll' the stitch can be used as a more advanced average in which all segments within a given spectrum container are stitched (if possible) to obtain one single spectrum.

The result is again a point spectrum that still may consist of several segments - in case gaps are found between consecutive segments and the fillGaps parameter is set to False (default). If the fillGaps parameter is set to True the gaps between segments are filled with NaNs and a single segment is returned per PointSpectrum.

The stitching will work on whatever unit the wavescale is expressed in (, wavelength, wavenumber, velocity).

The way how the stitching is done is defined by the parameter 'variant' (see below). With some of the options (crossoverPoints, splitPoints, midPoint) the segments are, in a first step, cut at suitable 'split points' within overlap regions and, then, the resulting segments are glued along these split points. For further details see below.

Once the stitching is done, the resulting point spectra are included in a new spectrum container. Here, the wavescale grids of the different point spectra need to have the same shapes (number of bins). In order to achieve that the point spectra are resampled to a common wavescale grid. In case no or a zero stepsize (which is the channel width for the resampled spectra) is specified, resampling is avoided which is only possible if all the stitched point spectra happen to have all the same shape. In case the stitch spectra have wavescale grids with different shapes the spectra are resampled nevertheless. See the `stepsize` for further details on when to specify the a stepsize and how it is computed when no stepsize is specified.

Examples

Example 1: stitching using split crossoverPoints variant

```
global spectra
# all defaults - cut at crossover points and concatenate
r = stitch(ds=spectra)
# cut at crossover points and concatenate
r = stitch(ds=spectra, variant="crossoverPoints")
# cut at crossover points and concatenate, consider crossover points only by 10
 percent off the border
```

Example 1: stitching using split crossoverPoints variant

```
# to of the overlapping region
r = stitch(ds=spectra, variant="crossoverPoints", edgeTolerance=0.1)
# cut at crossover points and concatenate, resample the resulting spectra to
 1MHz channel width
r = stitch(ds=spectra, variant="crossoverPoints", edgeTolerance=0.1,
 stepsize=1.0, unit="MHz")
```

Example 2: stitching using midPoints variant

```
global spectra
# cut at mid points of the overlapping regions and concatenate
r = stitch(ds=spectra, variant="midPoints")
# cut at mid points of the overlapping regions and concatenate, resample to 1.0
 channel width (in units of the
# underlying  scale)
r = stitch(ds=spectra, variant="midPoints", stepsize=1.0)
```

Example 3: stitching using splitPoints variant

```
global spectra
# cut at predefined split points and concatenate
r = stitch(ds=spectra, variant="splitPoints", splitPoints =
 [569.75,570.75,571.7])
# cut at predefined split points and concatenate, resample to 0.1 channel width
 (in units of the underlying  scale)
r = stitch(ds=spectra, variant="splitPoints", splitPoints =
 [569.75,570.75,571.7], stepsize=0.1)
# cut at predefined split points and concatenate, resample to 0.0001 THz
 channel width
r = stitch(ds=spectra, variant="splitPoints", splitPoints =
 [0.56975,0.57075,0.57170], stepsize=0.0001, unit="THz")
```

Example 4: stitching using average variant

```
global spectra
# average the parts of the spectra that are overlapping, resample to a 1.0
 channel width
# (in units of the underlying scale)
r = stitch(ds=spectra, variant="average", stepsize=1.0)
# average the parts of the spectra that are overlapping, resample to a 1.0 MHz
 channel width,
# do not consider flags and weights
# when computing teh average (but propagate them)
r = stitch(ds=spectra, variant="average", stepsize=1.0, unit="MHz",
 avg_variant="flux")
# average the parts of the spectra that are overlapping, avoid resampling as
 far as possible
r = stitch(ds=spectra, variant="average", avg_variant="flux")
```

API Summary

Properties
SpectrumContainer **ds** [INPUT, MANDATORY, default=no default value.]
SpectrumContainer **result** [OUTPUT, MANDATORY, default=no default value.]
String **variant** [INPUT, OPTIONAL, default="crossoverPoints".]
double **edgeTolerance** [INPUT, OPTIONAL, default=0.01.]

Properties
Object **splitPoints** [INPUT, OPTIONAL, default=no default value]
double **stepsize** [INPUT, OPTIONAL, default=no default value.]
String **unit** [INPUT, OPTIONAL, default=no default value.]
String **splitpoints unit** [INPUT, OPTIONAL, default=no default value.]
String **avg variant** [INPUT, OPTIONAL, default=no default value.]
Boolean **fillGaps** [INPUT, OPTIONAL, default=False.]

API details

Properties

SpectrumContainer ds [INPUT, MANDATORY, default=no default value.]

The input container with the spectra to be stitched. Examples of `SpectrumContainer` are `Spectrum1d, Spectrum2d, SpectralSimpleCube`.

SpectrumContainer result [OUTPUT, MANDATORY, default=no default value.]

The output container with the stitched spectra.

String variant [INPUT, OPTIONAL, default="crossoverPoints".]

The parameter that specifies the way how the stitching should be done. Possible values:

- "crossoverPoints": Within overlapping ranges of the spectra a crossover point is taken to split the spectra. In case no crossover point is found a point is taken where the spectra come closest. In case many crossover points are found the one closest to the midpoint of the overlapping range is taken. The range where the crossover points are searched for is reduced by setting a non-zero edge tolerance (see parameter `edgeTolerance` below). This option only works if at most two segments 'participate' in the same overlapping range.
- "midPoint": Within the overlapping ranges of the spectra the mid point of the wavescale range of the overlap region is taken to split the spectra. Similar to the above, this option only works if at most two segments 'participate' in the same overlapping range.
- "splitPoints": Within the overlapping ranges of the spectra manually specified points are taken to split the spectra (parameter `splitPoints`). the mid point is taken. Similar to the above, this option only works if at most two segments 'participate' in the same overlapping range.
- "average": In the overlapping ranges the average of the involved spectra is taken. This option allows to stitch spectra in which more than two spectra contribute to the same overlapping part.
- "stitchAll": Try to stitch all the segments found in the SpectrumContainer, irrespective of what point spectrum the segments stem from. In the overlapping ranges, take the average of the involved spectra.

double edgeTolerance [INPUT, OPTIONAL, default=0.01.]

Reduce the range (an overlapping range in the spectra) in which cross over points are searched. With `length` denoting the length of the original overlapping range `[n,m]` (in number of pix-

double edgeTolerance [INPUT, OPTIONAL, default=0.01.]

els) the reduced range is defined by `[n+p*length, m-p*length]` where `p` is the edge tolerance.

Object splitPoints [INPUT, OPTIONAL, default=no default value]

Specify the list of split points explicitly. This parameter is used only in combination with the option `variant="splitPoints"`. The split points are specified as a Python list. Note that the number of split points shoudl correspond to the number of overlapping ranges (per point spectrum).

double stepsize [INPUT, OPTIONAL, default=no default value.]

Specify the stepsize used to define a linear wavescale step (, wavelength,...) the spectra are resampled to. The resampling is important in the following situations:

- When including the stitched point spectra in a common container. In case a `stepsize>0` is specified, a wavescale grid is constructed with the given step size. The upper and lower bounds are determined from the minimum and maximum frequencies found in all the stitched point spectra (for a give segment index). In case a `stepsize=0` is specified, the wavescale grid found in the first (stitched) point spectrum is taken as the output grid.
- When using `variant="average"` the spectra contributing to common overlapping wavescale ranges are resampled to a common wavescale grid.

When searching for crossover points, one of the spectra also needs to be resampled to a common scale - here, the wavescale scale of the other spectrum is used, though so that the `stepsize` parameter is not needed here.

String unit [INPUT, OPTIONAL, default=no default value.]

The unit the step size and/or split points are expressed in. In case no unit is specified the step size and/or the split points are assumed to be in the same units as the underlying wave scale grid. If you wish to express the step size and the split points in different units please use this parameter in combination with the `splitpoints_unit`-parameter.

String splitpoints_unit [INPUT, OPTIONAL, default=no default value.]

The unit the split points are expressed in. In case no such unit is specified the step size and/or the split points are assumed to be in the same units as specified by the `unit`-parameter or, if no such parameter is specified, in the same units as the underlying wave scale grid is expressed in.

String avg_variant [INPUT, OPTIONAL, default=no default value.]

The variant of the average to be used: whether to include flags and weights in the processing.

Boolean fillGaps [INPUT, OPTIONAL, default=False.]

If set to True the non-overlapping segments remaining after applying the stitch segment operation are combined into a single segment by filling gaps with NaN values (on a per PointSpectrum basis). In the gaps, the wave scale is defined by filling in wave scale steps of size determined by the average of all the steps found in the PointSpectrum. Finally, NaN values are added to the edges of the spectra so that all the PointSpectra (now consisting of a single segment) have the same number of bins.

See also

- SpectrumContainer
- stitch

- Developers Manual: herschel.ia.toolbox.spectrum.StitchSpectrumTask

History

- `2011-08-08 - melchior`: renamed from StitchSpectrum

1.424. StrCalibration

Full Name:	herschel.ia.toolbox.pointing.StrCalibration
Type:	Jython Task -
Import:	from herschel.ia.toolbox.pointing import StrCalibration
Category	Toolboxes/Pointing

Description

Class providing time-dependent access to the Herschel Star Tracker (STR) properties.

API Summary

Jython Syntax

```
str = StrCalibration(<odNumber>)
# get the distortion polynomes applied on-board for the epoch
# of the observation
h = str.h
k = str.k
# get the distortion polynomes to be used after applying the
# distortion maps (= distortion polynome of reference period)
h = str.href
k = str.kref
# get the invertsion of the distortion polynomes applied on-board
hinv = str.hinv
kinv = str.kinv
# get the focal length applied onboard for this epoch:
f = str.f
# get the focal length used to use before applying the distortion
# maps (=focal length of reference period)
fRef = str.fRef
# Optics temperature in degC assumed constant for complete mission
t = str.t
# Reference temperature
t0 = str.t0
# Coefficient for thermal expansion
alphaT = str.alphaT
# Speed of light, [m/s]
c = str.c
# Pixel scale [mm]
py = str.py
pz = str.pz
# Bad stars in the catalogue
badStars = str.badStars
```

Jython Syntax

```
# Mean color alphaC
alphaC = str.alphaC
# subpixel distortion maps
fovimagey = str.fovimagey
fovimagez = str.fovimagez
# Subpixel distortion map pixel scale [mm]
dpix = str.dpix
# Time offset between STR time stamps and ACMS time stamps - mil-
lisec
# STR attitudes have a later time stamp than actual sampling time
timeOffset = str.timeOffset
# Reference measurement error [arcsec]
referenceMeasurementError = str.referenceMeasurementError
```

Property

```
integer odNumber [INPUT, MANDATORY, default=NO default value ]
```

API details

Property

`integer odNumber [INPUT, MANDATORY, default=NO default value ]`

OD number to retrieve the STR calibration / properties for

History

- `2013-04-09` - BV: Initial version
- `2013-04-10` - BV: Correct imports
- `2013-06-10` - BV: Corrected h/k swap
- `2013-08-04` - BV: Minor documentation improvements
- `2013-08-13` - BV: HCSS-18458 Use version v3 of the distortion map for the late mission phase
- `2013-08-26` - BV: HCSS-18371 Time offset STR attitude
- `2013-09-25` - BV: HCSS-18602 Add star 215 to list of bad stars
- `2013-10-08` - BV: HCSS-18566 Added reference measurement sigma
- `2013-10-08` - BV: HCSS-18566 Removed star 215 from the list of bad stars - should be found by large TASTE

1.425. strExtractVelocityFromTeleCommHistory

Full Name:	herschel.ia.toolbox.pointing.strExtractVelocityFromTeleCommHistory
Type:	Jython Task -
Import:	from herschel.ia.toolbox.pointing import strExtractVelocityFromTeleCommHistory
Category	Toolboxes/Pointing

Description

Function to extract the spacecraft velocity used for on-board Star tracker aberration correction from the telecommand history.

API Summary

Jython Syntax
`(tcTime,strVelX, strVelY, strVelZ) = strExtractVelocityFromTeleCommHistory(tcHistoryProduct)`

Property
`TcHistoryProduct tcHistoryProduct [INPUT, MANDATORY, default=NO default value ]`

API details

Property

`TcHistoryProduct tcHistoryProduct [INPUT, MANDATORY, default=NO default value ]`
telecommand history product from the observation context

History

- `2013-04-09 - BV`: Initial version
- `2014-06-15 - BV`: SPR HCSS-18903 - use a different command to find the uplinked SC velocity

1.426. String1d

Full Name:	herschel.ia.numeric.String1d
Type:	Java Class -
Import:	from herschel.ia.numeric import String1d
Category	Arrays and datasets

Description

A rectangular numeric String array of rank 1.

Numeric arrays are part of the Numeric library. More information is given in the *Numeric* chapter (link in the *See also* section below).

See also

- Section 2.2 in *Scripting Guide*
- Developers Manual: herschel.ia.numeric.String1d

1.427. subtract

Full Name:	herschel.ia.toolbox.spectrum.SubtractSpectrumTask
Alias:	subtract
Type:	Java Task -
Import:	from herschel.ia.toolbox.spectrum import SubtractSpectrumTask
Category	Spectra/Analysis

Description

Task for subtracting a scalar from the flux data of spectra or for pairwise subtraction of spectra.

For the scalar mode, 'ds' and 'param' need to be specified - the input spectra and the scalar to subtract; for the pair-wise mode, 'ds1' and 'ds2' need to be set - two spectrum containers. In this pair-wise mode, the first spectrum in the second container is subtracted from the first spectrum in the first container, and so on. In case the size of the two containers is different, the result will contain a number of point spectra equal to the minimum size. Hence, the remaining spectra in the larger container are ignored. The behavior is different in case one of the datasets only contains a single spectrum: Here, the task always refers to single spectrum while iterating over all the spectra in the other container.

In the pairwise mode, the individual spectra are processed on a per frequency or wavelength bin (or whatever the wavescale unit is) basis. Hence, there is no check of whether the frequencies (or wavelengths) assigned to these bins are well aligned across the two spectra. If this is not the case, the spectra should first be resampled to a common wavescale grid.

The input data with the spectra to be processed needs to be an object that implements `SpectrumContainer` (e.g. `Spectrum1d, Spectrum2d, SpectralSimpleCube`). `SpectrumContainer`'s provides a uniform way to access spectrum data organized in quite different structures.

For a successful processing in the pair-wise mode, the data specified with `ds1` and `ds2` should be consistent, i.e. they should have the same wavescale range and spectral sampling. Furthermore, the segments found in all the spectra in the (two) containers should be consistent (same number of segments, same lengths, same segment indices - e.g. check with `ds.getSegmentIndices()`). Otherwise an exception is thrown.

Different selection schemes are available for selecting the point spectra or the segments to be processed. The most simple scheme is to specify lists of point spectrum indices (`selection=[1,3,4,2]`). In this case, the operation (scalar- or pair-operation) is just taken over the point spectra with corresponding indices. Instead of specifying a list of indices (which requires the knowledge of these) other recipes to specify a selection are available. Typical recipes test attribute values to match certain values (e.g. `selection={"bbtype":[6031, 6032]}`) or to be included in a given interval (e.g. `selection={"LoFrequency":(999.98,1000.02)}`). These recipes we refer to as selection models. The different options to specify selections can be combined. Here, an AND logic is adopted. See the parameter descriptions and the examples below for further detail. Alternatively, look in the `SelectSpectrum`-task.

Similarly, you can specify what segments to consider. In the most simple case, you can just specify the indices of segments to be considered (for the scalar operation `segments=[1,3,4]` and for the pair operation `segments1=[1,3,4], segments2=[3,6,5]`). In more advanced situations you can use a `SegmentSelection` object. Note that the pairs of segments to be processed need to be consistent - otherwise the processing will fail.

You can specify whether the original data sets should be overwritten with the 'overwrite' flag. By default no data is overwritten. In the pair-wise mode, it is not always possible to overwrite the data - in particular if non-trivial segment selections are specified.

Examples

Example 1: for scalar subtract:

```
global spectra # defined elsewhere
subtract = SubtractSpectrumTask()
spectraOut = subtract(ds=spectra, param=2.1)
# restrict the subtraction to a suitable selection of spectra and return the
 processed
# just select by index:
spectraOut = subtract(ds=spectra, selection=[0,1,2,3], param=2.1)
# select by looking up suitable attributes attached to the spectra to be
 selected (here: "bbtype" matched):
spectraOut = subtract(ds=spectra, selection={"bbtype":[6031]}, param=2.1)
# select by looking up suitable attributes attached to the spectra to be
 selected (here: "Chopper" in given ranges):
spectraOut = subtract(ds=spectra, selection={"Chopper":([-4.4,5.9],0.2),
 "bbtype":[6031]}, param=2.1)
# select by looking up suitable attributes attached to the spectra to be
 selected
# (here: "LoFrequency" in given interval):
spectraOut = subtract(ds=spectra, selection={"LoFrequency":(4000.0,5000.0)},
 param=2.1)
# the same as above - this time by by modifying the input spectra:
spectra = subtract(ds=spectra, selection={"bbtype":[6613]}, param=2.1,
 overwrite=True)
spectra = subtract(ds=spectra, selection=[0,1,2,3], param=2.1, overwrite=True)
spectra = subtract(ds=spectra, selection={"Chopper":(4.,7.), "bbtype":[6613]},
 param=2.1, overwrite=True)
spectra = subtract(ds=spectra, selection={"LoFrequency":(550.0,570.0)},
 param=2.1, overwrite=True)
```

Example 2: for pairwise subtract:

```
global spectra1, spectra2 # defined elsewhere
subtract = SubtractSpectrumTask()
spectraOut = subtract(ds1=spectra1, ds2=spectra2)
# restrict the subtraction to a suitable selection of spectra and return the
 processed
# just select by index:
spectraOut = subtract(ds1=spectra1, ds2=spectra2, selection=[0,1,2,3])
# select by looking up suitable attributes attached to the spectra to be
 selected (here: "bbtype" matched):
spectraOut = subtract(ds1=spectra1, ds2=spectra2, selection={"bbtype":[6613,
 6031]})
# select by looking up suitable attributes attached to the spectra to be
 selected (here: "Chopper" in given ranges):
spectraOut = subtract(ds1=spectra1, ds2=spectra2, selection={"Chopper":(4.,7.),
 "bbtype":[6613]})
# select by looking up suitable attributes attached to the spectra to be
 selected
# (here: "LoFrequency" in given interval):
spectraOut = subtract(ds1=spectra1, ds2=spectra2, selection={"LoFrequency":
(550.0,570.0)})
```

API Summary

Properties
SpectrumContainer **ds** [INPUT, OPTIONAL, default=no default value.]
Double **param** [INPUT, OPTIONAL, default=no default value.]
Object **selection** [INPUT, OPTIONAL, default=None.]
PyDictionary|Map<String,Set<Object>> **lookup selection** [INPUT, OPTIONAL, default=no default value.]

Properties
PyList **index selection** [INPUT, OPTIONAL, default=No default value.]
Boolean **overwrite** [INPUT, OPTIONAL, default=False.]
SpectrumContainer **ds1** [INPUT, OPTIONAL, default=no default value.]
SpectrumContainer **ds2** [INPUT, OPTIONAL, default=no default value.]
Object **segments** [INPUT, OPTIONAL, default=no default value.]
Object **ds1Segments** [INPUT, OPTIONAL, default=no default value.]
Object **ds2Segments** [INPUT, OPTIONAL, default=no default value.]
SpectrumContainer **result** [OUTPUT, OPTIONAL, default=no default value]

API details

Properties

SpectrumContainer ds [INPUT, OPTIONAL, default=no default value.]

Input container to be processed by the task in case the task is used as a scalar operation. Examples of `SpectrumContainer` are `Spectrum1d, Spectrum2d, SpectralSimpleCube`.

Double param [INPUT, OPTIONAL, default=no default value.]

The scalar parameter to be considered when the task is used as scalar operation.

Object selection [INPUT, OPTIONAL, default=None.]

Specify what point spectra the operation should be applied to. Different ways to specify these selections are possible:

- Specify a list of indices (in jython) of the point spectra for which the subtract should be applied.
- Use a dictionary (in jython) to specify a selection model (lookup for attributes, filter attributes to be included in ranges or intervals).
- Pass a string representation of a jython list or a jython dictionary. The string will be parsed and the processing delegated to the two cases above.
- Pass any java instance that implements the SelectionModel interface (for the advanced user).

See the examples below or the SelectSpectrumTask for how to specify selections.

PyDictionary|Map<String,Set<Object>> lookup_selection [INPUT, OPTIONAL, default=no default value.]

Specify a PyDictionary with keys given by the attribute name to look up and a list of admissible values. Behind the scenes, a DiscreteValueSelectionModel is instantiated. This parameter is actually obsolete but is kept for historical reasons (use `selection`).

PyList index_selection [INPUT, OPTIONAL, default=No default value.]

Specify a PyList with the indices of the point spectra to be considered. This parameter is actually obsolete but is kept for historical reasons (use `selection`).

Boolean overwrite [INPUT, OPTIONAL, default=False.]

Specify whether the input data container can be reused - the values found therein is overwritten.

SpectrumContainer ds1 [INPUT, OPTIONAL, default=no default value.]

First input container for pair-wise operations.

SpectrumContainer ds2 [INPUT, OPTIONAL, default=no default value.]

Second input container for pair-wise operations.

Object segments [INPUT, OPTIONAL, default=no default value.]

Specify what segments the operation should be applied to. There are two options available:

- Either pass an instance of a SegmentSelection which gives the information on what segments for each point spectrum included in the container.
- Specify a PyList of segment indices.

Object ds1Segments [INPUT, OPTIONAL, default=no default value.]

Specify the segment selection to be associated with 'ds1'.

Object ds2Segments [INPUT, OPTIONAL, default=no default value.]

Specify the segment selection to be associated with 'ds2'.

SpectrumContainer result [OUTPUT, OPTIONAL, default=no default value]

Result object containing the results of the operation applied.

See also

- SpectrumContainer
- select
- Developers Manual: herschel.ia.toolbox.spectrum.SubtractSpectrumTask

History

- `2011-08-08 - melchior`: renamed from SubtractSpectrum

1.428. SUM

Full Name:	herschel.ia.numeric.toolbox.basic.Sum
Alias:	SUM
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Sum
Category	Arrays and datasets/Reduction

Description

Returns the sum of all the elements of an array.

This function can return a global sum for the whole array, or compute a sum along a given dimension of a multidimensional array if the `dim` parameter is specified. For example, you could sum the elements on each row, or column, of a two-dimensional array. See also the example below.

Example

Example 1: Applying SUM to an Int2d

```
x = Int2d( [ [1,2], [-1,3] ])
print SUM(x)    # 5
print SUM(x, 0) # [0,5]
print SUM(x, 1) # [3,2]
```

API Summary

Jython Syntax

```
&lt;y&gt; = SUM(&lt;x&gt; [,&lt;dim&gt;])
```

Properties

Array **x** [INPUT, MANDATORY, default=no default value]

Integer **dim** [INPUT, MANDATORY, default=no default value]

Number or array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Array x [INPUT, MANDATORY, default=no default value]

The array whose elements are to be summed.

Integer dim [INPUT, MANDATORY, default=no default value]

The dimension along which to compute the sum.

Number or array y [OUTPUT, MANDATORY, default=no default value]

The sum of all the elements of the array, or the sums along a given dimension if the dim parameter is specified.

See also

- PRODUCT
- Developers Manual: herschel.ia.numeric.toolbox.basic.Sum

1.429. SurfaceSplinesModel

Full Name:	herschel.ia.numeric.toolbox.fit.SurfaceSplinesModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import SurfaceSplinesModel
Category	Mathematics/Fitting

Description

General surface splines model of arbitrary order and with arbitrary knot settings.

It is a linear model.

Surface splines are direct product of a splines model in the x-direction with a splines model in the y-direction.

The number of parameters is (xknotlength + order - 1) * (yknotlength + order - 1)

The SplinesModel has more information about order and knots.

A more complete worked out example is found in the HCSS-DRM (developers reference manual).

Example

Example 1: to use SurfaceSplinesModel

```
nxk = 17
nyk = 11
xknots = Double1d.range( nxk ) * 10      # make knots from 0 to 160
yknots = Double1d.range( nyk ) * 10      # make knots from 0 to 100
csm = SurfaceSplinesModel( xknots, yknots, 2 )
print csm.getNumberOfParameters()      # 216 = (nxk + order - 1)*(nyk + order -
 1)
# ... fitter etc. see Fitter
```

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.SurfaceSplinesModel

1.430. Swt

Full Name:	herschel.ia.numeric.toolbox.wavelet.walgo.Swt
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.wavelet.walgo import Swt

Description

A stationary wavelet transform algorithm

Example

Example 1: Jython Usage Example

```
from herschel.ia.numeric.toolbox.wavelet.walgo    import Swt
from herschel.ia.numeric.toolbox.wavelet.wlibrary import WaveletLoader
from herschel.ia.numeric.toolbox.wavelet.wutil    import WBorder
from herschel.ia.numeric.toolbox.wavelet.wdemo    import WSigGenerator
from herschel.ia.numeric.toolbox.wavelet.wdemo    import WReadImage
# two-dimensional example
image = Double2d(129, 129)
for y in range(129):
    for x in range(129):
        image[y, x] = COS(y / 3) + COS(y / 7) + COS(x / 60) + COS(x / 97)
swt = Swt()
w = WaveletLoader.loadDiscreteWavelet("db2")
tree = swt.decompose(image, w, WBorder.SYMMETRIC)
d = Display(image)
# -- synthesis
swt.synthesis(tree)
res = tree.getRoot().getData()
```

See also

- Developers Manual: herschel.ia.numeric.toolbox.wavelet.walgo.Swt

History

- `2010-09-10 - BM`: Initial version.

1.431. TablePlotter

Full Name:	herschel.ia.gui.explorer.table.TablePlotter
Type:	Java Class -
Import:	from herschel.ia.gui.explorer.table import TablePlotter
Category	Arrays and datasets/Display

Description

TablePlotter is a GUI tool to view and analyze instances of TableDataset.

It can be invoked in HIPE by right-clicking on a TableDataset. It can also be invoked by a Jython command line in Hipe through jython script. By default, the second column is plotted as y data and the first column as x data unless x and y indices are specified. All the defaults can be changed via a group of buttons and selectors.

Examples

Example 1: Invoke TablePlotter from command line

```
from herschel.ia.dataset import Column, TableDataset
from herschel.ia.gui.explorer.table import TablePlotter
from herschel.share.component import WindowManager
from herschel.ia.numeric.toolbox.basic import Cos, Sin
#
#create a TableDataset
table = TableDataset()
x = Double1d.range(100)
y1=SIN(x)
y2=COS(x)
table["x-data"]=Column(x)
table["y1-data"] = Column(y1)
table["y2-data"] = Column(y2)
############################################ #

#Load a TablePlotter with a table input only
############################################ #
tpl=TablePlotter(table)
tpl.name = "Test TablePlotter"
wm=WindowManager.getDefault()
#Add the tpl to the window
wm.addWindow(tpl.name, tpl.component, 1)
############################################ #
# Load a TablePlotter with a 2d flags
############################################ #
flags2d = Bool2d(100, 3, 1) # all data points are deselected  (red cross)
columnState=0 # 1:all column; 0:current column
tpl1=TablePlotter(table, flags2d, columnState)
wm=WindowManager.getDefault()
#Load TablePlotter
wm.addWindow('test 2d flags', tpl1.component, 1)
############################################
# Load TablePlotter with a 1d flag
############################################
flags1d= Bool1d(100, 0) # set all datapoints are selected (blue dot)
flags1d[0:10]=1         # set 0-10 data points are deselected
flags1d[90:100]=1         # set 90-100 data points are deselected
tpl2=TablePlotter(table, flags1d )
wm=WindowManager.getDefault()
#Load TablePlotter
wm.addWindow('test 1d flags', tpl2.component, 1)
```

Example 2: create a TablePlotter where (x,y) are taken from columns (1,3)

```
from herschel.ia.gui.explorer.table import TablePlotter
from herschel.ia.dataset import TableDataset, Column
from herschel.ia.gui.explorer.table import OverPlotter
from herschel.share.component import WindowManager
from herschel.ia.numeric.toolbox.basic import Cos, Sin, ArcCos, ArcSin
#plot the 3rd column vs the first column from tds table
table = TableDataset()
x = Double1d.range(100)
y1=SIN(x)
y2=COS(x)
y3=ARCCOS(x)
y4=ARCSIN(x)
table["x-data"]=Column(x)
table["y1-data"] = Column(y1)
table["y2-data"] = Column(y2)
table["y3-data"] = Column(y3)
table["y4-data"] = Column(y4)
tablePlotter = TablePlotter(table, 0, 3)
wm=WindowManager.getDefault()
#Load TablePlotter
wm.addWindow('test TablePlotter', tablePlotter.component, 1)
```

Example 3: Use TablePlotter as a plug-in

```
from herschel.share.component import WindowManager
from javax.swing import JPanel
from javax.swing import JLabel
from javax.swing import BoxLayout
from javax.swing import JFrame
from java.awt import Dimension
from java.awt import BorderLayout
from herschel.ia.dataset import Column, TableDataset
from herschel.ia.gui.explorer.table import TablePlotter
from herschel.ia.numeric.toolbox.basic import Sin, Cos
from herschel.ia.numeric import Double1d
#create a TableDataset
table = TableDataset()
x = Double1d.range(100)
y1=SIN(x)
y2=COS(x)
table["x-data"]=Column(x)
table["y1-data"] = Column(y1)
table["y2-data"] = Column(y2)
#create a container to hold the controls/components
pane=JPanel()
#set the layout, there are many different kinds users can choose according to
 his/her need
pane.setLayout(BoxLayout(pane, BoxLayout.Y_AXIS))
#add control one, a lael
title = JLabel('Display TablePlotter')
pane.add(title)
#Add TablePlotter to the pane
tablePlotter=TablePlotter(table)
pane.add(tablePlotter.component)
pane.setPreferredSize(Dimension(tablePlotter.component.width,
 tablePlotter.component.height))
pane.setSize(Dimension(tablePlotter.component.width,
 tablePlotter.component.height))
#add to your application
frame =JFrame('TablePlotter as Plug-in demo')
frame.setDefaultLookAndFeelDecorated(1)
frame.setResizable(1)
frame.getContentPane().add(pane, BorderLayout.CENTER)
frame.setSize(800, 800)
frame.pack
frame.visible=1
```

API Summary

Constructors
TablePlotter Default constructor.
TablePlotter (TableDataset tds) Constructor which takes a TableDataset.
TablePlotter (TableDataset tds, integer xIndex, integer yIndex) Constructor which takes a TableDataset and coordinates.
TablePlotter (TableDataset tds, integer xIndex, integer yIndex, boolean initialColumnMode) Constructor which takes a TableDataset, coordinates and column mode.
TablePlotter (TableDataset tds, integer xIndex, integer yIndex, boolean initialColumnMode, boolean dataExtractionAllowed) Constructor which takes a TableDataset, coordinates, column and data extraction modes.
TablePlotter (TablePlotter tds, Bool1d flags) Constructor which takes a TableDataset and an array of flagged rows.
TablePlotter (TableDataset tds, Bool1d flags, integer xIndex, int yIndex) Constructor which takes a TableDataset, coordinates and an array of flagged rows.
TablePlotter (TableDataset tds, Bool1d flags, integer xIndex, int yIndex, boolean dataExtractionAllowed) Constructor which takes a TableDataset, coordinates, an array of flagged rows and data extraction mode.
TablePlotter (TableDataset tds, Bool2d flags, boolean initialColumnState) Constructor which takes a TableDataset, a matrix of flagged coordinates and initial column state.
TablePlotter (TableDataset tds, Bool2d flags, boolean initialColumnState, boolean dataExtractionAllowed) Constructor which takes a TableDataset, a matrix of flagged coordinates, column and data extraction modes.

Methods
JComponent **getComponent** Inherit from explorer This method will return a TablePlotter as a
String **getDescription**
String **getName**
setObject (Object data)
LayerStruct **getActiveLayerStruct** This is a command line API. It is a getter to get the active

Limitations

The TableDataset has to be one dimensional numeric array such as Double1d, Float1d, Long1d, Short1d or Complex1d.

Miscellaneous

All the examples here are given in Jython script

API Details

Constructors

`TablePlotter`

Default constructor.

This is a default constructor which will be called in DatasetInspector. It initializes the data object Listener support for data extraction.

`TablePlotter (TableDataset tds)`

Constructor which takes a TableDataset.

This constructor will initialize and create an instance of TablePlotter for the input dataset.

Argument

`TableDataset` **`tds`** `[INPUT, MANDATORY, default=no default value ]`

A TableDataset to be plotted. It contains one or more columns.

`TablePlotter (TableDataset tds, integer xIndex, integer yIndex)`

Constructor which takes a TableDataset and coordinates.

This constructor will initiate an instance of a TablePlotter with three inputs, a table dataset, the xIndex and the yIndex.

Arguments

`TableDataset` **`tds`** `[INPUT, MANDATORY, default=no default value ]`

A TableDataset to be plotted and viewed. It contains one or more columns.

`integer` **`xIndex`** `[INPUT, MANDATORY, default=no default value ]`

The xIndex is an integer which indicates the x column data

`integer` **`yIndex`** `[INPUT, MANDATORY, default=no default value ]`

The yIndex is an integer which indicates the y column data

Example

create a TablePlotter where (x,y) are taken from columns (1,3)

```
from herschel.ia.gui.explorer.table import TablePlotter
from herschel.ia.dataset import TableDataset, Column
from herschel.ia.gui.explorer.table import OverPlotter
from herschel.share.component import WindowManager
from herschel.ia.numeric.toolbox.basic import Cos, Sin, ArcCos, ArcSin
#plot the 3rd column vs the first column from tds table
table = TableDataset()
x = Double1d.range(100)
y1=SIN(x)
y2=COS(x)
y3=ARCCOS(x)
y4=ARCSIN(x)
table["x-data"]=Column(x)
table["y1-data"] = Column(y1)
table["y2-data"] = Column(y2)
table["y3-data"] = Column(y3)
table["y4-data"] = Column(y4)
tablePlotter = TablePlotter(table, 0, 3)
wm=WindowManager.getDefault()
```

TablePlotter (TableDataset tds, integer xIndex, integer yIndex)

```
#Load TablePlotter
wm.addWindow('test TablePlotter', tablePlotter.component, 1)
```

TablePlotter (TableDataset tds, integer xIndex, integer yIndex, boolean initialColumnMode)

Constructor which takes a TableDataset, coordinates and column mode.

This constructor will initiate an instance of an TablePlotter with three input, a table, the xIndex and the yIndex.

Arguments

`TableDataset` **`tds`** `[INPUT, MANDATORY, default=no default value ]`

A TableDataset to be plotted and viewed. It contains one or more columns.

`integer` **`xIndex`** `[INPUT, MANDATORY, default=no default value ]`

The xIndex is an integer which indicates the x column data

`integer` **`yIndex`** `[INPUT, MANDATORY, default=no default value]`

The yIndex is an integer which indicates the y column data

`boolean` **`initialColumnMode`** `[INPUT, MANDATORY, default=no default value]`

- It indicates the Column Selector's initial mode. It can be ALL_COLUMNS or !ALL_COLUMNS

TablePlotter (TableDataset tds, integer xIndex, integer yIndex, boolean initialColumnMode, boolean dataExtractionAllowed)

Constructor which takes a TableDataset, coordinates, column and data extraction modes.

This constructor will initiate an instance of a TablePlotter with three inputs: a table, the xIndex and the yIndex.

Arguments

`TableDataset` **`tds`** `[INPUT, MANDATORY, default=no default value]`

A TableDataset to be plotted and viewed. It contains one or more columns.

`integer` **`xIndex`** `[INPUT, MANDATORY, default=no default value]`

The xIndex is an integer which indicates the x column data

`integer` **`yIndex`** `[INPUT, MANDATORY, default=no default value]`

The yIndex is an integer which indicates the y column data

`boolean` **`initialColumnMode`** `[INPUT, MANDATORY, default=no default value ]`

- Input, boolean, it indicates the Column Selector's initial mode. It can be ALL_COLUMNS or !ALL_COLUMNS

`boolean` **`dataExtractionAllowed`** `[INPUT, MANDATORY, default=no default value]`

- boolean, true, the extraction is allowed, the "Extract" button is enabled; false, the extraction is not allowed, the "Extract" button is disabled.

TablePlotter (TablePlotter tds, Bool1d flags)

Constructor which takes a TableDataset and an array of flagged rows.

This constructor will initiate an instance of TablePlotter with pre selected and deselected columns. The Bool1d flag indicate which rows are selected and which rows are deselected.

TablePlotter (TablePlotter tds, Bool1d flags)

Arguments

`TablePlotter` **`tds`** `[INPUT, MANDATORY, default=no default value]`

The tds is the TableDataset to be plotted. It contains one or more columns.

`Bool1d` **`flags`** `[INPUT, MANDATORY, default=no default value ]`

The flags specifies which rows are deselected and which rows are selected

TablePlotter (TableDataset tds, Bool1d flags, integer xIndex, int yIndex)

Constructor which takes a TableDataset, coordinates and an array of flagged rows.

This constructor will initiate an instance of TablePlotter with specified x and y data.

Arguments

`TableDataset` **`tds`** `[INPUT, MANDATORY, default=no default value]`

A TableDataset to plot and view. The tds contains one or more columns.

`Bool1d` **`flags`** `[INPUT, MANDATORY, default=no default value]`

The flags specify which rows are de-selected and selected for all columns

`integer` **`xIndex`** `[INPUT, MANDATORY, default=no default value]`

The xIndex specifies the x axis data

`int` **`yIndex`** `[INPUT, MANDATORY, default=no default value]`

The yIndex specifies the y axis data

TablePlotter (TableDataset tds, Bool1d flags, integer xIndex, int yIndex, boolean dataExtractionAllowed)

Constructor which takes a TableDataset, coordinates, an array of flagged rows and data extraction mode.

This constructor will initiate an instance of TablePlotter with specified x and y data.

Arguments

`TableDataset` **`tds`** `[INPUT, MANDATORY, default=no default value ]`

A TableDataset to plot and view. The tds contains one or more columns.

`Bool1d` **`flags`** `[INPUT, MANDATORY, default=no default value]`

The flags specify which rows are de-selected and selected for all columns

`integer` **`xIndex`** `[INPUT, MANDATORY, default=no default value]`

The xIndex specifies the x axis data

`int` **`yIndex`** `[INPUT, MANDATORY, default=no default value ]`

The yIndex specifies the y axis data

`boolean` **`dataExtractionAllowed`** `[INPUT, MANDATORY, default=no default value. ]`

- boolean, true, the extraction is allowed, the "Extract" button is enabled; false, the extraction is not allowed, the "Extract" button is disabled.

TablePlotter (TableDataset tds, Bool2d flags, boolean initialColumnState)

Constructor which takes a TableDataset, a matrix of flagged coordinates and initial column state.

This constructor will initiate an instance of TablePlotter with two inputs, a TableDataset and its flags. The flags tell which data points are selected and deselected. TablePlotter will display two

TablePlotter (TableDataset tds, Bool2d flags, boolean initialColumnState)

layer plots and one of which is for selected data (blue colour) and the other is for deselected data (red colour).

Arguments

`TableDataset` **`tds`** `[INPUT, MANDATORY, default=no default value A]`

TableDataset to be plotted and viewed. It contains one or more columns.

`Bool2d` **`flags`** `[INPUT, MANDATORY, default=no default value A flags to]`

indicate which data points are selected and deselected.

`boolean` **`initialColumnState`** `[INPUT, MANDATORY, default=no default]`

value. This argument indicates the initial column selector's state

TablePlotter (TableDataset tds, Bool2d flags, boolean initialColumnState, boolean dataExtractionAllowed)

Constructor which takes a TableDataset, a matrix of flagged coordinates, column and data extraction modes.

Arguments

`TableDataset` **`tds`** `[INPUT, MANDATORY, default=no default value]`

A TableDataset to be plotted and viewed. It contains one or more columns.

`Bool2d` **`flags`** `[INPUT, MANDATORY, default=no default value ]`

Flags to indicate which data points are selected and deselected.

`boolean` **`initialColumnState`** `[INPUT, MANDATORY, default=no default value]`

This argument indicates the initial column selector's state

`boolean` **`dataExtractionAllowed`** `[INPUT, MANDATORY, default=no default value]`

true if the extraction is allowed, the "Extract" button is enabled; false, the extraction is not allowed, the "Extract" button is disabled.

Methods

JComponent **getComponent**

Inherit from explorer This method will return a TablePlotter as a

component so that users can plug it in his/her own applications.

Return

JComponent

- return the TablePlotter as a component

String **getDescription**

Return

String

the description of this class

String **getName**

String **getName**

Return

String

the name of the application

setObject (Object data)

Argument

`Object` **`data`** `[INPUT, MANDATORY, default=no default value ]`

If data is a TableDataset, it will be processed and plotted. If data is not a TableDataset, nothing will be displayed.

LayerStruct **getActiveLayerStruct**

This is a command line API. It is a getter to get the active

LayerStruct object. Through the active LayerStruct object, extracted table and flags can be retrieved.

Return

LayerStruct

return the current active LayerStruct

See also

- Developers Manual: herschel.ia.gui.explorer.table.TablePlotter

History

- `2006-10-06 - first`: version of TablePlotter
- `2007-12-21 - Major`: GUI changes
- `2011-09-29 - Implemented`: HCSS-11396
- `2011-10-14 - Implemented`: HCSS-14232

1.432. tableToVo

Full Name:	herschel.ia.toolbox.util.TableToVoTask
Alias:	tableToVo
Type:	Java Task -
Import:	from herschel.ia.toolbox.util import TableToVoTask

Description

Task for writing VO files

Task for writing VO files.

API Summary

Jython Syntax
`Full Signature:` `TableToVoTask(&lt;file&gt;, &lt;table&gt;)`

Properties
`String` **`file`** `[INPUT, MANDATORY, default=no default value]`
`TableDataset` **`table`** `[INPUT, MANDATORY, default=no default value]`

API details

Properties

`String file [INPUT, MANDATORY, default=no default value]`
Destination file.

`TableDataset table [INPUT, MANDATORY, default=no default value]`
table dataset.

See also

- Developers Manual: herschel.ia.toolbox.util.TableToVoTask

1.433. TAN

Full Name:	herschel.ia.numeric.toolbox.basic.Tan
Alias:	TAN
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Tan
Category	Mathematics/Trigonometry

Description

Returns the tangent of a number or array.

If the input is an array, the output is an array of the same type and size, with tangent values instead of the original elements. For complex numbers, the tangent is computed for the real and imaginary part.

Example

Example 1: Applying TAN to a Float1d

```
x = Float1d([0,0.5])
print  TAN(x) # [0.0,0.5463025]
```

API Summary

Jython Syntax
<y> = TAN(<x>)

Properties
Number or array **x** [INPUT, MANDATORY, default=no default value]
Number or array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Number or array x [INPUT, MANDATORY, default=no default value]

The angle or angles of which to compute the tangent, in radians.

Number or array y [OUTPUT, MANDATORY, default=no default value]

The tangent value or values.

See also

- ARCTAN
- COS
- SINH
- TANH

- Developers Manual: herschel.ia.numeric.toolbox.basic.Tan

1.434. TANH

Full Name:	herschel.ia.numeric.toolbox.basic.TanH
Alias:	TANH
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import TanH
Category	Mathematics/Trigonometry

Description

Returns the hyperbolic tangent of a number or array.

If the input is an array, the output is an array of the same type and size, with hyperbolic tangent values instead of the original elements. For complex numbers, the hyperbolic tangent is computed for the real and imaginary part.

API Summary

Jython Syntax
`<y> = TANH(<x>)`

Properties
`Number or array x [INPUT, MANDATORY, default=no default value]`
`Number or array y [OUTPUT, MANDATORY, default=no default value]`

API details

Properties

`Number or array x [INPUT, MANDATORY, default=no default value]`
The angle or angles of which to compute the hyperbolic tangent, in radians.

`Number or array y [OUTPUT, MANDATORY, default=no default value]`
The hyperbolic tangent value or values.

See also

- TAN
- ARCTAN
- Developers Manual: herschel.ia.numeric.toolbox.basic.TanH

1.435. tiledImage

Full Name:	herschel.ia.toolbox.image.TiledImageTask
Alias:	tiledImage
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import TiledImageTask
Category	Images/Analysis

Description

This is a task to convert an image to a TiledImage.

This is a task to convert an image to a TiledImage.

Examples

Example 1: This is how you can convert an image to a TiledImage, taking the flag into account:

```
tiled = tiledImage(image = myImage, flag = True)
```

Example 2: This is how you can convert an image to a TiledImage, not taking the flag into account:

```
tiled = tiledImage(image = myImage, flag = False)
```

API Summary

Jython Syntax

```
TiledImage(image, True)
```

Properties

```
Image image [INPUT, MANDATORY, default=None]
boolean flag [INPUT, MANDATORY, default=True]
TiledImage result [OUTPUT, MANDATORY, default=None]
```

API details

Properties

`Image image [INPUT, MANDATORY, default=None]`

This is the input image.

`boolean flag [INPUT, MANDATORY, default=True]`

This parameter indicates whether the flag must be taken into account.

`TiledImage result [OUTPUT, MANDATORY, default=None]`

THis is the resulting TiledImage.

See also

- Developers Manual: herschel.ia.toolbox.image.TiledImageTask

1.436. translate

Full Name:	herschel.ia.toolbox.image.TranslateTask
Alias:	translate
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import TranslateTask
Category	Images/Analysis

Description

This is a task to translate an image, and adapts its Wcs.

This is a task to translate an image, and adapts its Wcs. The image is translated over the given parameters (if ra = 1h00m00s, the image will be moved 1h00m00s to the right. At the left, 0.0's will be added which will be masked out.)

Examples

Example 1: This is an example of how you can translate an image using pixel coordinates :

```
translated = translate(image = myImage, x = 5, y = 7)
```

Example 2: This is an example of how you can translate an image using sky coordinates :

```
translated = translate(image = myImage, ra = 0.03, dec = 0.03)
```

API Summary

Jython Syntax
`translate()`
`translate(image, ra, dec)`
`translate(image, x, y)`

Properties
`Image image [INPUT, MANDATORY, default=None]`
`double x [INPUT, OPTIONAL, default=0.0]`
`double y [INPUT, OPTIONAL, default=0.0]`
`Angle ra [INPUT, OPTIONAL, default=Angle(0.0, ANGLE.DEGREES)]`
`Angle dec [INPUT, OPTIONAL, default=Angle(0.0, ANGLE.DEGREES)]`
`Image translatedImage [OUTPUT, MANDATORY, default=None]`

API details

Properties

`Image image [INPUT, MANDATORY, default=None]`

This is the input image.

double x [INPUT, OPTIONAL, default=0.0]

This is the number of pixels to translate in x-direction.

double y [INPUT, OPTIONAL, default=0.0]

This is tthe number of pixels to translate in y-direction.

Angle ra [INPUT, OPTIONAL, default=Angle(0.0, ANGLE.DEGREES)]

This is the amount of degrees to move in right ascension.

Angle dec [INPUT, OPTIONAL, default=Angle(0.0, ANGLE.DEGREES)]

This is amount of degrees to move in declination.

Image translatedImage [OUTPUT, MANDATORY, default=None]

This is the resulting translated image.

See also

- Developers Manual: herschel.ia.toolbox.image.TranslateTask

1.437. TRANSPOSE

Full Name:	herschel.ia.numeric.toolbox.matrix.MatrixTranspose
Alias:	TRANSPOSE
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.matrix import MatrixTranspose
Category	Mathematics/Matrices

Description

Transposes a matrix.

Example

Example 1: Transposing a 2D array

```
x=Int2d([ [1,2],[3,4],[5,6] ])
print TRANSPOSE(x) # [ [1,3,5],[2,4,6] ]
```

API Summary

Jython Syntax

```
&lt;y&gt;=TRANSPOSE(&lt;x&gt;)
```

Property

`Array2dData x [INPUT, MANDATORY, default=no default value]`

Miscellaneous

TRANSPOSE is an alias for MatrixTranspose.FUNCTION

API details

Property

`Array2dData x [INPUT, MANDATORY, default=no default value]`

Input must be a 2d array as defined by the numeric library.

See also

- Developers Manual: herschel.ia.numeric.toolbox.matrix.MatrixTranspose

1.438. transpose

Full Name:	herschel.ia.toolbox.image.TransposeTask
Alias:	transpose
Type:	Java Task -
Import:	from herschel.ia.toolbox.image import TransposeTask
Category	Images/Analysis

Description

This is a task to transpose an image, and adapts its Wcs.

This is a task to transpose an image, and adapts its Wcs. The following types of transposing are possible :

- FLIP_VERTICAL : The image is flipped upside-down
- FLIP_HORIZONTAL : The image is flipped left-right
- FLIP_DIAGONAL : The image is mirrored around the diagonal
- FLIP_ANTIDIAGONAL : The image is mirrored around the antidiagonal
- ROTATE_90 : The image is rotated 90 degrees
- ROTATE_180 : The image is rotated 180 degrees
- ROTATE_270 : The mirror is rotated 270 degrees

Example

Example 1: This is how you can flip an image horizontally :

```
flipped = transpose(image = myImage, type = TransposeTask.FLIP_HORIZONTAL)
```

API Summary

Jython Syntax

```
transpose()
transpose(image, TransposeTask.FLIP_VERTICAL)
```

Properties

`Image image [INPUT, MANDATORY, default=None]`
`TransposeType type [INPUT, MANDATORY, default=None]`
`Image transposedImage [OUTPUT, MANDATORY, default=None]`

API details

Properties

`Image image [INPUT, MANDATORY, default=None]`

This is the input image.

TransposeType type [INPUT, MANDATORY, default=None]
This is the type of transposition to apply (FLIP_VERTICAL, FLIP_HORIZONTAL, FLIP_DIAGONAL, FLIP_ANTIDIAGONAL, ROTATE_90, ROTATE_180 or ROTATE_270).

Image transposedImage [OUTPUT, MANDATORY, default=None]
This is the resulting transposed image.

See also

- Developers Manual: herschel.ia.toolbox.image.TransposeTask

1.439. UniformPrior

Full Name:	herschel.ia.numeric.toolbox.fit.UniformPrior
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import UniformPrior
Category	Mathematics/Fitting

Description

Uniform prior distribution.

A uniform prior is a improper prior (i.e. its integral is unbound). Because of that it always needs limits, low and high, such that -Inf < low < high < +Inf. `Pr(x) = 1 / ( high - low ) if low < x < high 0 otherwise` For location parameters.

domain2Unit: u = (d - lo) / (hi - lo)

unit2Domain: d = u * (hi - lo) + lo

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.UniformPrior

1.440. UNIQ_SORTED

Full Name:	herschel.ia.numeric.toolbox.basic.UniqSorted
Alias:	UNIQ_SORTED
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import UniqSorted
Category	Arrays and datasets/Element selection

Description

Returns the unique elements of a sorted array.

Note that this function gives incorrect results if applied to an unsorted array. On the other hand, it offers a much better performance than UNIQ applied to an unsorted array.

Two accessory functions provide additional features:

- UNIQ.BY_INDEX returns an array with the *indices* of the unique array elements, rather than their values. When an element appears multiple times, the output array includes the index of the first encountered instance.
- UNIQ.COUNT_UNIQ_VALUES returns the number of unique values in an array.

Examples

Example 1: Apply UNIQ_SORTED on a Double1d

```
x = Double1d([-1,4,2,7,2,-6,-8,3,2])
print UNIQ_SORTED(SORT(x))        # [-8.0,-6.0,-1.0,2.0,3.0,4.0,7.0]
```

Example 2: Apply UNIQ_SORTED.BY_INDEX

```
x = SORT(Double1d([-1,4,2,7,2,-6,-8,3,2]))
print UNIQ_SORTED.BY_INDEX(x)     # [0,1,2,3,6,7,8]
```

Example 3: Apply UNIQ_SORTED.COUNT_UNIQ_VALUES

```
x = Double1d([4, 5, 7, 4, 8, 8, 3, 1, 5])
print UNIQ_SORTED.COUNT_UNIQ_VALUES(SORT(x))  # 6
```

API Summary

Jython Syntax

```
&lt;y&gt;=UNIQ_SORTED(&lt;x&gt;)
```

Properties

Sorted 1-D array **x** [INPUT, MANDATORY, default=no default value]

an array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Sorted 1-D array x [INPUT, MANDATORY, default=no default value]
The array of which to return the unique values.

an array y [OUTPUT, MANDATORY, default=no default value]
The array containing the unique elements of the input array.

See also

- Developers Manual: herschel.ia.numeric.toolbox.basic.UniqSorted

1.441. UNIQ

Full Name:	herschel.ia.numeric.toolbox.basic.Uniq
Alias:	UNIQ
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Uniq
Category	Arrays and datasets/Element selection

Description

Returns the unique elements of an array.

For a much improved performance, sort the array beforehand with SORT and use the UNIQ_SORTED function instead.

Three accessory functions provide additional features:

- UNIQ.BY_INDEX returns an array with the *indices* of the unique array elements, rather than their values. When an element appears multiple times, the output array includes the index of the first encountered instance.
- UNIQ.COUNT_UNIQ_VALUES returns the number of unique values in an array. Passing sorted arrays greatly improves performance.
- UNIQ.IS_UNIQ returns *True* if the array has no duplicated values, *False* otherwise. This function is deprecated: use UNIQ.COUNT_UNIQ_VALUES and compare the result with the size of the input array.

Examples

Example 1: Apply UNIQ to a Double1d

```
x = Double1d([-1,4,2,7,2,-6,-8,3,2])
print UNIQ(x)                    # [-8.0,-6.0,-1.0,2.0,3.0,4.0,7.0]
print UNIQ(SORT(x))              # [-8.0,-6.0,-1.0,2.0,3.0,4.0,7.0]
```

Example 2: Use UNIQ.BY_INDEX

```
x = Double1d([-1,4,2,7,2,-6,-8,3,2])
print UNIQ.BY_INDEX(x)         # [0,1,2,3,5,6,7]
print UNIQ.BY_INDEX(SORT(x))   # [0,1,2,3,6,7,8]
# SORT(x) = [-8.0,-6.0,-1.0,2.0,2.0,2.0,3.0,4.0,7.0]
```

Example 3: Use UNIQ.COUNT_UNIQ_VALUES

```
x = Double1d([4, 5, 7, 4, 8, 8, 3, 1, 5])
print UNIQ.COUNT_UNIQ_VALUES(x)  # 6
```

Example 4: Use UNIQ.IS_UNIQ

```
x = Double1d([-1,4,7,-2,-6,-8,3,2])
print UNIQ.IS_UNIQ(x)        # 0 (false)
print UNIQ.IS_UNIQ(SORT(x))  # 1 (true)
```

API Summary

Jython Syntax
<y>=UNIQ(<x>)

Properties
1-D array **x** [INPUT, MANDATORY, default=no default value]
1-D array **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

1-D array x [INPUT, MANDATORY, default=no default value]
The array of which to return the unique elements.

1-D array y [OUTPUT, MANDATORY, default=no default value]
The array containing the unique elements of the input array.

See also

- Developers Manual: herschel.ia.numeric.toolbox.basic.Uniq

1.442. UNWRAP

Full Name:	herschel.ia.numeric.toolbox.basic.UnWrap
Alias:	UNWRAP
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import UnWrap
Categories	Arrays and datasets/Manipulation Mathematics/Trigonometry

Description

Unwraps the radian phase angles in an array.

It adds multiples of 2pi when absolute jumps between consecutive elements are greater than the tolerance parameter. The original array is not modified.

Example

Example 1: example

```
x = 2*Math.PI*Double1d.range(100)/100
# Define original
y = (SIN(x)+3*SIN(2*x))
# Wrap original
yTest = Double1d(y)
idx = yTest.where(yTest &gt; Math.PI)
yTest[idx]=yTest[idx]-2*Math.PI
idx = yTest.where(yTest &lt; -Math.PI)
yTest[idx]=yTest[idx]+2*Math.PI
# Unwrap
yp = UNWRAP(yTest)
print y == yp
```

API Summary

Jython Syntax

```
&lt;result&gt;=UNWRAP(&lt;array&gt;,[&lt;alpha&gt;])
```

Properties

```
Array array [INPUT, MANDATORY, default=no default value]
Double alpha [INPUT, OPTIONAL, default=Math.PI/4]
```

API details

Properties

Array array [INPUT, MANDATORY, default=no default value]

The input array.

Double alpha [INPUT, OPTIONAL, default=Math.PI/4]

The tolerance parameter. The default values is the equivalent of 45 degrees in radians.

See also

- Developers Manual: herschel.ia.numeric.toolbox.basic.UnWrap

1.443. updateDataset

Full Name:	herschel.ia.toolbox.util.UpdateDatasetTask
Alias:	updateDataset
Type:	Java Task -
Import:	from herschel.ia.toolbox.util import UpdateDatasetTask
Category	Arrays and datasets/Manipulation

Description

Updates a dataset in a context hierarchy so that the changes are preserved in the context tree.

Example

Example 1: Modify the value of the "mask" dataset within the first product of "level1" inside "obs"

```
updateDataset(root=obs, path='.refs["level1"].product["mask"]', row=2,
 column=1, value=55)
```

API Summary

Jython Syntax

```
updateDataset(&lt;root&gt;, &lt;path&gt;, &lt;row&gt;, [&lt;colum-
n&gt;,] &lt;value&gt;)
```

Properties

Product **root** [INPUT, MANDATORY, default=no default value]
String **path** [INPUT, MANDATORY, default=no default value]
Integer **row** [INPUT, MANDATORY, default=no default value]
Integer **column** [INPUT, OPTIONAL, default=no default value]
Object **value** [INPUT, OPTIONAL, default=no default value]

API details

Properties

Product root [INPUT, MANDATORY, default=no default value]

Root product or context to modify.

String path [INPUT, MANDATORY, default=no default value]

Relative path from the root context to the dataset to be modified.

Integer row [INPUT, MANDATORY, default=no default value]

Row of the cell in the dataset to be updated.

Integer column [INPUT, OPTIONAL, default=no default value]

Column of the cell in the dataset to be updated. Can be null in case of array data of dimension 1.

Object value [INPUT, OPTIONAL, default=no default value]
The new value for the given cell in the dataset.

See also

- Developers Manual: herschel.ia.toolbox.util.UpdateDatasetTask

History

- `2013-11-08 - JSS`: Initial version.

1.444. updateMetadata

Full Name:	herschel.ia.toolbox.util.UpdateMetadataTask
Alias:	updateMetadata
Type:	Java Task -
Import:	from herschel.ia.toolbox.util import UpdateMetadataTask
Category	Arrays and datasets/Manipulation

Description

Updates metadata in a context hierarchy so that the changes are preserved in the context tree.

Example

Example 1: Modify the value of the "object" parameter within the "level1" metadata inside "obs"

```
updateMetadata(root=obs, path='.refs["level1"].product.meta',
               command='set("object",
 herschel.ia.dataset.StringParameter("Helix", "Target name"))')
```

API Summary

Jython Syntax

```
updateMetadata(&lt;root&gt;, &lt;path&gt;, &lt;command&gt;)
```

Properties

Product **root** [INPUT, MANDATORY, default=no default value]

String **path** [INPUT, MANDATORY, default=no default value]

String **command** [INPUT, MANDATORY, default=no default value]

API details

Properties

Product root [INPUT, MANDATORY, default=no default value]

Root product or context to modify.

String path [INPUT, MANDATORY, default=no default value]

Relative path from the root context to the dataset to be modified.

String command [INPUT, MANDATORY, default=no default value]

Jython command to update the metadata object.

See also

- Developers Manual: herschel.ia.toolbox.util.UpdateMetadataTask

History

- `2013-11-08 - JSS`: Initial version.

1.445. VARIANCE

Full Name:	herschel.ia.numeric.toolbox.basic.Variance
Alias:	VARIANCE
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import Variance
Category	Mathematics/Statistics

Description

Returns the variance of an array.

See this website for a definition of variance.

Returns NaN (Not a Number) if any of the array elements is NaN.

Examples

Example 1: Applying VARIANCE to a Float1d

```
x = Float1d([1,3,2,3,4])
print VARIANCE(x) # 1.3
x = Float1d([1,Double.NaN,2,3,4])
print VARIANCE(x) # NaN
# To remove NaN values (the original array is not modified):
print VARIANCE(NAN_FILTER(x)) # 1.6666666666666667 (VARIANCE is applied to
 [1,2,3,4])
```

Example 2: Applying Variance to a Float1d using a specified mean value

```
# Note the explicit import statement and the different
# name: Variance instead of VARIANCE
from herschel.ia.numeric.toolbox.basic import Variance
x = Float1d([1,3,2,3,4])
mean = 2.6
print Variance(mean)(x) # 1.3
```

API Summary

Jython Syntax
<y> = VARIANCE(<x>)

Properties
Array **x** [INPUT, MANDATORY, default=no default value]
Double **mean** [INPUT, OPTIONAL, default=no default value]
Double **y** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Array x [INPUT, MANDATORY, default=no default value]

The array of which to compute the variance. Complex arrays are not allowed.

Double mean [INPUT, OPTIONAL, default=no default value]
The mean value to use for computing the variance. See the second example for details.

Double y [OUTPUT, MANDATORY, default=no default value]
The variance of the input array.

See also

- MEAN
- MEDIAN
- STDDEV
- SKEWNESS
- KURTOSIS
- Developers Manual: herschel.ia.numeric.toolbox.basic.Variance

1.446. VoigtModel

Full Name:	herschel.ia.numeric.toolbox.fit.VoigtModel
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.fit import VoigtModel
Category	Mathematics/Fitting

Description

Voigt Model.

The Voigt function is a convolution of a Gauss and a Lorentz function. Physicaly it is the result of thermal and pressure broadening of a spectral line.

The models takes 4 parameters: amplitude, center frequency, half-width of the Gaussian, and half-width of the Lorentzian. These are initialised to 1, 0, 1, 1. Parameters 2 & 3 (widths) is always kept positive (>=0).

Example

Example 1: VoigtModel

```
voigt = VoigtModel()
voigt.setParameters( Double1d( [5, 4, 1, 2] ) )
print voigt( Double1d.range( 41 ) / 5 )      # from [0,8]
# ... fitter etc. see LevenbergMarquardtFitter
```

See also

- Developers Manual: herschel.ia.numeric.toolbox.fit.VoigtModel

1.447. voToTable

Full Name:	herschel.ia.toolbox.util.VoToTableTask
Alias:	voToTable
Type:	Java Task -
Import:	from herschel.ia.toolbox.util import VoToTableTask

Description

Task for reading VO files

Task for reading VO files. 'data' can be a file or a votable. If 'data' starts with '<VOTABLE' or '<? xml' it is xml.

Example

Example 1: Open a VOTable file and load it into a TableDataset

```
vizier_votable = voToTable(data="~/votables/vizier_votable.vot", name="my-
table")
```

API Summary

Jython Syntax

```
Full Signature:
table = VoToTableTask(data, name)
```

Properties

String **data** [INPUT, MANDATORY, default=no default value]

String **tableName** [INPUT, MANDATORY, default=no default value]

API details

Properties

String data [INPUT, MANDATORY, default=no default value]

Input data (xml text).

String tableName [INPUT, MANDATORY, default=no default value]

table dataset name.

See also

- Developers Manual: herschel.ia.toolbox.util.VoToTableTask

1.448. Wcs

Full Name:	herschel.ia.dataset.image.wcs.Wcs
Type:	Java Class -
Import:	from herschel.ia.dataset.image.wcs import Wcs
Categories	Images Data cubes

Description

A class to create a Wcs.

This class creates a Wcs. The Wcs describes the World coordinate system, which is used to convert between image coordinates and world coordinates.

Example

Example 1: A basic example on how to create a Wcs

```
wcs = Wcs(crval1=30.0, crval2=-89.50, crpix1=109, crpix2=109)
```

API Summary

Constructors
Wcs The default constructor.
Wcs (int naxis) Constructor for 2 or 3 dimensional Wcs.
Wcs (Wcs orig) The copy constructor.

Methods
Wcs **copy** Returns a copy of the Wcs object.
setParameter (String paramname, Object param, String description) Adds a new parameter to the Wcs.
isCompleteWcs Checks whether the Wcs is complete.
Object **getParameter** (String paramname) Returns a parameter from the Wcs.
Object[] **getParameters** Returns all parameters.
removeParameter (String paramname) Removes a parameter.
setNAxis (int naxis) Sets the number of axes.
int **getNAxis**

Methods
Returns the number of axes of the Wcs.
setImageIndex (Double1d index, Unit unit)
Sets the image index for a non-equidistant 3rd dimension.
boolean **hasImageIndex**
Checks the image index.
TableDataset **getImageIndex**
Returns the image index.
Double1d **getImageIndexArray**
Returns the image index.
boolean **isEquidistantInZ**
Checks whether the Wcs is equidistant in the 3rd dimension.
setCrval1 (double crval1)
Sets crval1.
double **getCrval1**
Returns crval1.
setCrval2 (double crval2)
Sets crval2.
double **getCrval2**
Returns crval2.
setCrval3 (double crval3)
Sets crval3.
double **getCrval3**
Returns crval3.
setCrpix1 (double crpix1)
Sets crpix1.
double **getCrpix1**
Returns crpix1.
setCrpix2 (double crpix2)
Sets crpix2.
double **getCrpix2**
Returns crpix2.
setCrpix3 (double crpix3)
Sets crpix3.
double **getCrpix3**
Returns crpix3.
setCdelt1 (double cdelt1)
Sets cdelt1.
double **getCdelt1**
Returns cdelt1.
setCdelt2 (double cdelt2)
Sets cdelt2.

Methods
double **getCdelt2** Returns cdelt2.
setCdelt3 (double cdelt3) Sets cdelt3.
double **getCdelt3** Returns cdelt3.
boolean **checkCtypeValidity** (String ctype) Checks the validity of the ctype parameter.
setCtype1 (String ctype1) Sets ctype1.
String **getCtype1** Returns ctype1.
setCtype2 (String ctype2) Sets ctype2.
String **getCtype2** Returns ctype2.
setCtype3 (String ctype3) Sets ctype3.
String **getCtype3** Returns ctype3.
setCdesc3 (String cdesc3) Sets cdesc3.
String **getCdesc3** Returns cdesc3.
setCunit1 (String cunit1) Sets cunit1.
String **getCunit1** Returns cunit1.
setCunit2 (String cunit2) Sets cunit2.
String **getCunit2** Returns cunit2.
setCunit3 (String cunit3) Sets cunit3.
String **getCunit3** Returns cunit3.
boolean **hasParameter** (String parameter) Checks the availability of a parameter.
setEpoch (double epoch) Sets the epoch.
double **getEpoch**

Methods
Returns the epoch.
setEquinox (double equinox) Sets the equinox.
double **getEquinox** Returns the equinox.
setRadesys (String Radesys) Sets the reference frame.
String **getRadesys** Returns the reference frame.
setCd1_1 (double cd1_1) Sets the cd1_1 element.
setCd1_2 (double cd1_2) Sets the cd1_2 element.
setCd1_3 (double cd1_3) Sets the cd1_3 element.
setCd2_1 (double cd2_1) Sets the cd2_1 element.
setCd2_2 (double cd2_2) Sets the cd2_2 element.
setCd2_3 (double cd2_3) Sets the cd2_3 element.
setCd3_1 (double cd31) Sets the cd3_1 element.
setCd3_2 (double cd3_2) Sets the cd3_2 element.
setCd3_3 (double cd3_3) Sets the cd3_3 element.
double **getCd1_1** Returns cd1_1.
double **getCd1_2** Returns cd1_2.
double **getCd2_1** Returns cd2_1.
double **getCd2_2** Returns cd2_2.
double **getCd1_3** Returns cd1_3.
double **getCd2_3** Returns cd2_3.
double **getCd3_1** Returns cd3_1.

Methods
double **getCd3_2** Returns cd3_2.
double **getCd3_3** Returns cd3_3.
setPc1_1 (double pc1_1) Sets the pc1_1 element.
double **getPc1_1** Returns pc1_1.
setPc1_2 (double pc1_2) Sets the pc1_2 element.
double **getPc1_2** Returns pc1_2.
setPc1_3 (double pc1_3) Sets the pc1_3 element.
double **getPc1_3** Returns pc1_3.
setPc2_1 (double pc2_1) Sets the pc2_1 element.
double **getPc2_1** Returns pc2_1.
setPc2_2 (double pc2_2) Sets the pc2_2 element.
double **getPc2_2** Returns pc2_2.
setPc2_3 (double pc2_3) Sets the pc2_3 element.
double **getPc2_3** Returns pc2_3.
setPc3_1 (double pc3_1) Sets the pc3_1 element.
double **getPc3_1** Returns pc3_1.
setPc3_2 (double pc3_2) Sets the pc3_2 element.
double **getPc3_2** Returns pc3_2.
setPc3_3 (double pc3_3) Sets the pc3_3 element.
double **getPc3_3** Returns pc3_3.
setProjection (String projection)

Methods
Sets the projection type.
String **getProjection** Returns the projection type.
MetaData **getMeta** Return the Wcs as metadata.
String **toString** Returns a string representation of the Wcs.
double[] **getWorldCoordinates** (double row, double column) Returns the world coordinates of the given image coordinates.
double[] **getWorldCoordinates** (double[] coords) Returns the world coordinates of the given image coordinates.
double **getZCoordinate** (int depth) Returns the world coordinates of the given layer.
double[] **getPixelCoordinates** (double c1, double c2) Returns the pixel coordinates.
double[] **getPixelCoordinates** (double[] c) Returns the pixel coordinates.
double[] **getPixelCoordinates** (Point2D.Double p, double[] coordinates) Returns the pixel coordinates.
setCrota2 (double crota2) Sets crota2.
setNaxis1 (int naxis1) Sets naxis1.
int **getNaxis1** Returns the number of pixels of the first axis.
setNaxis2 (int naxis2) Sets naxis2.
int **getNaxis2** Returns the number of pixels of the second axis of the Wcs.
setNaxis3 (int naxis3) Sets naxis3.
int **getNaxis3** Returns the number of pixels of the third axis of the Wcs.
double **getCrota2** Returns crota2.
boolean **isValid** Checks the possibility to convert from pixel to world coordinates.
setWavelength (double wavelength) Sets the wavelength in the Wcs.
getWavelength Returns the wavelength which is stored in the Wcs.

Methods
containsNorthCelestialPole Returns true when the north celestial pole is in the Wcs.
containsSouthCelestialPole Returns true when the south celestial pole is in the Wcs.
Type **getType** Returns the type for this Wcs.

API Details

Constructors

Wcs

The default constructor.

A constructor which creates a standard Wcs object. The standard Wcs sets naxis = 2 (NAXIS = 2)

Wcs (int naxis)

Constructor for 2 or 3 dimensional Wcs.

A constructor which creates a Wcs object with the chosen number of axes. The standard Wcs has only parameter naxis set

Argument

`int` **naxis** `[INPUT, MANDATORY, default=no default value]`

The number of axes (2 or 3), as int

Example

Typical example on how to create a Wcs.

```
wcs = Wcs(3)
wcs.setCrval1(30.0)
```

Wcs (Wcs orig)

The copy constructor.

Constructor for Wcs A constructor which creates a copy of an existing Wcs object.

Argument

`Wcs` **orig** `[INPUT, MANDATORY, default=no default value]`

The wcs to copy (2 or 3), as Wcs

Methods

***Wcs* copy**

Returns a copy of the Wcs object.

Return

Wcs

A copy of the Wcs.

setParameter (String paramname, Object param, String description)

Adds a new parameter to the Wcs.

setParameter (String paramname, Object param, String description)

Arguments

String **paramname** [INPUT, MANDATORY, default=no default value]

The name of the new new parameter, as String

Object **param** [INPUT, MANDATORY, default=no default value]

The new parameter to add to the wcs or the parameter to adopt, as Object

String **description** [INPUT, MANDATORY, default=no default value]

The description of the parameter, as String

isCompleteWcs

Checks whether the Wcs is complete.

Returns true if the Wcs has enough information to convert to and from World Coordinates.

***Object* getParameter (String paramname)**

Returns a parameter from the Wcs.

Returns the requested parameter of the Wcs.

Argument

String **paramname** [INPUT, MANDATORY, default=no default value]

The requested parameter of the Wcs, as String

Return

Object

The requested parameter.

***Object[]* getParameters**

Returns all parameters.

Returns the list of all Wcs parameters.

Return

Object[]

The list of all Wcs parameters.

removeParameter (String paramname)

Removes a parameter.

Removes the requested parameter of the Wcs.

Argument

String **paramname** [INPUT, MANDATORY, default=no default value]

The parameter to remove, as String

setNAxis (int naxis)

Sets the number of axes.

Sets the number of axes of the Wcs.

Argument

int **naxis** [INPUT, MANDATORY, default=no default value]

setNAxis (int naxis)

The number of Axes, as int

***int* getNAxis**

Returns the number of axes of the Wcs.

Return

int

The number of axes.

setImageIndex (Double1d index, Unit unit)

Sets the image index for a non-equidistant 3rd dimension.

Arguments

`Double1d` **index** `[INPUT, MANDATORY, default=no default value]`

A Double1d describing the depth dimension values for every Layer, as Double1d

`Unit` **unit** `[INPUT, MANDATORY, default=no default value]`

The unit of the depth dimension, as Unit

***boolean* hasImageIndex**

Checks the image index.

Returns true if there is an image index.

Return

boolean

True if there is an image index.

***TableDataset* getImageIndex**

Returns the image index.

Returns the image index for a non-equidistant 3rd dimension.

Return

TableDataset

The image index for a non-equidistant 3rd dimension.

***Double1d* getImageIndexArray**

Returns the image index.

Returns the image index for a non-equidistant 3rd dimension.

Return

Double1d

The image index for a non-equidistant 3rd dimension.

***boolean* isEquidistantInZ**

Checks whether the Wcs is equidistant in the 3rd dimension.

isEquidistantInZ returns true when the 3rd axis is equidistant. In this case, the crval3, crpix3 and cdelt3 keywords are used. If the 3rd axis is not equidistant, the ImageIndex is used.

Return

boolean isEquidistantInZ

boolean

True if the Wcs is equidistant.

setCrval1 (double crval1)

Sets crval1.

Sets the first coordinate of the center.

Argument

`double` **`crval1`** `[INPUT, MANDATORY, default=no default value]`

The first coordinate of the reference pixel, as double

double getCrval1

Returns crval1.

Returns the first coordinate of the reference pixel.

Return

double

The first coordinate of the reference pixel.

setCrval2 (double crval2)

Sets crval2.

Sets the second coordinate of the center.

Argument

`double` **`crval2`** `[INPUT, MANDATORY, default=no default value]`

The second coordinate of the reference pixel, as double

double getCrval2

Returns crval2.

Returns the second coordinate of the reference pixel.

Return

double

The second coordinate of the reference pixel.

setCrval3 (double crval3)

Sets crval3.

Sets the third coordinate of the center.

Argument

`double` **`crval3`** `[INPUT, MANDATORY, default=no default value]`

The 3rd coordinate of the reference pixel (in degrees), as double

double getCrval3

Returns crval3.

Returns the third coordinate of the reference pixel.

double getCrval3

Return

double

The third coordinate of the reference pixel.

setCrpix1 (double crpix1)

Sets crpix1.

Sets the reference pixel position of axis 1. WARNING: CRPIX1 should be given in x,y coordinates. The standard specifies that the image starts at pixel (1,1) and not at pixel (0,0). So to set the first pixel, you should use 1 as value for crpix1.

Argument

`double` **`crpix1`** `[INPUT, MANDATORY, default=no default value]`

The reference pixel position of axis 1, as double

double getCrpix1

Returns crpix1.

Returns the reference pixel position of axis 1.

Return

double

The reference pixel position of axis 1.

setCrpix2 (double crpix2)

Sets crpix2.

Sets the reference pixel position of axis 2. WARNING: CRPIX2 should be given in x,y coordinates. The standard specifies that the image starts at pixel (1,1) and not at pixel (0,0). So to set the first pixel, you should use 1 as value for crpix2.

Argument

`double` **`crpix2`** `[INPUT, MANDATORY, default=no default value]`

The reference pixel position of axis 2, as double

double getCrpix2

Returns crpix2.

Returns the reference pixel position of axis 2.

Return

double

The reference pixel position of axis 2.

setCrpix3 (double crpix3)

Sets crpix3.

Sets the reference pixel position of axis 3.

Argument

`double` **`crpix3`** `[INPUT, MANDATORY, default=no default value]`

he reference layer index, as double

***double* getCrpix3**

Returns crpix3.

Returns the reference pixel position of axis 3.

Return

double

The reference pixel position of axis 3.

setCdelt1 (double cdelt1)

Sets cdelt1.

Sets the pixel scale of axis 1.

Argument

`double` **`cdelt1`** `[INPUT, MANDATORY, default=no default value]`

The pixel scale of axis 1, as double

***double* getCdelt1**

Returns cdelt1.

Returns the pixel scale of axis 1.

Return

double

The pixel scale of axis 1.

setCdelt2 (double cdelt2)

Sets cdelt2.

Sets the pixel scale of axis 2.

Argument

`double` **`cdelt2`** `[INPUT, MANDATORY, default=no default value]`

The pixel scale of axis 2, as double

***double* getCdelt2**

Returns cdelt2.

Returns the pixel scale of axis 2.

Return

double

The pixel scale of axis 2.

setCdelt3 (double cdelt3)

Sets cdelt3.

Sets the pixel scale of axis 3.

Argument

`double` **`cdelt3`** `[INPUT, MANDATORY, default=no default value]`

The scale in 3rd dimension, as double

***double* getCdelt3**

Returns cdelt3.

Returns the pixel scale of axis 3.

Return

double

The pixel scale of axis 3.

***boolean* checkCtypeValidity (String ctype)**

Checks the validity of the ctype parameter.

Argument

`String` **ctype** `[INPUT, MANDATORY, default=no default value]`

The parameter to check, as String

Return

boolean

true if the ctype parameter is valid.

setCtype1 (String ctype1)

Sets ctype1.

Sets the projection type of axis 1.

Argument

`String` **ctype1** `[INPUT, MANDATORY, default=no default value]`

The projection type of axis 1, as String

***String* getCtype1**

Returns ctype1.

Returns the projection type of axis 1.

Return

String

The projection type of axis 1.

setCtype2 (String ctype2)

Sets ctype2.

Sets the projection type of axis 2.

Argument

`String` **ctype2** `[INPUT, MANDATORY, default=no default value]`

The projection type of axis 2, as String

***String* getCtype2**

Returns ctype2.

Returns the projection type of axis 2.

Return

String

String **getCtype2**

The projection type of axis 2.

setCtype3 (String ctype3)

Sets ctype3.

Sets the projection type of axis 3.

Argument

`String` **ctype3** `[INPUT, MANDATORY, default=no default value]`

Describes what the 3rd axis represents, as String

String **getCtype3**

Returns ctype3.

Returns the projection type of axis 3.

Return

String

The projection type of axis 3.

setCdesc3 (String cdesc3)

Sets cdesc3.

Sets the description of axis 3.

Argument

`String` **cdesc3** `[INPUT, MANDATORY, default=no default value]`

Describes what the 3rd axis represents, as String

String **getCdesc3**

Returns cdesc3.

Returns what axis 3 represents.

Return

String

The description what axis 3 represents.

setCunit1 (String cunit1)

Sets cunit1.

Sets the unit of axis 1.

Argument

`String` **cunit1** `[INPUT, MANDATORY, default=no default value]`

The unit of axis 1, as String

String **getCunit1**

Returns cunit1.

Returns the unit of axis 1.

Return

String

String **getCunit1**

The unit of axis 1.

setCunit2 (String cunit2)

Sets cunit2.

Sets the unit of axis 2.

Argument

`String` **cunit2** `[INPUT, MANDATORY, default=no default value]`

The unit of axis 2, as String

String **getCunit2**

Returns cunit2.

Returns the unit of axis 2.

Return

String

The unit of axis 2.

setCunit3 (String cunit3)

Sets cunit3.

Sets the unit of axis 3.

Argument

`String` **cunit3** `[INPUT, MANDATORY, default=no default value]`

The unit of axis 3, as String

String **getCunit3**

Returns cunit3.

Returns the unit of axis 3.

Return

String

The unit of axis 3.

boolean **hasParameter (String parameter)**

Checks the availability of a parameter.

Argument

`String` **parameter** `[INPUT, MANDATORY, default=no default value]`

The parameter to check, as String

Return

boolean

True if the parameter is available.

setEpoch (double epoch)

Sets the epoch.

Argument

setEpoch (double epoch)

`double` **epoch** `[INPUT, MANDATORY, default=no default value]`

The epoch, as double

double **getEpoch**

Returns the epoch.

Return

double

The epoch.

setEquinox (double equinox)

Sets the equinox.

Argument

`double` **equinox** `[INPUT, MANDATORY, default=no default value]`

The equinox, as double

double **getEquinox**

Returns the equinox.

Return

double

The equinox.

setRadesys (String Radesys)

Sets the reference frame.

Argument

`String` **Radesys** `[INPUT, MANDATORY, default=no default value]`

The reference frame, as String

String **getRadesys**

Returns the reference frame.

Return

String

The reference frame.

setCd1_1 (double cd1_1)

Sets the cd1_1 element.

Sets element (1, 1) of the corrected CD matrix CDi_j..

Argument

`double` **cd1_1** `[INPUT, MANDATORY, default=no default value]`

Element (1,1) of the corrected CD matrix CDi_j, as double

setCd1_2 (double cd1_2)

Sets the cd1_2 element.

Sets element (1, 2) of the corrected CD matrix CDi_j..

setCd1_2 (double cd1_2)

Argument

`double` **`cd1_2`** `[INPUT, MANDATORY, default=no default value]`

Element (1,2) of the corrected CD matrix CDi_j, as double

setCd1_3 (double cd1_3)

Sets the cd1_3 element.

Sets element (1, 3) of the corrected CD matrix CDi_j..

Argument

`double` **`cd1_3`** `[INPUT, MANDATORY, default=no default value]`

Element (1,3) of the corrected CD matrix CDi_j, as double

setCd2_1 (double cd2_1)

Sets the cd2_1 element.

Sets element (2, 1) of the corrected CD matrix CDi_j..

Argument

`double` **`cd2_1`** `[INPUT, MANDATORY, default=no default value]`

Element (2,1) of the corrected CD matrix CDi_j, as double

setCd2_2 (double cd2_2)

Sets the cd2_2 element.

Sets element (2, 2) of the corrected CD matrix CDi_j..

Argument

`double` **`cd2_2`** `[INPUT, MANDATORY, default=no default value]`

Element (2,2) of the corrected CD matrix CDi_j, as double

setCd2_3 (double cd2_3)

Sets the cd2_3 element.

Sets element (2, 3) of the corrected CD matrix CDi_j..

Argument

`double` **`cd2_3`** `[INPUT, MANDATORY, default=no default value]`

Element (2,3) of the corrected CD matrix CDi_j, as double

setCd3_1 (double cd31)

Sets the cd3_1 element.

Sets element (3, 1) of the corrected CD matrix CDi_j..

Argument

`double` **`cd31`** `[INPUT, MANDATORY]`

setCd3_2 (double cd3_2)

Sets the cd3_2 element.

Sets element (3, 2) of the corrected CD matrix CDi_j..

Argument

setCd3_2 (double cd3_2)

double **cd3_2** [INPUT, MANDATORY, default=no default value]

Element (3,2) of the corrected CD matrix CDi_j, as double

setCd3_3 (double cd3_3)

Sets the cd3_3 element.

Sets element (3, 3) of the corrected CD matrix CDi_j..

Argument

double **cd3_3** [INPUT, MANDATORY, default=no default value]

Element (3,3) of the corrected CD matrix CDi_j, as double

***double* getCd1_1**

Returns cd1_1.

Returns element (1,1) of the corrected CD matrix CDi_j.

Return

double

Element (1,1) of the corrected CD matrix CDi_j.

***double* getCd1_2**

Returns cd1_2.

Returns element (1,2) of the corrected CD matrix CDi_j.

Return

double

Element (1,2) of the corrected CD matrix CDi_j.

***double* getCd2_1**

Returns cd2_1.

Returns element (2,1) of the corrected CD matrix CDi_j.

Return

double

Element (2,1) of the corrected CD matrix CDi_j.

***double* getCd2_2**

Returns cd2_2.

Returns element (2,2) of the corrected CD matrix CDi_j.

Return

double

Element (2,2) of the corrected CD matrix CDi_j.

***double* getCd1_3**

Returns cd1_3.

Returns element (1,3) of the corrected CD matrix CDi_j.

***double* getCd1_3**

Return

double

Element (1,3) of the corrected CD matrix CDi_j.

***double* getCd2_3**

Returns cd2_3.

Returns element (2,3) of the corrected CD matrix CDi_j.

Return

double

Element (2,3) of the corrected CD matrix CDi_j.

***double* getCd3_1**

Returns cd3_1.

Returns element (3,1) of the corrected CD matrix CDi_j.

Return

double

Element (3,1) of the corrected CD matrix CDi_j.

***double* getCd3_2**

Returns cd3_2.

Returns element (3,2) of the corrected CD matrix CDi_j.

Return

double

Element (3,2) of the corrected CD matrix CDi_j.

***double* getCd3_3**

Returns cd3_3.

Returns element (3,3) of the corrected CD matrix CDi_j.

Return

double

Element (3,3) of the corrected CD matrix CDi_j.

setPc1_1 (double pc1_1)

Sets the pc1_1 element.

Sets element (1, 1) of the linear transformation matrix PCi_j..

Argument

`double` **`pc1_1`** `[INPUT, MANDATORY, default=no default value]`

Element (1,1) of the linear transformation matrix PCi_j, as double

***double* getPc1_1**

Returns pc1_1.

***double* getPc1_1**

Returns element (1,1) of the linear transformation matrix PCi_j.

Return

double

Element (1,1) of the linear transformation matrix PCi_j.

setPc1_2 (double pc1_2)

Sets the pc1_2 element.

Sets element (1, 2) of the linear transformation matrix PCi_j..

Argument

`double` **`pc1_2`** `[INPUT, MANDATORY, default=no default value]`

Element (1,2) of the linear transformation matrix PCi_j, as double

***double* getPc1_2**

Returns pc1_2.

Returns element (1,2) of the linear transformation matrix PCi_j.

Return

double

Element (1,2) of the linear transformation matrix PCi_j.

setPc1_3 (double pc1_3)

Sets the pc1_3 element.

Sets element (1, 3) of the linear transformation matrix PCi_j..

Argument

`double` **`pc1_3`** `[INPUT, MANDATORY, default=no default value]`

Element (1,3) of the linear transformation matrix PCi_j, as double

***double* getPc1_3**

Returns pc1_3.

Returns element (1,3) of the linear transformation matrix PCi_j.

Return

double

Element (1,3) of the linear transformation matrix PCi_j.

setPc2_1 (double pc2_1)

Sets the pc2_1 element.

Sets element (2, 1) of the linear transformation matrix PCi_j..

Argument

`double` **`pc2_1`** `[INPUT, MANDATORY, default=no default value]`

Element (2,1) of the linear transformation matrix PCi_j, as double

***double* getPc2_1**

Returns pc2_1.

double getPc2_1

Returns element (2,1) of the linear transformation matrix PCi_j.

Return

double

Element (2,1) of the linear transformation matrix PCi_j.

setPc2_2 (double pc2_2)

Sets the pc2_2 element.

Sets element (2, 2) of the linear transformation matrix PCi_j..

Argument

double **pc2_2** [INPUT, MANDATORY, default=no default value]

Element (2,2) of the linear transformation matrix PCi_j, as double

double getPc2_2

Returns pc2_2.

Returns element (2,2) of the linear transformation matrix PCi_j.

Return

double

Element (2,2) of the linear transformation matrix PCi_j.

setPc2_3 (double pc2_3)

Sets the pc2_3 element.

Sets element (2, 3) of the linear transformation matrix PCi_j..

Argument

double **pc2_3** [INPUT, MANDATORY, default=no default value]

Element (2,3) of the linear transformation matrix PCi_j, as double

double getPc2_3

Returns pc2_3.

Returns element (2,3) of the linear transformation matrix PCi_j.

Return

double

Element (2,3) of the linear transformation matrix PCi_j.

setPc3_1 (double pc3_1)

Sets the pc3_1 element.

Sets element (3, 1) of the linear transformation matrix PCi_j..

Argument

double **pc3_1** [INPUT, MANDATORY, default=no default value]

Element (3,1) of the linear transformation matrix PCi_j, as double

double getPc3_1

Returns pc3_1.

***double* getPc3_1**

Returns element (3,1) of the linear transformation matrix PCi_j.

Return

double

Element (3,1) of the linear transformation matrix PCi_j.

setPc3_2 (double pc3_2)

Sets the pc3_2 element.

Sets element (3, 2) of the linear transformation matrix PCi_j..

Argument

`double` **`pc3_2`** `[INPUT, MANDATORY, default=no default value]`

Element (3,2) of the linear transformation matrix PCi_j, as double

***double* getPc3_2**

Returns pc3_2.

Returns element (3,2) of the linear transformation matrix PCi_j.

Return

double

Element (3,2) of the linear transformation matrix PCi_j.

setPc3_3 (double pc3_3)

Sets the pc3_3 element.

Sets element (3, 3) of the linear transformation matrix PCi_j..

Argument

`double` **`pc3_3`** `[INPUT, MANDATORY, default=no default value]`

Element (3,3) of the linear transformation matrix PCi_j, as double

***double* getPc3_3**

Returns pc3_3.

Returns element (3,3) of the linear transformation matrix PCi_j.

Return

double

Element (3,3) of the linear transformation matrix PCi_j.

setProjection (String projection)

Sets the projection type.

Argument

`String` **`projection`** `[INPUT, MANDATORY, default=no default value]`

A String describing the projection, as String

***String* getProjection**

Returns the projection type.

Return

String getProjection

String

The projection type.

MetaData getMeta

Return the Wcs as metadata.

Return

MetaData

The Wcs as metadata.

String toString

Returns a string representation of the Wcs.

Returns a string representation of the values of the Wcs object. The format of this string is undefined and subject to change.

Return

String

a string representation of the values of the Wcs object.

double[] getWorldCoordinates (double row, double column)

Returns the world coordinates of the given image coordinates.

Arguments

`double` **`row`** `[INPUT, MANDATORY, default=no default value]`

The row of the image, as double

`double` **`column`** `[INPUT, MANDATORY, default=no default value]`

The column of the image, as double

Return

double[]

The corresponding world coordinates as a 2 dimensional array. When the ctype1 and ctype2 keywords of the wcs are defined as "RA---TAN" and "DEC--TAN", the first coordinates describes the right ascension and the second the declination.

double[] getWorldCoordinates (double[] coords)

Returns the world coordinates of the given image coordinates.

Argument

`double[]` **`coords`** `[INPUT, MANDATORY, default=no default value]`

The pixel coordinates (row, column) of the image, as double[]

Return

double[]

The corresponding world coordinates as a 2 dimensional array. When the ctype1 and ctype2 keywords of the wcs are defined as "RA---TAN" and "DEC--TAN", the first coordinates describes the right ascension and the second the declination.

double getZCoordinate (int depth)

Returns the world coordinates of the given layer.

double getZCoordinate (int depth)

Argument

`int` **depth** `[INPUT, MANDATORY, default=no default value]`

The layer of the cube, as double

Return

double

The corresponding world coordinates of the Z axis.

double[] getPixelCoordinates (double c1, double c2)

Returns the pixel coordinates.

Returns the pixelCoordinates of the given SkyCoordinates. The pixel-coordinates are the row and the column of the image and are counted from 0, unlike the Wcs CRPIXx and FITS header keywords convention, which assume 1 as the first pixel of the image.

Arguments

`double` **c1** `[INPUT, MANDATORY, default=no default value]`

The world coordinate (right ascension in case of equatorial coordinates) for which the x-coordinate should be calculated, as double

`double` **c2** `[INPUT, MANDATORY, default=no default value]`

The world coordinate (right ascension in case of equatorial coordinates) for which the y-coordinate should be calculated, as double

Return

double[]

A 2-dimensional array of doubles describing the pixel coordinates of the given world coordinates.

double[] getPixelCoordinates (double[] c)

Returns the pixel coordinates.

Returns the pixelCoordinates of the given SkyCoordinates. The pixel-coordinates are the row and the column of the image and are counted from 0, unlike the Wcs CRPIXx and FITS header keywords convention, which assume 1 as the first pixel of the image.

Argument

`double[]` **c** `[INPUT, MANDATORY, default=no default value]`

The world coordinates (right ascension and declination in case of equatorial coordinates) for which the x-coordinate should be calculated, as double[]

Return

double[]

A 2-dimensional array of doubles describing the pixel coordinates of the given world coordinates.

double[] getPixelCoordinates (Point2D.Double p, double[] coordinates)

Returns the pixel coordinates.

Returns the pixelCoordinates of the given SkyCoordinates. The pixel-coordinates are the row and the column of the image and are counted from 0, unlike the Wcs CRPIXx and FITS header keywords convention, which assume 1 as the first pixel of the image.

double[] getPixelCoordinates (Point2D.Double p, double[] coordinates)

Arguments

Point2D.Double **p** [INPUT, MANDATORY, default=no default value]

a Point2d.Double that contains the world coordinate (right ascension in case of equatorial coordinates) for which the x-coordinate should be calculated and the world coordinate(declination in case of equatorial coordinates) for which the y-coordinate should be calculated, as Point2D.Double

double[] **coordinates** [INPUT, MANDATORY, default=no default value]

An empty double[2], that is filled with the coordinate values and returned, as double[]

Return

double[]

A 2-dimensional array of doubles describing the pixel coordinates of the given world coordinates.

setCrota2 (double crota2)

Sets crota2.

Sets the rotation angle in degrees.

Argument

double **crota2** [INPUT, MANDATORY, default=no default value]

The rotation angle in degrees, as double

setNaxis1 (int naxis1)

Sets naxis1.

Sets the number of pixels of the first axis.

Argument

int **naxis1** [INPUT, MANDATORY, default=no default value]

The number of pixels of the first axis, as int

int getNaxis1

Returns the number of pixels of the first axis.

Return

int

The number of pixels of the first axis.

setNaxis2 (int naxis2)

Sets naxis2.

Sets the number of pixels of the second axis.

Argument

int **naxis2** [INPUT, MANDATORY, default=no default value]

The number of pixels of the second axis, as int

int getNaxis2

Returns the number of pixels of the second axis of the Wcs.

int getNaxis2

Return

int

The number of pixels of the second axis.

setNaxis3 (int naxis3)

Sets naxis3.

Sets the number of pixels of the third axis.

Argument

`int` **naxis3** `[INPUT, MANDATORY, default=no default value]`

The number of pixels of the third axis, as int

int getNaxis3

Returns the number of pixels of the third axis of the Wcs.

Return

int

The number of pixels of the third axis.

double getCrota2

Returns crota2.

Returns the rotation angle in degrees.

Return

double

The rotation angle in degrees.

boolean isValid

Checks the possibility to convert from pixel to world coordinates.

Checks if the conversion from pixel to world coordinates is possible and returns true if this is the case.

Return

boolean

true is the conversion from pixel to world coordinates is possible.

setWavelength (double wavelength)

Sets the wavelength in the Wcs.

Sets the wavelength in the Wcs (in micrometers). The WAVELNTH and the WAVEUNIT parameter are set.

Argument

`double` **wavelength** `[INPUT, MANDATORY, default=no default value]`

The wavelength to set (in micrometers), as double

getWavelength

Returns the wavelength which is stored in the Wcs.

getWavelength

Returns the wavelength in micrometer.

containsNorthCelestialPole

Returns true when the north celestial pole is in the Wcs.

containsSouthCelestialPole

Returns true when the south celestial pole is in the Wcs.

Type **getType**

Returns the type for this Wcs.

This is either equatorial, galactic, ecliptic, or unknown.

Return

Type

The type for this Wcs. This is either equatorial, galactic, ecliptic, or unknown.

See also

- Developers Manual: herschel.ia.dataset.image.wcs.Wcs

1.449. WeightedMean

Full Name:	herschel.ia.numeric.toolbox.basic.WeightedMean
Type:	Java Class -
Import:	from herschel.ia.numeric.toolbox.basic import WeightedMean
Category	Mathematics/Statistics

Description

Returns the quadratically weighted mean of an array.

The function returns the error of the mean based on the errors of the input data, assuming that the distribution of the errors is Gaussian.

The mean is calculated with the formula *wmean = SUM(x[i]/(dx[i]^2)) / sum(1/(dx[i]^2))*, where *x* is the input array, *dx* the array of errors and *SUM* is the sum over all array elements.

The following error measures are available. See the example for how to retrieve them.

- Standard deviation of the distribution:

  errorDistr = SQRT(SUM((x[i]-wmean)^2/(dx[i]^2)) / SUM(1/(dx[i]^2)))

- Standard deviation of the mean:

  errorMean = SQRT((1/SUM(1/dx[i]^2)) * (1/(N-1)) * SUM((x[i]-wmean)^2/(dx[i]^2)))

- Standard deviation of the mean for trusted errors:

  errorMeanTrusted = SQRT(1/SUM(1/dx[i]^2))

Example

Example 1: Apply to Double1d and Complex1d arrays

```
values = Double1d([1,3,5])
errors = Double1d([0.1,0.2,0.3])
wmean = WeightedMean(values,errors)
print wmean.mean()  # Quadratically weighted mean
print wmean.errorDistr()  # Standard deviation of the distribution
print wmean.errorMean()  # Standard deviation of the mean
print wmean.errorMeanTrusted()  # Standard deviation of the mean for trusted
 errors
values = Complex1d([1+1j,3+1j,5+1j])
errors = Complex1d([0.1+1j,0.2+1j,0.3+1j])
wmean = WeightedMean(values,errors)
print wmean.mean()
print wmean.errorDistr()
print wmean.errorMean()
print wmean.errorMeanTrusted()
```

API Summary

Jython Syntax

```
wm = WeightedMean(&lt;values&gt;, &lt;errors&gt; [,&lt;ignore-
NaN&gt;])
wm.mean()
```

Jython Syntax
`wm.errorDistr()`
`wm.errorMean()`
`wm.errorMeanTrusted()`

Properties
`1-D array` **`values`** `[INPUT, MANDATORY, default=no default value]`
`1-D array` **`errors`** `[INPUT, MANDATORY, default=no default value]`
`Boolean` **`ignoreNaN`** `[INPUT, OPTIONAL, default=False]`

API details

Properties

1-D array values [INPUT, MANDATORY, default=no default value]
The array of which to compute the weighted mean. Complex arrays are allowed.

1-D array errors [INPUT, MANDATORY, default=no default value]
The errors of the input array values.

Boolean ignoreNaN [INPUT, OPTIONAL, default=False]
If *False*, the function returns NaN if it finds NaN values in the input data. If *True*, NaN values are ignored.

See also

- Developers Manual: herschel.ia.numeric.toolbox.basic.WeightedMean

1.450. xyz2raDecRoll

Full Name:	herschel.ia.toolbox.pointing.xyz2raDecRoll
Type:	Jython Task -
Import:	from herschel.ia.toolbox.pointing import xyz2raDecRoll
Category	Toolboxes/Pointing

Description

Function to go from difference in spacecraft x,y,z to ra/dec/roll.

Example

Example 1: ra,dec,roll = xyz2raDecRoll(1.,1.,1.,0.,0.,0.,2.,2.,2.)

```
ra,dec,roll = xyz2raDecRoll(90.,90.,45.,0.,0.,0., 0.,0.,0.,degrees=True)
x= Double1d([45., 90.])
y = Double1d([90., 45.])
z = Double1d([90., 90.])
ra,dec,roll = xyz2raDecRoll(x,y,z,0.,0.,0., 0.,0.,0., degrees=True)
```

API Summary

Jython Syntax

```
ra,dec,roll = xyz2raDecRoll(x,y,z,xref,yref,zref,refra,refdec,refroll,degrees=False|True)
```

Properties
Double1d **x** [INPUT, MANDATORY, default=NO default value]
Double1d **y** [INPUT, MANDATORY, default=NO default value]
Double1d **z** [INPUT, MANDATORY, default=NO default value]
Double1d **xref** [INPUT, MANDATORY, default=NO default value]
Double1d **yref** [INPUT, MANDATORY, default=NO default value]
Double1d **zref** [INPUT, MANDATORY, default=NO default value]
double/Double1d **refRa** [INPUT, MANDATORY, default=NO default value]
double/Double1d **refDec** [INPUT, MANDATORY, default=NO default value]
double/Double1d **refRoll** [INPUT, MANDATORY, default=NO default value]
Boolean **degrees** [INPUT, OPTIONAL, default=Default False]
double/Double1d **ra** [OUTPUT, MANDATORY, default=NO default value]
double/Double1d **dec** [OUTPUT, MANDATORY, default=NO default value]
double/Double1d **roll** [OUTPUT, MANDATORY, default=NO default value]

API details

Properties

Double1d x [INPUT, MANDATORY, default=NO default value]

rotation angles over x spacraft axis (degrees or radians)

Double1d y [INPUT, MANDATORY, default=NO default value]

rotation angles over y spacraft axis (degrees or radians)

Double1d z [INPUT, MANDATORY, default=NO default value]

rotation angles over z spacraft axis (degrees or radians)

Double1d xref [INPUT, MANDATORY, default=NO default value]

reference rotation angles over x spacraft axis (degrees or radians)

Double1d yref [INPUT, MANDATORY, default=NO default value]

reference rotation angles over y spacraft axis (degrees or radians)

Double1d zref [INPUT, MANDATORY, default=NO default value]

reference rotation angles over z spacraft axis (degrees or radians)

double/Double1d refRa [INPUT, MANDATORY, default=NO default value]

reference position right Ascension (degrees or radians)

double/Double1d refDec [INPUT, MANDATORY, default=NO default value]

reference position declination (degrees or radians)

double/Double1d refRoll [INPUT, MANDATORY, default=NO default value]

reference position declination (degrees or radians)

Boolean degrees [INPUT, OPTIONAL, default=Default False]

input and output angles in degrees; if False: input / output in radians

double/Double1d ra [OUTPUT, MANDATORY, default=NO default value]

right Ascension (degrees or radians)

double/Double1d dec [OUTPUT, MANDATORY, default=NO default value]

declination (degrees or radians)

double/Double1d roll [OUTPUT, MANDATORY, default=NO default value]

declination (degrees or radians)

History

- 2013-04-22 - BV: Initial version
- 2013-08-24 - BV: HCSS-18486 Invert orientation of dx,dy,dz vector
- 2013-09-26 - BV: HCSS-18567 Review disabled jexamples