pydl Documentation

Release 0.7.0

Benjamin Alan Weaver

Feb 23, 2019

Contents

Part I

Introduction

This package consists of Python replacements for functions that are part of the IDL® built-in library or part of astronomical IDL® libraries. The emphasis is on reproducing results of the astronomical library functions. Only the bare minimum of IDL® built-in functions are implemented to support this.

There are four astronomical libraries targeted:

- idlutils : a general suite of tools heavily used by SDSS.
- Goddard utilities : The IDL® Astronomy User's Libary, maintained by Wayne Landsman and distributed with idlutils.
- idlspec2d : tools for working with SDSS, BOSS and eBOSS spectroscopic data.
- photoop : tools for working with SDSS imaging data.

This package affiliated with the astropy project and is registered with PyPI.

Part II

Components

Most of the functionality of PyDL is in sub-packages.

CHAPTER 1

SDSS Utilities (`pydl.pydlutils`)

1.1 Introduction

This package provides functionality similar to idlutils, a general suite of tools heavily used by SDSS.

idlutils is itself divided into a number of subpackages. Below we list the subpackages and the usability of the PyDL equivalent. The readiness levels are defined as:

Obsolete
No point in implementing because the purpose of the code lapsed many years ago.

Not Applicable (NA)
No point in implementing because another built-in or numpy/scipy/astropy package completely replaces this.

None
Not (yet) implemented at all.

Rudimentary
Only a few functions are implemented.

Fair
Enough functions are implemented to be useful, but some are missing.

Good
Pretty much anything you could do with the idlutils code you can do with the equivalent here.

Subpackage	Readiness Level	Comments
2mass	None	For use with matching 2MASS catalogs to SDSS data.
astrom	None	For use with SDSS astrometric data structures. Largely superseded by WCS.
bspline	Good	Fitting B-splines to data, especially for resampling.
cooling	Good	See `pydl.pydlutils.cooling.read_ds_cooling()`.
coord	Fair	Some functionality already provided by `astropy.coordinates`.
cosmography	NA	Tools for computing lookback time, angular sizes at cosmological distances, etc. Use `astropy.`
dimage	None	Interface to C code used for sky subtraction.

Continued

Table 1 – continued from previous page

Subpackage	Readiness Level	Comments
djsphot	None	A simple aperture photometry code.
dust	None	For use with the SFD galactic dust map.
first	None	For use with matching FIRST catalogs to SDSS data.
fits	NA	Use `astropy.io.fits`.
healpix	NA	Interact with HEALPix data. Use healpy.
image	Rudimentary	Image manipulation functions.
json	NA	Use `json` or other packages.
mangle	Fair	Some work still required on polygon area calculations.
math	Fair	Generic mathematical functions. Many are implemented in numpy or scipy.
mcmc	None	But there are plenty of good Python MCMC packages out there.
mglib	Obsolete	An IDL object-oriented configuration file reader.
misc	Fair	General purpose utility functions.
mpeg	None	Wrapper for **ppmtompeg**, makes movies from data.
mpfit	None	Appears to be an out-of-sync copy of the "markwardt" package in the The IDL® Astronomy Us
physics	None	Implementation of physical formulas, *e.g.* free-free scattering.
plot	None	Much functionality already exists in matplotlib.
psf	Obsolete	Point-spread function fitting.
rgbcolor	Good	Some functionality is duplicated in `astropy.visualization`, especially `make_lupton_rgb()`.
rosat	None	For use with matching ROSAT catalogs to SDSS data.
sdss	Good	Most important functionalities are bitmasks and reading sweep files.
slatec	None	Fit B-splines using C code.
spheregroup	Good	Used for matching arbitrary RA, Dec coordinates to other arbitrary RA, Dec coordinates.
TeXtoIDL	NA	This package is for including TeX in IDL plots. Since matplotlib understands TeX natively, this
trace	Fair	Used for fitting orthogonal functions to spectroscopic wavelength solutions.
ukidss	None	Used for matching UKIDSS catalogs to SDSS data.
wise	None	Used for matching WISE catalogs to SDSS data.
yanny	Good	Tools for manipulating SDSS parameter files.

1.2 API

1.2.1 pydl.pydlutils Package

This subpackage implements functions from the idlutils package.

Classes

`PydlutilsException`	Exceptions raised by `pydl.pydlutils` that don't fit into a standard exception class like `ValueError`.
`PydlutilsUserWarning`	Class for warnings issued by `pydl.pydlutils`.

PydlutilsException

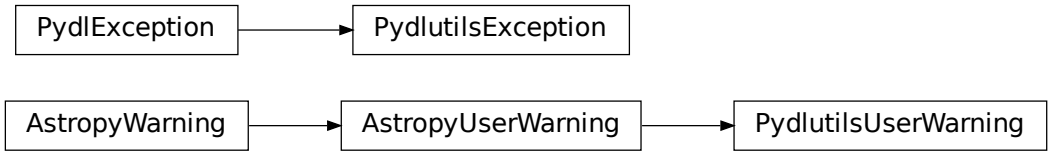
exception pydl.pydlutils.**PydlutilsException**

Exceptions raised by `pydl.pydlutils` that don't fit into a standard exception class like `ValueError`.

PydlutilsUserWarning

exception pydl.pydlutils.**PydlutilsUserWarning**
Class for warnings issued by pydl.pydlutils.

Class Inheritance Diagram



1.2.2 pydl.pydlutils.bspline Module

This module corresponds to the bspline directory in idlutils.

Functions

cholesky_band(l[, mininf])	Compute *lower* Cholesky decomposition of a banded matrix.
cholesky_solve(a, bb)	Solve the equation $Ax = b$ where a is a *lower* Cholesky-banded matrix.
iterfit(xdata, ydata[, invvar, upper, ...])	Iteratively fit a B-spline set to data, with rejection.

cholesky_band

pydl.pydlutils.bspline.**cholesky_band**(*l*, *mininf=0.0*)
Compute *lower* Cholesky decomposition of a banded matrix.

This function provides informative error messages to pass back to the bspline machinery; the actual computation is delegated to scipy.linalg.cholesky_banded().

Parameters

l
[numpy.ndarray] A matrix on which to perform the Cholesky decomposition. The matrix must be in a special, *lower* form described in scipy.linalg.cholesky_banded(). In addition, the input must be padded. If the original, square matrix has size $N \times N$, and the width of the band is b, l must be $b \times (N + b)$.

mininf
[float, optional] Entries in the l matrix are considered negative if they are less than this value (default 0.0).

Returns

:func:‘tuple‘
If problems were detected, the first item will be the index or indexes where the problem was detected, and the second item will simply be the input matrix. If no problems were detected, the first item will be -1, and the second item will be the Cholesky decomposition.

cholesky_solve

pydl.pydlutils.bspline.**cholesky_solve**(*a*, *bb*)
Solve the equation $Ax = b$ where a is a *lower* Cholesky-banded matrix.

In the bspline machinery, a needs to be padded. This function should only used with the output of cholesky_band(), to ensure the proper padding on a. Otherwise the computation is delegated to scipy.linalg.cho_solve_banded().

Parameters

a
[numpy.ndarray] *Lower* Cholesky decomposition of A in $Ax = b$.

bb
[numpy.ndarray] b in $Ax = b$.

Returns

:class:‘numpy.ndarray‘
The solution, padded to be the same shape as bb.

iterfit

pydl.pydlutils.bspline.**iterfit**(*xdata*, *ydata*, *invvar=None*, *upper=5*, *lower=5*, *x2=None*, *maxiter=10*, ***kwargs*)
Iteratively fit a B-spline set to data, with rejection.

Parameters

xdata
[numpy.ndarray] Independent variable.

ydata
[numpy.ndarray] Dependent variable.

invvar
[numpy.ndarray, optional] Inverse variance of ydata. If not set, it will be calculated based on the standard deviation.

upper
[int or float, optional] Upper rejection threshold in units of sigma, defaults to 5 sigma.

lower
[int or float, optional] Lower rejection threshold in units of sigma, defaults to 5 sigma.

x2
[numpy.ndarray, optional] Orthogonal dependent variable for 2d fits.

maxiter
[int, optional] Maximum number of rejection iterations, default 10. Set this to zero to disable rejection.

Returns

:func:‘tuple‘
A tuple containing the fitted bspline object and an output mask.

Classes

bspline(x[, nord, npoly, bkpt, bkspread])	B-spline class.

bspline

class pydl.pydlutils.bspline.**bspline**(*x*, *nord=4*, *npoly=1*, *bkpt=None*, *bkspread=1.0*, ***kwargs*)
Bases: object

B-spline class.

Functions in the idlutils bspline library are implemented as methods on this class.

Parameters

x
[numpy.ndarray] The data.

nord
[int, optional] The order of the B-spline. Default is 4, which is cubic.

npoly
[int, optional] Polynomial order to fit over 2nd variable, if supplied. If not supplied the order is 1.

bkpt
[numpy.ndarray, optional] To be documented.

bkspread
[float, optional] To be documented.

Attributes

breakpoints
To be documented.

nord
To be documented.

npoly
To be documented.

mask
To be documented.

coeff
To be documented.

icoeff
To be documented.

xmin
To be documented.

xmax
To be documented.

funcname
To be documented.

Init creates an object whose attributes are similar to the structure returned by the `create_bsplineset()` function.

Methods Summary

`action`(x[, x2])	Construct banded B-spline matrix, with dimensions [ndata, bandwidth].
`bsplvn`(x, ileft)	Calculates the value of all possibly nonzero B-splines at x of a certain order.
`fit`(xdata, ydata, invvar[, x2])	Calculate a B-spline in the least-squares sense.
`intrv`(x)	Find the segment between breakpoints which contain each value in the array x.
`maskpoints`(err)	Perform simple logic of which breakpoints to mask.
`value`(x[, x2, action, lower, upper])	Evaluate a B-spline at specified values.

Methods Documentation

action(*x*, *x2=None*)
Construct banded B-spline matrix, with dimensions [ndata, bandwidth].

Parameters

x
[`numpy.ndarray`] Independent variable.

x2
[`numpy.ndarray`, optional] Orthogonal dependent variable for 2d fits.

Returns

:func:'tuple'
A tuple containing the B-spline action matrix; the 'lower' parameter, a list of pixel positions, each corresponding to the first occurence of position greater than breakpoint indx; and 'upper', Same as lower, but denotes the upper pixel positions.

bsplvn(*x*, *ileft*)
Calculates the value of all possibly nonzero B-splines at x of a certain order.

Parameters

x
[`numpy.ndarray`] Independent variable.

ileft
[int] Breakpoint segements that contain x.

Returns

:class:'numpy.ndarray'
B-spline values.

fit(*xdata*, *ydata*, *invvar*, *x2=None*)
Calculate a B-spline in the least-squares sense.

Fit is based on two variables: xdata which is sorted and spans a large range where breakpoints are required ydata which can be described with a low order polynomial.

Parameters

xdata
[numpy.ndarray] Independent variable.

ydata
[numpy.ndarray] Dependent variable.

invvar
[numpy.ndarray] Inverse variance of ydata.

x2
[numpy.ndarray, optional] Orthogonal dependent variable for 2d fits.

Returns

:func:'tuple'
A tuple containing an integer error code, and the evaluation of the b-spline at the input values. An error code of -2 is a failure, -1 indicates dropped breakpoints, 0 is success, and positive integers indicate ill-conditioned breakpoints.

intrv(*x*)
Find the segment between breakpoints which contain each value in the array x.

The minimum breakpoint is nbkptord - 1, and the maximum is nbkpt - nbkptord - 1.

Parameters

x
[numpy.ndarray] Data values, assumed to be monotonically increasing.

Returns

:class:'numpy.ndarray'
Position of array elements with respect to breakpoints.

maskpoints(*err*)
Perform simple logic of which breakpoints to mask.

Parameters

err
[numpy.ndarray] The list of indexes returned by the cholesky routines.

Returns

:class:‘int‘
An integer indicating the results of the masking. -1 indicates that the error points were successfully masked. -2 indicates failure; the calculation should be aborted.

Notes

The mask attribute is modified, assuming it is possible to create the mask.

value(*x*, *x2=None*, *action=None*, *lower=None*, *upper=None*)
Evaluate a B-spline at specified values.

Parameters

x
[numpy.ndarray] Independent variable.

x2
[numpy.ndarray, optional] Orthogonal dependent variable for 2d fits.

action
[numpy.ndarray, optional] Action matrix to use. If not supplied it is calculated.

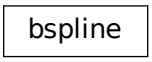
lower
[numpy.ndarray, optional] If the action parameter is supplied, this parameter must also be supplied.

upper
[numpy.ndarray, optional] If the action parameter is supplied, this parameter must also be supplied.

Returns

:func:‘tuple‘
A tuple containing the results of the bspline evaluation and a mask indicating where the evaluation was good.

Class Inheritance Diagram



1.2.3 pydl.pydlutils.cooling Module

This module corresponds to the cooling directory in idlutils.

Functions

read_ds_cooling(fname[, logT])	Read in the Sutherland & Dopita (1993) cooling function.

read_ds_cooling

pydl.pydlutils.cooling.**read_ds_cooling**(*fname*, *logT=None*)

Read in the Sutherland & Dopita (1993) cooling function.

Parameters

fname
[{ 'm-00.cie', 'm-05.cie', 'm+05.cie', 'm-10.cie', 'm-15.cie', 'm-20.cie', 'm-30.cie', 'mzero.cie' }] Name of the data file to read.

logT
[numpy.ndarray, optional] If provided, values will be interpolated to the provided values. If not provided, the values in the data files will be returned.

Returns

:func:'tuple'
A tuple containing logT and loglambda, respectively.

Raises

:exc:'ValueError'
If the input file name is invalid.

Notes

The data have been retrieved from http://www.mso.anu.edu.au/~ralph/data/cool/ and stored in the package.

Examples

```
>>> from pydl.pydlutils.cooling import read_ds_cooling
>>> logT, loglambda = read_ds_cooling('m-15.cie')
>>> logT[0:5] # doctest: +NORMALIZE_WHITESPACE
array([ 4.  ,  4.05,  4.1 ,  4.15,  4.2 ])
>>> loglambda[0:5] # doctest: +NORMALIZE_WHITESPACE
array([-26.  , -24.66, -23.52, -22.62, -22.11])
```

1.2.4 pydl.pydlutils.coord Module

This module corresponds to the coord directory in idlutils.

Functions

current_mjd()	Return the current MJD.
munu_to_radec(munu, icrs_frame)	Convert from SDSS great circle coordinates to equatorial coordinates.
radec_to_munu(icrs_frame, munu)	Convert from equatorial coordinates to SDSS great circle coordinates.
stripe_to_eta(stripe)	Convert from SDSS great circle coordinates to equatorial coordinates.
stripe_to_incl(stripe)	Convert from SDSS stripe number to an inclination relative to the equator.

current_mjd

pydl.pydlutils.coord.**current_mjd**()
Return the current MJD.

munu_to_radec

pydl.pydlutils.coord.**munu_to_radec**(*munu*, *icrs_frame*)
Convert from SDSS great circle coordinates to equatorial coordinates.

Parameters

munu
[SDSSMuNu] SDSS great circle coordinates (mu, nu).

Returns

:class:‘~astropy.coordinates.ICRS‘
Equatorial coordinates (RA, Dec).

radec_to_munu

pydl.pydlutils.coord.**radec_to_munu**(*icrs_frame*, *munu*)
Convert from equatorial coordinates to SDSS great circle coordinates.

Parameters

icrs_frame
[ICRS] Equatorial coordinates (RA, Dec).

Returns

:class:‘~pydl.pydlutils.coord.SDSSMuNu‘
SDSS great circle coordinates (mu, nu).

stripe_to_eta

pydl.pydlutils.coord.**stripe_to_eta**(*stripe*)
Convert from SDSS great circle coordinates to equatorial coordinates.

Parameters

stripe
[int or numpy.ndarray] SDSS Stripe number.

Returns

:class:'float' or :class:'numpy.ndarray'
The eta value in the SDSS (lambda,eta) coordinate system.

stripe_to_incl

pydl.pydlutils.coord.**stripe_to_incl**(*stripe*)
Convert from SDSS stripe number to an inclination relative to the equator.

Parameters

stripe
[int or numpy.ndarray] SDSS Stripe number.

Returns

:class:'float' or :class:'numpy.ndarray'
Inclination of the stripe relative to the equator (Dec = 0).

Classes

SDSSMuNu(*args[, copy, representation_type, ...])	SDSS Great Circle Coordinates

SDSSMuNu

class pydl.pydlutils.coord.**SDSSMuNu**(**args*, *copy=True*, *representation_type=None*, *differential_type=None*, ***kwargs*)
Bases: astropy.coordinates.BaseCoordinateFrame

SDSS Great Circle Coordinates

Parameters

mu
[Angle] Angle corresponding to longitude measured along a stripe.

nu
[Angle] Angle corresponding to latitude measured perpendicular to a stripe.

Notes

https://www.sdss.org/dr14/algorithms/surveycoords/

Attributes

stripe
SDSS Stripe Number .

node
Node of the great circle with respect to the celestial equator. In SDSS, this is almost always RA = 95.0 degrees.

incl
Inclination of the great circle with respect to the celestial equator.

phi
Counter-clockwise position angle w.r.t. north for an arc in the +nu direction.

Attributes Summary

default_differential
default_representation
frame_attributes
frame_specific_representation_info
incl
name
node
stripe

Attributes Documentation

default_differential

default_representation

frame_attributes = {'node': <astropy.coordinates.attributes.QuantityAttribute object>, 'stripe': <astro

frame_specific_representation_info

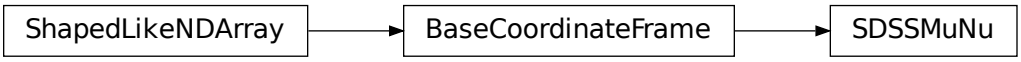
incl

name = 'sdssmunu'

node = <Quantity 95.0 deg>

stripe = 0

Class Inheritance Diagram



1.2.5 pydl.pydlutils.image Module

This module corresponds to the image directory in idlutils.

Functions

djs_maskinterp(yval, mask[, xval, axis, const])	Interpolate over masked pixels in a vector, image or 3-D array.
djs_maskinterp1(yval, mask[, xval, const])	Interpolate over a masked, 1-d array.

djs_maskinterp

pydl.pydlutils.image.**djs_maskinterp**(*yval*, *mask*, *xval=None*, *axis=None*, *const=False*)

Interpolate over masked pixels in a vector, image or 3-D array.

Parameters

yval
[numpy.ndarray] The input values

mask
[numpy.ndarray] The mask

xval
[numpy.ndarray, optional] If set, use these x values, otherwise use an array

axis
[int, optional] Must be set if yval has more than one dimension. If set to zero, interpolate along the first axis of the array, if set to one, interpolate along the second axis of the array, and so on.

const
[bool, optional] This value is passed to a helper function, djs_maskinterp1.

Returns

:class:‘numpy.ndarray‘
The interpolated array.

djs_maskinterp1

pydl.pydlutils.image.**djs_maskinterp1**(*yval*, *mask*, *xval=None*, *const=False*)
Interpolate over a masked, 1-d array.

Parameters

yval
[numpy.ndarray] The input values.

mask
[numpy.ndarray] The mask.

xval
[numpy.ndarray, optional] If set, use these x values, otherwise use an array.

const
[bool, optional] If set to True, bad values around the edges of the array will be set to a constant value. Because of the default behavior of numpy.interp(), this value actually makes no difference in the output.

Returns

:class:'numpy.ndarray'
The yval array with masked values replaced by interpolated values.

1.2.6 pydl.pydlutils.mangle Module

This module corresponds to the mangle directory in idlutils.

Mangle[1] is a software suite that supports the concept of polygons on a sphere, and is used to, for example, describe the window function of the Sloan Digital Sky Survey.

This implementation is intended to support the portions of Mangle that are included in idlutils. To simplify the coding somewhat, unlike idlutils, the caps information is accessed through polygon.x and polygon.cm, not polygon.caps.x or polygon.caps.cm.

Window function operations are already supported by is_in_polygon() and is_in_window(). However, calculation of polygon area (solid angle) from first principles is not yet implemented.

Note that in traditional geometry "spherical polygon" means a figure bounded by *great circles*. Mangle allows polygons to be bounded by *any* circle, great or not. Use care when comparing formulas in this module to formulas in the mathematical literature.

References

Functions

angles_to_x(points[, latitude])	Convert spherical angles to unit Cartesian vectors.
cap_distance(x, cm, points)	Compute the distance from a point to a cap, and also determine whether the point is inside or outside the cap.

Continued on next page

[1] Swanson, M. E. C.; Tegmark, Max; Hamilton, Andrew J. S.; Hill, J. Colin, 2008 MNRAS 387, 1391.

Table 11 – continued from previous page

circle_cap(radius, points)	Construct caps based on input points (directions on the unit sphere).
is_cap_used(use_caps, i)	Returns True if a cap is used.
is_in_cap(x, cm, points)	Are the points in a (single) given cap?
is_in_polygon(polygon, points[, ncaps])	Are the points in a given (single) polygon?
is_in_window(polygons, points[, ncaps])	Check to see if points lie within a set of polygons.
read_fits_polygons(filename[, convert])	Read a "polygon" format FITS file.
read_mangle_polygons(filename)	Read a "polygon" format ASCII file in Mangle's own format.
set_use_caps(polygon, index_list[, add, ...])	Set the bits in use_caps for a polygon.
x_to_angles(points[, latitude])	Convert unit Cartesian vectors to spherical angles.

angles_to_x

pydl.pydlutils.mangle.**angles_to_x**(*points*, *latitude=False*)

Convert spherical angles to unit Cartesian vectors.

Parameters

points
[ndarray] A set of angles in the form phi, theta (in degrees).

latitude
[bool, optional] If True, assume that the angles actually represent longitude, latitude, or equivalently, RA, Dec.

Returns

:class:'~numpy.ndarray'
The corresponding Cartesian vectors.

cap_distance

pydl.pydlutils.mangle.**cap_distance**(*x*, *cm*, *points*)

Compute the distance from a point to a cap, and also determine whether the point is inside or outside the cap.

Parameters

x
[ndarray or recarray] X value of the cap (3-vector).

cm
[ndarray or recarray] CM value of the cap.

points
[ndarray or recarray] If points is a 3-vector, or set of 3-vectors, then assume the point is a Cartesian unit vector. If point is a 2-vector or set of 2-vectors, assume the point is RA, Dec.

Returns

:class:‘~numpy.ndarray‘
The distance(s) to the point(s) in degrees. If the distance is negative, the point is outside the cap.

circle_cap

pydl.pydlutils.mangle.**circle_cap**(*radius*, *points*)
Construct caps based on input points (directions on the unit sphere).

Parameters

radius
[float or ndarray] Radii of the caps in degrees. This will become the CM value.

points
[ndarray or recarray] If points is a 3-vector, or set of 3-vectors, then assume the point is a Cartesian unit vector. If point is a 2-vector or set of 2-vectors, assume the point is RA, Dec.

Returns

:func:‘tuple‘
A tuple containing X and CM values for the cap.

is_cap_used

pydl.pydlutils.mangle.**is_cap_used**(*use_caps*, *i*)
Returns True if a cap is used.

Parameters

use_caps
[int] Bit mask indicating which cap is used.

i
[int] Number indicating which cap we are interested in.

Returns

:class:‘bool‘
True if a cap is used.

is_in_cap

pydl.pydlutils.mangle.**is_in_cap**(*x*, *cm*, *points*)
Are the points in a (single) given cap?

Parameters

x
[ndarray or recarray] X value of the cap (3-vector).

cm
[ndarray or recarray] CM value of the cap.

points
[ndarray or recarray] If points is a 3-vector, or set of 3-vectors, then assume the point is a Cartesian unit vector. If point is a 2-vector or set of 2-vectors, assume the point is RA, Dec.

Returns

:class:‘~numpy.ndarray‘
A boolean vector giving the result for each point.

is_in_polygon

pydl.pydlutils.mangle.**is_in_polygon**(*polygon*, *points*, *ncaps=0*)
Are the points in a given (single) polygon?

Parameters

polygon
[ManglePolygon] A polygon object.

points
[ndarray or recarray] If points is a 3-vector, or set of 3-vectors, then assume the point is a Cartesian unit vector. If point is a 2-vector or set of 2-vectors, assume the point is RA, Dec.

ncaps
[int, optional] If set, use only the first ncaps caps in polygon.

Returns

:class:‘~numpy.ndarray‘
A boolean vector giving the result for each point.

is_in_window

pydl.pydlutils.mangle.**is_in_window**(*polygons*, *points*, *ncaps=0*)
Check to see if points lie within a set of polygons.

Parameters

polygons
[PolygonList or FITS_polygon] A set of polygons.

points
[ndarray or recarray] If points is a 3-vector, or set of 3-vectors, then assume the point is a Cartesian unit vector. If point is a 2-vector or set of 2-vectors, assume the point is RA, Dec.

ncaps
[int, optional] If set, use only the first ncaps caps in polygon. This only exists to be passed to is_in_polygon().

Returns

:func:‘tuple‘
A tuple containing two ndarray. First, a boolean vector giving the result for each point. Second, an integer vector giving the index of the polygon that contains the point.

read_fits_polygons

pydl.pydlutils.mangle.**read_fits_polygons**(*filename*, *convert=False*)
Read a “polygon” format FITS file.

This function returns a subclass of FITS_rec that provides some convenient aliases for the columns of a typical FITS polygon file (which might be all upper-case).

Parameters

filename
[str] Name of FITS file to read.

convert
[bool, optional] If True, convert the data to a list of ManglePolygon objects. *Caution: This could result in some data being discarded!*

Returns

:class:‘~pydl.pydlutils.mangle.FITS_polygon‘ or :class:‘list‘
The data contained in HDU 1 of the FITS file.

read_mangle_polygons

pydl.pydlutils.mangle.**read_mangle_polygons**(*filename*)
Read a “polygon” format ASCII file in Mangle’s own format. These files typically have extension .ply or .pol.

Parameters

filename
[str] Name of FITS file to read.

Returns

:class:‘~pydl.pydlutils.mangle.PolygonList‘
A list-like object containing ManglePolygon objects and any metadata.

set_use_caps

pydl.pydlutils.mangle.**set_use_caps**(*polygon*, *index_list*, *add=False*, *tol=1e-10*, *allow_doubles=False*, *allow_neg_doubles=False*)
Set the bits in use_caps for a polygon.

Parameters

polygon
[ManglePolygon] A polygon object.

index_list
[array-like] A list of indices of caps to set in the polygon. Should be no longer, nor contain indices greater than the number of caps (polygon.ncaps).

add
[bool, optional] If True, don't initialize the use_caps value to zero, use the existing value associated with polygon instead.

tol
[float, optional] Tolerance used to determine whether two caps are identical.

allow_doubles
[bool, optional] Normally, this routine automatically sets use_caps such that no two caps with use_caps set are identical.

allow_neg_doubles
[bool, optional] Normally, two caps that are identical except for the sign of cm would be set unused. This inhibits that behaviour.

Returns

:class:'int'
Value of use_caps.

x_to_angles

pydl.pydlutils.mangle.**x_to_angles**(*points*, *latitude=False*)
Convert unit Cartesian vectors to spherical angles.

Parameters

points
[ndarray] A set of x, y, z coordinates.

latitude
[bool, optional] If True, convert the angles to longitude, latitude, or equivalently, RA, Dec.

Returns

:class:'~numpy.ndarray'
The corresponding spherical angles.

Classes

FITS_polygon	Handle polygons read in from a FITS file.
ManglePolygon(*args, **kwargs)	Simple object to represent a polygon.
PolygonList(*args, **kwargs)	A list that contains ManglePolygon objects and possibly some metadata.

FITS_polygon

class pydl.pydlutils.mangle.**FITS_polygon**
Bases: astropy.io.fits.FITS_rec

Handle polygons read in from a FITS file.

This class provides aliases for the columns in a typical FITS polygons file.

Construct a FITS record array from a recarray.

ManglePolygon

class pydl.pydlutils.mangle.**ManglePolygon**(**args*, ***kwargs*)
Bases: object

Simple object to represent a polygon.

A polygon may be instantiated with a row (FITS_record) from a FITS_polygon object, another ManglePolygon object (copy constructor), keyword arguments, or with no arguments at all, in which case it represents the whole sky.

Attributes

cm
[ndarray] The size of each cap in the polygon.

id
[int] An arbitrary ID number.

pixel
[int] Pixel this polygon is in.

use_caps
[int] Bitmask indicating which caps to use.

weight
[float] Weight factor assigned to the polygon.

Attributes Summary

ncaps	Number of caps in the polygon.
str	Solid angle of this polygon (steradians).
x	The orientation of each cap in the polygon.

Methods Summary

add_caps(x, cm)	Create a new polygon that contains additional caps.
cmminf()	The index of the smallest cap in the polygon, accounting for negative caps and use_caps.
copy()	Return an exact copy of the polygon.
garea()	Compute the area of a polygon.

Continued on next page

Table 14 – continued from previous page

gzeroar()	If at least one cap has zero area, then the whole polygon has zero area.
polyn(other, n[, complement])	Intersection of a polygon with the n th cap of another polygon.

Attributes Documentation

ncaps
Number of caps in the polygon.

str
Solid angle of this polygon (steradians).

x
The orientation of each cap in the polygon. The direction is specified by a unit 3-vector.

Methods Documentation

add_caps(*x*, *cm*)
Create a new polygon that contains additional caps. The caps are appended to the existing polygon's caps.

Parameters

x
[ndarray or recarray] X values of the new caps.

cm
[ndarray or recarray] CM values of the new caps.

Returns

:class:'ManglePolygon'
A new polygon object.

cmminf()
The index of the smallest cap in the polygon, accounting for negative caps and use_caps.

Returns

:class:'int'
Integer index of the smallest cap.

copy()
Return an exact copy of the polygon.

Returns

:class:'ManglePolygon'
A new polygon object.

garea()
Compute the area of a polygon.

See[1] for the detailed area formula, which is summarized here:

[1] Hamilton, A. J. S.; Tegmark, Max, 2004 MNRAS 349, 115.

- An empty polygon with no caps is defined to be the whole sky.
- A polygon with one cap has area `2*pi*self.cm`.
- A polygon with at least one cap with an area less than 2π has an area less than 2π.
- If every cap has an area greater than 2π, split the polygon into two smaller polygons and sum the two areas.

Returns

:class:'float'
The area of the polygon.

References

`gzeroar`()
If at least one cap has zero area, then the whole polygon has zero area.

Returns

:class:'bool'
`True` if the area is zero.

`polyn`(*other*, *n*, *complement=False*)
Intersection of a polygon with the `n` th cap of another polygon.

Parameters

other
[`ManglePolygon`] Polygon containing a cap to intersect the first polygon with.

n
[`int`] Index of the cap in `other`.

complement
[`bool`, optional] If `True`, set the sign of the cm value of `other` to be the complement of the original value.

Returns

:class:'~pydl.pydlutils.mangle.ManglePolygon'
A polygon containing the intersected caps.

PolygonList

`class` pydl.pydlutils.mangle.`PolygonList`(**args*, ***kwargs*)
Bases: `list`

A `list` that contains `ManglePolygon` objects and possibly some metadata.

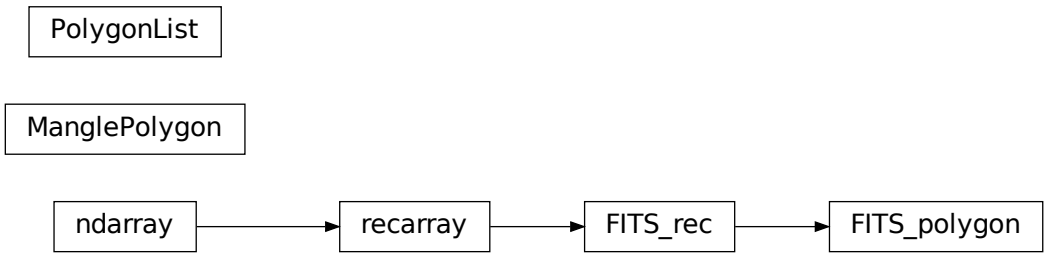
Parameters

header
[`list`, optional] Set the `header` attribute.

Attributes

header
[list] A list of strings containing metadata.

Class Inheritance Diagram



1.2.7 pydl.pydlutils.math Module

This module corresponds to the math directory in idlutils.

Functions

djs_median(array[, dimension, width, boundary])	Compute the median of an array.
djs_reject(data, model[, outmask, inmask, ...])	Routine to reject points when doing an iterative fit to data.
find_contiguous(x)	Find the longest sequence of contiguous non-zero array elements.

djs_median

pydl.pydlutils.math.**djs_median**(*array*, *dimension=None*, *width=None*, *boundary='none'*)
Compute the median of an array.

Use a filtering box or collapse the image along one dimension.

Parameters

array
[numpy.ndarray] input array

dimension
[int, optional] Compute the median over this dimension. It is an error to specify both `dimension` and `width`.

width
[int, optional] Width of the median window. In general, this should be an odd integer. It is an error to specify both `dimension` and `width`.

boundary
[{ 'none', 'reflect', 'nearest', 'wrap' }, optional] Boundary condition to impose. 'none' means no filtering is done within `width`/2 of the boundary. 'reflect' means reflect pixel values around the boundary. 'nearest' means use the values of the nearest boundary pixel. 'wrap' means wrap pixel values around the boundary. 'nearest' and 'wrap' are not implemented.

Returns

:class:'numpy.ndarray'
The output. If neither `dimension` nor `width` are set, this is a scalar value, just the output of `numpy.median()`. If `dimension` is set, then the result simply `numpy.median(array, dimension)`. If `width` is set, the result has the same shape as the input array.

djs_reject

pydl.pydlutils.math.**djs_reject**(*data*, *model*, *outmask=None*, *inmask=None*, *sigma=None*, *invvar=None*, *lower=None*, *upper=None*, *maxdev=None*, *maxrej=None*, *groupdim=None*, *groupsize=None*, *groupbadpix=False*, *grow=0*, *sticky=False*)

Routine to reject points when doing an iterative fit to data.

Parameters

data
[numpy.ndarray] The data

model
[numpy.ndarray] The model, must have the same number of dimensions as `data`.

outmask
[numpy.ndarray, optional] Output mask, generated by a previous call to `djs_reject`. If not supplied, this mask will be initialized to a mask that masks nothing. Although this parameter is technically optional, it will almost always be set.

inmask
[numpy.ndarray, optional] Input mask. Bad points are marked with a value that evaluates to `False`. Must have the same number of dimensions as `data`.

sigma
[numpy.ndarray, optional] Standard deviation of the data, used to reject points based on the values of `upper` and `lower`.

invvar
[numpy.ndarray, optional] Inverse variance of the data, used to reject points based on the values of `upper` and `lower`. If both `sigma` and `invvar` are set, `invvar` will be ignored.

lower
[int or float, optional] If set, reject points with data < model - lower * sigma.

upper
[int or float, optional] If set, reject points with data > model + upper * sigma.

maxdev
[int or float, optional] If set, reject points with abs(data-model) > maxdev. It is permitted to set all three of lower, upper and maxdev.

maxrej
[int or numpy.ndarray, optional] Maximum number of points to reject in this iteration. If groupsize or groupdim are set to arrays, this should be an array as well.

groupdim
To be documented.

groupsize
To be documented.

groupbadpix
[bool, optional] If set to True, consecutive sets of bad pixels are considered groups, overriding the values of groupsize.

grow
[int, optional] If set to a non-zero integer, N, the N nearest neighbors of rejected pixels will also be rejected.

sticky
[bool, optional] If set to True, pixels rejected in one iteration remain rejected in subsequent iterations, even if the model changes.

Returns

:func:'tuple'
A tuple containing a mask where rejected data values are False and a boolean value set to True if djs_reject believes there is no further rejection to be done.

Raises

:exc:'ValueError'
If dimensions of various inputs do not match.

find_contiguous

pydl.pydlutils.math.**find_contiguous**(*x*)
Find the longest sequence of contiguous non-zero array elements.

Parameters

x
[numpy.ndarray] A 1d array. A dtype of bool is preferred although any dtype where the operation `if x[k]:` is well-defined should work.

Returns

:class:'list'
A list of indices of the longest contiguous non-zero sequence.

Examples

```
>>> import numpy as np
>>> from pydl.pydlutils.math import find_contiguous
>>> find_contiguous(np.array([0,1,1,1,0,1,1,0,1]))
[1, 2, 3]
```

Classes

computechi2(bvec, sqivar, amatrix)	Solve the linear set of equations $Ax = b$ using SVD.

computechi2

class pydl.pydlutils.math.**computechi2**(*bvec*, *sqivar*, *amatrix*)

Bases: object

Solve the linear set of equations $Ax = b$ using SVD.

The attributes of this class are all read-only properties, implemented with lazyproperty.

Parameters

bvec
[numpy.ndarray] The b vector in $Ax = b$. This vector has length N.

sqivar
[numpy.ndarray] The reciprocal of the errors in bvec. The name comes from the square root of the inverse variance, which is what this is.

amatrix
[numpy.ndarray] The matrix A in $Ax = b$. The shape of this matrix is (N, M).

Initialize the object and perform initial computations.

Attributes Summary

acoeff	(ndarray) The fit parameters, x, in $Ax = b$.
chi2	(float) The χ^2 value of the fit.
covar	(ndarray) The covariance matrix.
dof	(int) The degrees of freedom of the fit.
var	(ndarray) The variances of the fit.
yfit	(ndarray) The evaluated best-fit at each point.

Attributes Documentation

acoeff
(ndarray) The fit parameters, x, in $Ax = b$. This vector has length M.

chi2
(float) The χ^2 value of the fit.

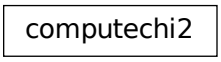
covar
(ndarray) The covariance matrix. The shape of this matrix is (M, M).

dof
(int) The degrees of freedom of the fit. This is the number of values of bvec that have sqivar > 0 minus the number of fit paramaters, which is equal to M.

var
(ndarray) The variances of the fit. This is identical to the diagonal of the covariance matrix. This vector has length M.

yfit
(ndarray) The evaluated best-fit at each point. This vector has length N.

Class Inheritance Diagram



1.2.8 pydl.pydlutils.misc Module

This module corresponds to the misc directory in idlutils.

Functions

decode_mixed(x)	Convert bytes in Numpy arrays into strings.
djs_laxisgen(dims[, iaxis])	Returns an integer array where each element of the array is set equal to its index number along the specified axis.
djs_laxisnum(dims[, iaxis])	Returns an integer array where each element of the array is set equal to its index number in the specified axis.
hogg_iau_name(ra, dec[, prefix, precision])	Properly format astronomical source names to the IAU convention.
hogg_iau_name_main()	
struct_print(array[, filename, formatcodes, ...])	Print a NumPy record array (analogous to an IDL structure) in a nice way.

decode_mixed

pydl.pydlutils.misc.**decode_mixed**(x)
Convert bytes in Numpy arrays into strings. Leave other stuff alone.

Parameters

x
[object] Input object.

Returns

object
If x has a decode() method, x.decode() will be returned. Otherwise x will be returned unchanged.

djs_laxisgen

pydl.pydlutils.misc.**djs_laxisgen**(*dims*, *iaxis=0*)
Returns an integer array where each element of the array is set equal to its index number along the specified axis.

Parameters

dims
[list] Dimensions of the array to return.

iaxis
[int, optional] Index along this dimension.

Returns

:class:'numpy.ndarray'
An array of indexes with dtype=int32.

Raises

:exc:'ValueError'
If iaxis is greater than or equal to the number of dimensions.

Notes

For two or more dimensions, there is no difference between this routine and djs_laxisnum().

Examples

```
>>> from pydl.pydlutils.misc import djs_laxisgen
>>> print(djs_laxisgen([4,4]))
[[0 0 0 0]
 [1 1 1 1]
 [2 2 2 2]
 [3 3 3 3]]
```

djs_laxisnum

pydl.pydlutils.misc.**djs_laxisnum**(*dims*, *iaxis=0*)
Returns an integer array where each element of the array is set equal to its index number in the specified axis.

Parameters

dims
[list] Dimensions of the array to return.

iaxis
[int, optional] Index along this dimension.

Returns

:class:‘numpy.ndarray‘
An array of indexes with `dtype=int32`.

Raises

:exc:‘ValueError‘
If `iaxis` is greater than or equal to the number of dimensions, or if number of dimensions is greater than three.

Notes

For two or more dimensions, there is no difference between this routine and `djs_laxisgen()`.

Examples

```
>>> from pydl.pydlutils.misc import djs_laxisnum
>>> print(djs_laxisnum([4,4]))
[[0 0 0 0]
 [1 1 1 1]
 [2 2 2 2]
 [3 3 3 3]]
```

hogg_iau_name

pydl.pydlutils.misc.**hogg_iau_name**(*ra*, *dec*, *prefix='SDSS'*, *precision=1*)
Properly format astronomical source names to the IAU convention.

Parameters

ra
[float or numpy.ndarray] Right ascencion in decimal degrees

dec
[float or numpy.ndarray] Declination in decimal degrees.

prefix
[str, optional] Add this prefix to the string, defaults to ‘SDSS’.

precision
[int, optional] Display this many digits of precision on seconds, default 1.

Returns

:class:‘str‘ or :class:‘list‘
The IAU name for the coordinates.

Examples

```
>>> from pydl.pydlutils.misc import hogg_iau_name
>>> hogg_iau_name(354.120375,-0.544777778)
'SDSS J233628.89-003241.2'
```

hogg_iau_name_main

pydl.pydlutils.misc.**hogg_iau_name_main**()

struct_print

pydl.pydlutils.misc.**struct_print**(*array*, *filename=None*, *formatcodes=None*, *alias=None*, *fdigit=5*, *ddigit=7*, *html=False*, *no_head=False*, *silent=False*)

Print a NumPy record array (analogous to an IDL structure) in a nice way.

Parameters

array
[numpy.ndarray] A record array to print.

filename
[str or file-like, optional] If supplied, write to this file.

formatcodes
[dict, optional] If supplied, use explicit format for certain columns.

alias
[dict, optional] If supplied, use this mapping of record array column names to printed column names.

fdigit
[int, optional] Width of 32-bit floating point columns, default 5.

ddigit
[int, optional] Width of 64-bit floating point columns, default 7.

html
[bool, optional] If True, print an html table.

no_head
[bool, optional] If True, *don't* print a header line.

silent
[bool, optional] If True, do not print the table, just return it.

Returns

:func:'tuple'
A tuple containing a list of the lines in the table. If html is True, also returns a list of lines of CSS for formatting the table.

Examples

```
>>> import numpy as np
>>> from pydl.pydlutils.misc import struct_print
>>> struct_print(np.array([(1,2.34,'five'),(2,3.456,'seven'),(3,4.5678,'nine')],dtype=[('a','i4'),
↪('bb','f4'),('ccc','S5')]),silent=True)
(['a bb          ccc  ', '- ----------- -----', '1        2.34 five ', '2       3.456 seven', '3
↪    4.5678 nine '], [])
```

1.2.9 pydl.pydlutils.rgbcolor Module

This module corresponds to the rgbcolor directory in idlutils.

This code reproduces the algorithms of Lupton *et al.* (2004)[2]. The purpose is to produce nice color (RGB, JPEG, PNG, etc.) from FITS images in astronomical filter bands.

References

Functions

nw_arcsinh(colors[, nonlinearity])	Scales colors by a degree of nonlinearity specified by user.
nw_cut_to_box(colors[, origin])	Limits the pixel values of the image to a 'box', so that the colors do not saturate to white but to a specific color.
nw_float_to_byte(image[, bits])	Converts an array of floats in [0.0, 1.0] to bytes in [0, 255].
nw_scale_rgb(colors[, scales])	Multiply RGB image by color-dependent scale factor.

nw_arcsinh

pydl.pydlutils.rgbcolor.**nw_arcsinh**(*colors*, *nonlinearity=3.0*)

Scales colors by a degree of nonlinearity specified by user.

The input image must have zero background (*i.e.*, it must already be background-subtracted).

Parameters

colors
[ndarray] 3D Array containing RGB image. The dimensions should be (X, Y, 3).

nonlinearity
[float] Amount of nonlinear scaling. If set to zero, no scaling will be performed (this is equivalent to linear scaling).

Returns

:class:'~numpy.ndarray'
The scaled image.

Raises

[2] Lupton, Robert, et al., 2004 PASP 116, 113.

:exc:‘ValueError‘
If `colors` has the wrong shape.

nw_cut_to_box

`pydl.pydlutils.rgbcolor.`**`nw_cut_to_box`**(*colors*, *origin=(0.0, 0.0, 0.0)*)
Limits the pixel values of the image to a ‘box’, so that the colors do not saturate to white but to a specific color.

Parameters

colors
[`ndarray`] 3D Array containing RGB image. The dimensions should be (X, Y, 3).

origin
[`tuple()` or `ndarray`] An array with 3 elements. The “distance” from this origin is considered saturated.

Returns

:class:‘~numpy.ndarray‘
The “boxed” image.

Raises

:exc:‘ValueError‘
If `colors` or `origin` has the wrong shape.

nw_float_to_byte

`pydl.pydlutils.rgbcolor.`**`nw_float_to_byte`**(*image*, *bits=8*)
Converts an array of floats in [0.0, 1.0] to bytes in [0, 255].

Parameters

image
[`ndarray`] Image to convert.

bits
[`int`, optional] Number of bits in final image.

Returns

:class:‘~numpy.ndarray‘
Converted image.

nw_scale_rgb

`pydl.pydlutils.rgbcolor.`**`nw_scale_rgb`**(*colors*, *scales=(1.0, 1.0, 1.0)*)
Multiply RGB image by color-dependent scale factor.

Parameters

colors
[ndarray] 3D Array containing RGB image. The dimensions should be (X, Y, 3).

scales
[tuple() or ndarray, optional] An array with 3 elements.

Returns

:class:‘~numpy.ndarray‘
The scaled image.

Raises

:exc:‘ValueError‘
If colors or scales has the wrong shape.

1.2.10 pydl.pydlutils.sdss Module

This module corresponds to the sdss directory in idlutils.

Functions

default_skyversion()	Returns skyversion number to use for photoop imaging.
sdss_astrombad(run, camcol, field[, ...])	For a list of RUN, CAMCOL, FIELD, return whether each field has bad astrometry.
sdss_flagexist(flagname, bitname[, ...])	Check for the existence of flags.
sdss_flagname(flagname, flagvalue[, concat])	Return a list of flag names corresponding to the values.
sdss_flagval(flagname, bitname)	Convert bitmask names into values.
sdss_objid(run, camcol, field, objnum[, ...])	Convert SDSS photometric identifiers into CAS-style ObjID.
sdss_specobjid(plate, fiber, mjd, run2d[, ...])	Convert SDSS spectrum identifiers into CAS-style specObjID.
sdss_sweep_circle(ra, dec, radius[, stype, ...])	Read the SDSS datasweep files and return objects around a location.
set_maskbits([idlutils_version, maskbits_file])	Populate the maskbits cache.
unwrap_specobjid(specObjID[, run2d_integer, ...])	Unwrap CAS-style specObjID into plate, fiber, mjd, run2d.

default_skyversion

pydl.pydlutils.sdss.**default_skyversion**()
Returns skyversion number to use for photoop imaging.

Returns

:class:‘int‘
The default skyversion number.

Notes

The skyversion number is hard-coded to 2.

Examples

```
>>> from pydl.pydlutils.sdss import default_skyversion
>>> default_skyversion()
2
```

sdss_astrombad

pydl.pydlutils.sdss.**sdss_astrombad**(*run*, *camcol*, *field*, *photolog_version='dr10'*)
For a list of RUN, CAMCOL, FIELD, return whether each field has bad astrometry.

Parameters

run, camcol, field
[int or array of int] Run, camcol and field. If arrays are passed, all must have the same length.

photolog_version
[str, optional] Use this version of photolog to obtain the obBadfields.par file, if PHOTOLOG_DIR is not set.

Returns

:class:'numpy.ndarray' of :class:'bool'
Array of bool. True indicates the field is bad.

Raises

:exc:'ValueError'
If the sizes of the arrays don't match or if the array values are out of bounds.

Notes

Reads data from $PHOTOLOG_DIR/opfiles/opBadFields.par.

If there is a problem with one camcol, we assume a problem with all camcols.

sdss_flagexist

pydl.pydlutils.sdss.**sdss_flagexist**(*flagname*, *bitname*, *flagexist=False*, *whichexist=False*)
Check for the existence of flags.

Parameters

flagname
[str] The name of a bitmask group. Not case-sensitive.

bitname
[str or list] The name(s) of the specific bitmask(s) within the flagname group.

flagexist
[bool, optional] If flagexist is True, return a tuple with the second component indicating whether the binary flag named flagname exists, even if bitname is wrong.

whichexist
[bool, optional] If whichexist is True, return a list containing existence test results for each individual flag.

Returns

:class:‘bool‘ or :func:‘tuple‘
A boolean value or a tuple of bool.

sdss_flagname

pydl.pydlutils.sdss.**sdss_flagname**(*flagname*, *flagvalue*, *concat=False*)
Return a list of flag names corresponding to the values.

Parameters

flagname
[str] The name of a bitmask group. Not case-sensitive.

flagvalue
[long] The value to be converted into bitmask names.

concat
[bool, optional] If set to True, the list of names is converted to a space-separated string.

Returns

:class:‘str‘ or :class:‘list‘
The names of the bitmasks encoded in flagvalue.

Raises

:exc:‘KeyError‘
If flagname is invalid

Examples

```
>>> from pydl.pydlutils.sdss import sdss_flagname
>>> sdss_flagname('ANCILLARY_TARGET1',2310346608843161600) # doctest: +REMOTE_DATA
['BRIGHTGAL', 'BLAZGX', 'ELG']
```

sdss_flagval

pydl.pydlutils.sdss.**sdss_flagval**(*flagname*, *bitname*)
Convert bitmask names into values.

Converts human-readable bitmask names into numerical values. The inputs are not case-sensitive; all inputs are converted to upper case internally.

Parameters

flagname
[str] The name of a bitmask group.

bitname
[str or list] The name(s) of the specific bitmask(s) within the flagname group.

Returns

:class:'numpy.uint64'
The value of the bitmask name(s).

Raises

:exc:'KeyError'
If flagname or bitname are invalid names.

Examples

```
>>> from pydl.pydlutils.sdss import sdss_flagval
>>> sdss_flagval('ANCILLARY_TARGET1',['BLAZGX','ELG','BRIGHTGAL']) # doctest: +REMOTE_DATA
2310346608843161600
```

sdss_objid

pydl.pydlutils.sdss.**sdss_objid**(*run*, *camcol*, *field*, *objnum*, *rerun=301*, *skyversion=None*, *firstfield=None*)

Convert SDSS photometric identifiers into CAS-style ObjID.

Bits are assigned in ObjID thus:

Bits	Name	Comment
63	empty	unassigned
59-62	skyVersion	resolved sky version (0-15)
48-58	rerun	number of pipeline rerun
32-47	run	run number
29-31	camcol	camera column (1-6)
28	firstField	is this the first field in segment? Usually 0.
16-27	field	field number within run
0-15	object	object number within field

Parameters

run, camcol, field, objnum
[int or array of int] Run, camcol, field and object number within field. If arrays are passed, all must have the same length.

rerun, skyversion, firstfield
[int or array of int, optional] rerun, skyversion and firstfield usually don't change at all, especially for ObjIDs in DR8 and later. If supplied, make sure the size matches all the other values.

Returns

:class:'numpy.ndarray' of :class:'numpy.int64'
The ObjIDs of the objects.

Raises

:exc:'ValueError'
If the sizes of the arrays don't match or if the array values are out of bounds.

Notes

- The firstField flag is never set in ObjIDs from DR8 and later.
- On 32-bit systems, makes sure to explicitly declare all inputs as 64-bit integers.

Examples

```
>>> from pydl.pydlutils.sdss import sdss_objid
>>> print(sdss_objid(3704,3,91,146))
[1237661382772195474]
```

sdss_specobjid

pydl.pydlutils.sdss.**sdss_specobjid**(*plate*, *fiber*, *mjd*, *run2d*, *line=None*, *index=None*)
Convert SDSS spectrum identifiers into CAS-style specObjID.

Bits are assigned in specObjID thus:

Bits	Name	Comment
50-63	Plate ID	14 bits
38-49	Fiber ID	12 bits
24-37	MJD	Date plate was observed minus 50000 (14 bits)
10-23	run2d	Spectroscopic reduction version
0-9	line/index	0 for use in SpecObj files see below for other uses (10 bits)

Parameters

plate, fiber, mjd
[int or array of int] Plate, fiber ID, and MJD for a spectrum. If arrays are passed, all must have the same length. The MJD value must be greater than 50000.

run2d
[int, str or array of int or str] The run2d value must be an integer or a string of the form 'vN_M_P'. If an array is passed, it must have the same length as the other inputs listed

above. If the string form is used, the values are restricted to $5 \leq N \leq 6$, $0 \leq M \leq 99$, $0 \leq P \leq 99$.

line
[int, optional] A line index, only used for defining specObjID for SpecLine files. `line` and `index` cannot both be non-zero.

index
[int, optional] An index measure, only used for defining specObjID for SpecLineIndex files. `line` and `index` cannot both be non-zero.

Returns

:class:‘numpy.ndarray‘ of :class:‘numpy.uint64‘
The specObjIDs of the objects.

Raises

:exc:‘ValueError‘
If the sizes of the arrays don’t match or if the array values are out of bounds.

Notes

- On 32-bit systems, makes sure to explicitly declare all inputs as 64-bit integers.
- This function defines the SDSS-III/IV version of specObjID, used for SDSS DR8 and subsequent data releases. It is not compatible with SDSS DR7 or earlier.
- If the string form of `run2d` is used, the bits are assigned by the formula $(N-5) \times 10000 + M \times 100 + P$.

Examples

```
>>> from pydl.pydlutils.sdss import sdss_specobjid
>>> print(sdss_specobjid(4055,408,55359,'v5_7_0'))
[4565636362342690816]
```

sdss_sweep_circle

pydl.pydlutils.sdss.**sdss_sweep_circle**(*ra*, *dec*, *radius*, *stype='star'*, *allobj=False*)
Read the SDSS datasweep files and return objects around a location.

Parameters

ra, dec
[float] The sky location to search, J2000 degrees.

radius
[float] The radius around `ra`, `dec` to search.

stype
[str, optional] The type of object to search, ‘star’, ‘gal’ or ‘sky’. The default is ‘star’.

allobj
[bool, optional] If set to `True`, return all objects found, not just SURVEY_PRIMARY.

Returns

:class:‘numpy.ndarray‘
The data extracted from the sweep files.

Raises

:exc:‘PydlutilsException‘
If PHOTO_SWEEP is not set.

Notes

Assumes that the sweep files exist in PHOTO_SWEEP and that index files have been created.

set_maskbits

pydl.pydlutils.sdss.**set_maskbits**(*idlutils_version='v5_5_24'*, *maskbits_file=None*)
Populate the maskbits cache.

Parameters

idlutils_version
[str, optional] Fetch the sdssMaskbits.par file corresponding to this idlutils version.

maskbits_file
[str, optional] Use an explicit file instead of downloading the official version. This should only be used for tests.

Returns

:class:‘dict‘
A dictionary of bitmasks suitable for caching.

Raises

:exc:‘URLError‘
If the data file could not be retrieved.

unwrap_specobjid

pydl.pydlutils.sdss.**unwrap_specobjid**(*specObjID*, *run2d_integer=False*, *specLineIndex=False*)
Unwrap CAS-style specObjID into plate, fiber, mjd, run2d.

See sdss_specobjid() for details on how the bits within a specObjID are assigned.

Parameters

specObjID
[numpy.ndarray] An array containing 64-bit integers or strings. If strings are passed, they will be converted to integers internally.

run2d_integer
[bool, optional] If True, do *not* attempt to convert the encoded run2d values to a string of the form 'vN_M_P'.

specLineIndex
[bool, optional] If True interpret any low-order bits as being an 'index' rather than a 'line'.

Returns

:class:'numpy.recarray'
A record array with the same length as specObjID, with the columns 'plate', 'fiber', 'mjd', 'run2d', 'line'.

Examples

```
>>> from numpy import array, uint64
>>> from pydl.pydlutils.sdss import unwrap_specobjid
>>> unwrap_specobjid(array([4565636362342690816], dtype=uint64))
rec.array([(4055, 408, 55359, 'v5_7_0', 0)],
          dtype=[('plate', '<i4'), ('fiber', '<i4'), ('mjd', '<i4'), ('run2d', '<U8'), ('line', '
↪<i4')])
```

1.2.11 pydl.pydlutils.spheregroup Module

This module corresponds to the spheregroup directory in idlutils.

Functions

spheregroup(ra, dec, linklength[, chunksize])	Perform friends-of-friends grouping given ra/dec coordinates.
spherematch(ra1, dec1, ra2, dec2, matchlength)	Match points on a sphere.

spheregroup

pydl.pydlutils.spheregroup.**spheregroup**(*ra*, *dec*, *linklength*, *chunksize=None*)
Perform friends-of-friends grouping given ra/dec coordinates.

Parameters

ra, dec
[numpy.ndarray] Arrays of coordinates to group in decimal degrees.

linklength
[float] Linking length for the groups in decimal degrees.

chunksize
[float, optional] Break up the sphere into chunks of this size in decimal degrees.

Returns

:func:'tuple'

A tuple containing the group number of each object, the multiplicity of each group, the first member of each group, and the next member of the group for each object.

Raises

:exc:‘PydlutilsException‘
If the array of coordinates only contains one point.

Notes

It is important that `chunksize` >= 4 * `linklength`. This is enforced.

Warning: Behavior at the poles is not well tested.

spherematch

pydl.pydlutils.spheregroup.**spherematch**(*ra1*, *dec1*, *ra2*, *dec2*, *matchlength*, *chunksize=None*, *maxmatch=1*)

Match points on a sphere.

Parameters

ra1, dec1, ra2, dec2
[`numpy.ndarray`] The sets of coordinates to match. Assumed to be in decimal degrees

matchlength
[`float`] Two points closer than this separation are matched. Assumed to be in decimal degrees.

chunksize
[`float`, optional] Value to pass to chunk assignment.

maxmatch
[`int`, optional] Allow up to `maxmatch` matches per coordinate. Default 1. If set to zero, All possible matches will be returned.

Returns

:func:‘tuple‘
A tuple containing the indices into the first set of points, the indices into the second set of points and the match distance in decimal degrees.

Notes

If you have sets of coordinates that differ in size, call this function with the larger list first. This exploits the inherent asymmetry in the underlying code to reduce memory use.

Warning: Behavior at the poles is not well tested.

Classes

chunks(ra, dec, minSize)	chunks class
groups(coordinates, distance[, separation])	Group a set of objects (a list of coordinates in some space) based on a friends-of-friends algorithm

chunks

class pydl.pydlutils.spheregroup.**chunks**(*ra*, *dec*, *minSize*)

Bases: object

chunks class

Functions for creating and manipulating spherical chunks are implemented as methods on this class.

Init creates an object whose attributes are similar those created by the setchunks() function in the spheregroup library.

Methods Summary

assign(ra, dec, marginSize)	Take the objects and the chunks (already defined in the constructor) and assign the objects to the appropriate chunks, with some leeway given by the parameter marginSize.
chunkfriendsoffriends(ra, dec, chunkList, ...)	Does friends-of-friends on the ra, dec that are defined by chunkList.
cosDecMin(i)	Frequently used utility function.
friendsoffriends(ra, dec, linkSep)	Friends-of-friends using chunked data.
get(ra, dec)	Find the chunk to which a given point belongs.
getbounds(ra, dec, marginSize)	Find the set of chunks a point (with margin) belongs to.
getraminmax(ra, raOffset)	Utility function used by rarange.
rarange(ra, minSize)	Finds the offset which yields the smallest raRange & returns both.

Methods Documentation

assign(*ra*, *dec*, *marginSize*)

Take the objects and the chunks (already defined in the constructor) and assign the objects to the appropriate chunks, with some leeway given by the parameter marginSize. Basically, at the end, each chunk should be associated with a list of the objects that belong to it.

chunkfriendsoffriends(*ra*, *dec*, *chunkList*, *linkSep*)

Does friends-of-friends on the ra, dec that are defined by chunkList.

cosDecMin(*i*)

Frequently used utility function.

friendsoffriends(*ra*, *dec*, *linkSep*)

Friends-of-friends using chunked data.

get(*ra*, *dec*)

Find the chunk to which a given point belongs.

getbounds(*ra*, *dec*, *marginSize*)
Find the set of chunks a point (with margin) belongs to.

getraminmax(*ra*, *raOffset*)
Utility function used by rarange.

rarange(*ra*, *minSize*)
Finds the offset which yields the smallest raRange & returns both.

Notes

> **Warning:** This is not (yet) well-defined for the case of only one point.

groups

class pydl.pydlutils.spheregroup.**groups**(*coordinates*, *distance*, *separation='euclid'*)
Bases: object

Group a set of objects (a list of coordinates in some space) based on a friends-of-friends algorithm

Init creates an object and performs the friends-of-friends algorithm. The coordinates can have arbitrary dimensions, with each column representing one of the dimensions. Each row defines an object. If separation is not defined it defaults to Euclidean space.

Methods Summary

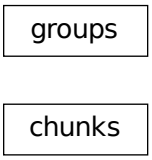
euclid(x1, x2)	Pythagorean theorem in Euclidean space with arbitrary number of dimensions.
sphereradec(x1, x2)	Separation of two points on a 2D-sphere, assuming they are in longitude-latitude or right ascension-declination form.

Methods Documentation

static euclid(*x1*, *x2*)
Pythagorean theorem in Euclidean space with arbitrary number of dimensions.

static sphereradec(*x1*, *x2*)
Separation of two points on a 2D-sphere, assuming they are in longitude-latitude or right ascension-declination form. Assumes everything is already in radians.

Class Inheritance Diagram



1.2.12 pydl.pydlutils.trace Module

This module corresponds to the trace directory in idlutils.

Functions

fchebyshev(x, m)	Compute the first m Chebyshev polynomials.
fchebyshev_split(x, m)	Compute the first m Chebyshev polynomials, but modified to allow a split in the baseline at $x = 0$.
fpoly(x, m)	Compute the first m simple polynomials.
func_fit(x, y, ncoeff[, invvar, ...])	Fit x, y positions to a functional form.
traceset2xy(tset[, xpos, ignore_jump])	Convert from a trace set to an array of x,y positions.
xy2traceset(xpos, ypos, **kwargs)	Convert from x,y positions to a trace set.

fchebyshev

pydl.pydlutils.trace.**fchebyshev**(x, m)

Compute the first m Chebyshev polynomials.

Parameters

x
[array-like] Compute the Chebyshev polynomials at these abscissa values.

m
[int] The number of Chebyshev polynomials to compute. For example, if $m = 3$, $T_0(x)$, $T_1(x)$ and $T_2(x)$ will be computed.

Returns

:class:'numpy.ndarray'

fchebyshev_split

pydl.pydlutils.trace.**fchebyshev_split**(*x*, *m*)

Compute the first `m` Chebyshev polynomials, but modified to allow a split in the baseline at $x = 0$. The intent is to allow a model fit where a constant term is different for positive and negative `x`.

Parameters

x
[array-like] Compute the Chebyshev polynomials at these abscissa values.

m
[`int`] The number of Chebyshev polynomials to compute. For example, if $m = 3$, $T_0(x)$, $T_1(x)$ and $T_2(x)$ will be computed.

Returns

:class:'numpy.ndarray'

fpoly

pydl.pydlutils.trace.**fpoly**(*x*, *m*)

Compute the first `m` simple polynomials.

Parameters

x
[array-like] Compute the simple polynomials at these abscissa values.

m
[`int`] The number of simple polynomials to compute. For example, if $m = 3$, x^0, x^1 and x^2 will be computed.

Returns

:class:'numpy.ndarray'

func_fit

pydl.pydlutils.trace.**func_fit**(*x*, *y*, *ncoeff*, *invvar=None*, *function_name='legendre'*, *ia=None*, *inputans=None*, *inputfunc=None*)

Fit `x`, `y` positions to a functional form.

Parameters

x
[array-like] X values (independent variable).

y
[array-like] Y values (dependent variable).

ncoeff
[int] Number of coefficients to fit.

invvar
[array-like, optional] Weight values; inverse variance.

function_name
[str, optional] Function name, default 'legendre'.

ia
[array-like, optional] An array of bool of length ncoeff specifying free (True) and fixed (False) parameters.

inputans
[array-like, optional] An array of values of length ncoeff specifying the values of the fixed parameters.

inputfunc
[array-like, optional] Multiply the function fit by these values.

Returns

:func:'tuple' of array-like
Fit coefficients, length ncoeff; fitted values.

Raises

:exc:'KeyError'
If an invalid function type is selected.

:exc:'ValueError'
If input dimensions do not agree.

traceset2xy

pydl.pydlutils.trace.**traceset2xy**(*tset*, *xpos=None*, *ignore_jump=False*)
Convert from a trace set to an array of x,y positions.

Parameters

tset
[TraceSet] A TraceSet object.

xpos
[array-like, optional] If provided, evaluate the trace set at these positions. Otherwise the positions will be constructed from the trace set object iself.

ignore_jump
[bool, optional] If True, ignore any jump information in the tset object

Returns

:func:'tuple' of array-like
The x, y positions.

xy2traceset

pydl.pydlutils.trace.**xy2traceset**(*xpos*, *ypos*, ***kwargs*)

Convert from x,y positions to a trace set.

Parameters

xpos, ypos
[array-like] X,Y positions corresponding as [nx,Ntrace] arrays.

invvar
[array-like, optional] Inverse variances for fitting.

func
[str, optional] Function type for fitting; defaults to 'legendre'.

ncoeff
[int, optional] Number of coefficients to fit. Defaults to 3.

xmin, xmax
[float, optional] Explicitly set minimum and maximum values, instead of computing them from xpos.

maxiter
[int, optional] Maximum number of rejection iterations; set to 0 for no rejection; default to 10.

inmask
[array-like, optional] Mask set to 1 for good points and 0 for rejected points; same dimensions as xpos, ypos. Points rejected by inmask are always rejected from the fits (the rejection is "sticky"), and will also be marked as rejected in the outmask attribute.

ia, inputans, inputfunc
[array-like, optional] These arguments will be passed to func_fit().

xjumplo
[float, optional] x position locating start of an x discontinuity

xjumphi
[float, optional] x position locating end of that x discontinuity

xjumpval
[float, optional] magnitude of the discontinuity "jump" between those bounds (previous 3 keywords motivated by BOSS 2-phase readout)

Returns

:class:'TraceSet'
A TraceSet object.

Classes

TraceSet(*args, **kwargs)	Implements the idea of a trace set.

TraceSet

class pydl.pydlutils.trace.**TraceSet**(**args*, ***kwargs*)

Bases: object

Implements the idea of a trace set.

Attributes

func
[str] Name of function type used to fit the trace set.

xmin
[float-like] Minimum x value.

xmax
[float-like] Maximum x value.

coeff
[array-like] Coefficients of the trace set fit.

nTrace
[int] Number of traces in the object.

ncoeff
[int] Number of coefficients of the trace set fit.

xjumplo
[float-like] Jump value, for BOSS readouts.

xjumphi
[float-like] Jump value, for BOSS readouts.

xjumpval
[float-like] Jump value, for BOSS readouts.

outmask
[array-like] When initialized with x,y positions, this contains the rejected points.

yfit
[array-like] When initialized with x,y positions, this contains the fitted y values.

This class can be initialized either with a set of xy positions, or with a trace set HDU from a FITS file.

Attributes Summary

has_jump	True if jump conditions are set.
nx	Number of x values.
xRange	Range of x values.
xmid	Midpoint of x values.

Methods Summary

xnorm(xinput, jump)	Convert input x coordinates to normalized coordinates suitable for input to special polynomials.
xy([xpos, ignore_jump])	Convert from a trace set to an array of x,y positions.

Attributes Documentation

has_jump
`True` if jump conditions are set.

nx
Number of x values.

xRange
Range of x values.

xmid
Midpoint of x values.

Methods Documentation

xnorm(*xinput*, *jump*)
Convert input x coordinates to normalized coordinates suitable for input to special polynomials.

Parameters

xinput
[array-like] Input coordinates.

jump
[`bool`] Set to `True` if there is a jump.

Returns

array-like
Normalized coordinates.

xy(*xpos=None*, *ignore_jump=False*)
Convert from a trace set to an array of x,y positions.

Parameters

xpos
[array-like, optional] If provided, evaluate the trace set at these positions. Otherwise the positions will be constructed from the trace set object iself.

ignore_jump
[`bool`, optional] If `True`, ignore any jump information in the `tset` object

Returns

:func:‘tuple‘ of array-like
The x, y positions.

Class Inheritance Diagram

1.2.13 pydl.pydlutils.yanny Module

This module corresponds to the yanny directory in idlutils.

This is a Python library for reading & writing yanny files.

yanny is an object-oriented interface to SDSS Parameter files (a.k.a. "FTCL" or "yanny" files) following these specifications. Parameter files typically have and can be recognized by the extension `.par`. These files may also be recognized if the first line of the file is:

```
#%yanny
```

This is not part of the standard specification, but it suffices to identify, *e.g.*, files that have not yet been written to disk, but only exist as file objects.

Because Parameter files can contain multiple tables, as well as metadata, there is no simple, one-to-one correspondence between these files and, say, an astropy `Table` object. Thus, `yanny` objects are implemented as a subclass of `OrderedDict` (to remember the order of keyword-value pairs), and the actual data are values accessed by keyword. Still, it is certainly possible to *write* a table-like object to a yanny file.

Given the caveats above, we have introduced *experimental* support for reading and writing yanny files directly to/from `Table` objects. Because of the experimental nature of this support, it must be activated "by hand" by including this snippet in your code:

```
from astropy.table import Table
from astropy.io.registry import (register_identifier, register_reader,
                                 register_writer)
from pydl.pydlutils.yanny import (is_yanny, read_table_yanny,
                                  write_table_yanny)

register_identifier('yanny', Table, is_yanny)
register_reader('yanny', Table, read_table_yanny)
register_writer('yanny', Table, write_table_yanny)
```

Currently multidimensional arrays are only supported for type `char`, and a close reading of the specifications indicates that multidimensional arrays were only ever intended to be supported for type `char`. So no multidimensional arrays, sorry.

Functions

is_yanny(origin, path, fileobj, *args, **kwargs)	Identifies Yanny files or objects.
read_table_yanny(filename[, tablename])	Read a yanny file into a Table.

Continued on next page

Table 29 – continued from previous page

write_ndarray_to_yanny(filename, datatables)	Converts a NumPy record array into a new FTCL/yanny file.
write_table_yanny(table, filename[, ...])	Write a Table to a yanny file.

is_yanny

pydl.pydlutils.yanny.**is_yanny**(*origin*, *path*, *fileobj*, **args*, ***kwargs*)

Identifies Yanny files or objects.

This function is for use with register_identifier().

Parameters

origin
[str] 'read' or 'write'

path
[str] Path to the file.

fileobj
[file object] Open file object, if available.

Returns

:class:'bool'
True if the file or object is a Yanny file.

read_table_yanny

pydl.pydlutils.yanny.**read_table_yanny**(*filename*, *tablename=None*)

Read a yanny file into a Table.

Because yanny files can contain multiple tables, it is necessary to specify the table name to return. However, all "headers" (keyword-value pairs) will be included in the Table metadata.

This function is for use with register_reader().

Parameters

filename
[str] Name of the file to read.

tablename
[str] The name of the table to read from the file.

Returns

:class:'~astropy.table.Table'
The table read from the file.

Raises

:exc:'PydlutilsException'
If tablename is not set.

:exc:‘KeyError‘
If `tablename` does not exist in the file.

write_ndarray_to_yanny

`pydl.pydlutils.yanny.`**`write_ndarray_to_yanny`**(*filename*, *datatables*, *structnames=None*, *enums=None*, *hdr=None*, *comments=None*)

Converts a NumPy record array into a new FTCL/yanny file.

Returns a new yanny object corresponding to the file.

Parameters

filename
[`str`] The name of a parameter file.

datatables
[`numpy.ndarray`, `numpy.recarray` or `list` of these.] A NumPy record array containing data that can be copied into a `yanny` object.

structnames
[`str` or `list` of `str`, optional] The name(s) to give the structure(s) in the yanny file. Defaults to ‘MYSTRUCT0’.

enums
[`dict`, optional] A dictionary containing enum information. See the documentation for the `dtype_to_struct()` method of the yanny object.

hdr
[`dict`, optional] A dictionary containing keyword/value pairs for the ‘header’ of the yanny file.

comments
[`str` or `list` of `str`, optional] A string containing comments that will be added to the start of the new file.

Returns

:class:‘yanny‘
The `yanny` object resulting from writing the file.

Raises

PydlutilsException
If `filename` already exists, or if the metadata are incorrect.

write_table_yanny

`pydl.pydlutils.yanny.`**`write_table_yanny`**(*table*, *filename*, *tablename=None*, *overwrite=False*)

Write a `Table` to a yanny file.

If `table` has any metadata, it will be written to the file as well.

This function is for use with `register_writer()`.

Parameters

table
[astropy.table.Table] The object to be written.

filename
[str] Name of the file to write to.

tablename
[str, optional] Name to give table within the file.

overwrite
[bool, optional] If True, any existing file will be silently overwritten.

Classes

yanny([filename, raw])	An object interface to a yanny file.

yanny

class pydl.pydlutils.yanny.**yanny**(*filename=None*, *raw=False*)
Bases: collections.OrderedDict

An object interface to a yanny file.

Create a yanny object using a yanny file, filename. If the file exists, it is read, & the dict structure of the object will be basically the same as that returned by read_yanny() in the efftickle package.

If the file does not exist, or if no filename is given, a blank structure is returned. Other methods allow for subsequent writing to the file.

Parameters

filename
[str or file-like, optional] The name of a yanny file or a file-like object representing a yanny file.

raw
[bool, optional] If True, data in a yanny file will *not* be converted into astropy Table objects, but will instead be retained as raw Python lists.

Attributes

raw
[bool] If True, data in a yanny file will *not* be converted into astropy Table objects, but will instead be retained as raw Python lists.

filename
[str] The name of a yanny parameter file. If a file-like object was used to initialize the object, this will have the value 'in_memory.par'.

_symbols
[dict] A dictionary containing the metadata describing the tables.

_contents
[str] The complete contents of a yanny parameter file.

_struct_type_caches
[dict] A dictionary of dictionaries, one dictionary for every structure definition in a yanny

parameter file. Contains the types of each column

_struct_isarray_caches
[dict] A dictionary of dictionaries, one dictionary for every structure definition in a yanny parameter file. Contains a boolean value for every column.

_enum_cache
[dict] Initially None, this attribute is initialized the first time the isenum() method is called. The keyword is the name of the enum type, the value is a list of the possible values of that type.

Create a yanny object using a yanny file.

Methods Summary

append(datatable)	Appends data to an existing FTCL/yanny file.
array_length(structure, variable)	Returns the length of an array type or 1 if the variable is not an array.
basetype(structure, variable)	Returns the bare type of a variable, stripping off any array information.
char_length(structure, variable)	Returns the length of a character field.
columns(table)	Returns an ordered list of column names associated with a particular table.
convert(structure, variable, value)	Converts value into the appropriate (Python) type.
dtype(structure)	Returns a NumPy dtype object suitable for describing a table as a record array.
dtype_to_struct(dt[, structname, enums])	Convert a NumPy dtype object describing a record array to a typedef struct statement.
get_token(string)	Removes the first 'word' from string.
isarray(structure, variable)	Returns True if the variable is an array type.
isenum(structure, variable)	Returns true if a variable is an enum type.
list_of_dicts(table)	Construct a list of dictionaries.
new_dict_from_pairs()	Returns a new dictionary of keyword/value pairs.
pairs()	Returns a list of keys to keyword/value pairs.
protect(x)	Used to appropriately quote string that might contain whitespace.
row(table, index)	Returns a list containing a single row from a specified table in column order.
size(table)	Returns the number of rows in a table.
tables()	Returns a list of all the defined structures.
trailing_comment(line)	Identify a trailing comment and strip it.
type(structure, variable)	Returns the type of a variable defined in a structure.
write([newfile, comments])	Write a yanny object to a file.

Methods Documentation

append(*datatable*)
Appends data to an existing FTCL/yanny file.

Tries as much as possible to preserve the ordering & format of the original file. The datatable should adhere to the format of the yanny object, but it is not necessary to reproduce the 'symbols' dictionary. It will not try to append data to a file that does not exist. If the append is successful, the data in the object will be updated.

Parameters

datatable
[dict] The data to append.

array_length(*structure*, *variable*)
Returns the length of an array type or 1 if the variable is not an array.

For character types, this is the length of a two-dimensional array, *e.g.*, char[5][20] has length 5.

Parameters

structure
[str] The name of the structure that contains variable.

variable
[str] The name of the column to check for array length.

Returns

:class:'int'
The length of the array variable

basetype(*structure*, *variable*)
Returns the bare type of a variable, stripping off any array information.

Parameters

structure
[str] The name of the structure that contains variable.

variable
[str] The name of the column whose type you want.

Returns

:class:'str'
The type of the variable, stripped of array information.

char_length(*structure*, *variable*)
Returns the length of a character field.

e.g. char[5][20] is an array of 5 strings of length 20. Returns None if the variable is not a character type. If the length is not specified, *i.e.* char[], it returns the length of the largest string.

Parameters

structure
[str] The name of the structure that contains variable.

variable
[str] The name of the column to check for char length.

Returns

:class:'int' or None
The length of the char variable.

columns(*table*)
Returns an ordered list of column names associated with a particular table.

The order is the same order as they are defined in the yanny file.

Parameters

table
[str] The table whose columns are desired.

Returns

:class:'list'
The list of column names.

convert(*structure*, *variable*, *value*)
Converts value into the appropriate (Python) type.

- short & int are converted to Python int.
- long is converted to Python long.
- float & double are converted to Python float.
- Other types are not altered.

There may be further conversions into NumPy types, but this is the first stage.

Parameters

structure
[str] The name of the structure that contains variable.

variable
[str] The name of the column undergoing conversion.

value
[str] The value contained in a particular row of variable.

Returns

:class:'int', :class:'long', :class:'float' or :class:'str'
value converted to a Python numerical type.

dtype(*structure*)
Returns a NumPy dtype object suitable for describing a table as a record array.

Treats enums as string, which is what the IDL reader does.

Parameters

structure
[str] The name of the structure.

Returns

:class:'numpy.dtype'
A dtype object suitable for describing the yanny structure as a record array.

static dtype_to_struct(*dt*, *structname='mystruct'*, *enums=None*)
Convert a NumPy dtype object describing a record array to a typedef struct statement.

The second argument is the name of the structure. If any of the columns are enum types, enums must be a dictionary with the keys the column names, and the values are a tuple containing the name of the enum type as the first item and a tuple or list of possible values as the second item.

Parameters

dt
[numpy.dtype] The dtype of a NumPy record array.

structname
[str, optional] The name to give the structure in the yanny file. Defaults to 'MYSTRUCT'.

enums
[dict, optional] A dictionary containing enum information. See details above.

Returns

:class:'dict'
A dictionary suitable for setting the 'symbols' dictionary of a new yanny object.

static get_token(*string*)
Removes the first 'word' from string.

If the 'word' is enclosed in double quotes, it returns the contents of the double quotes. If the 'word' is enclosed in braces, it returns the contents of the braces, but does not attempt to split the array. If the 'word' is the last word of the string, remainder is set equal to the empty string. This is basically a wrapper on some convenient regular expressions.

Parameters

string
[str] A string containing words.

Returns

:func:'tuple'
A tuple containing the first word and the remainder of the string.

Examples

```
>>> from pydl.pydlutils.yanny import yanny
>>> yanny.get_token("The quick brown fox")
('The', 'quick brown fox')
```

isarray(*structure*, *variable*)
Returns True if the variable is an array type.

For character types, this means a two-dimensional array, *e.g.*: char[5][20].

Parameters

structure
[str] The name of the structure that contains variable.

variable
[str] The name of the column to check for array type.

Returns

:class:'bool'
`True` if the variable is an array.

isenum(*structure*, *variable*)
Returns true if a variable is an enum type.

Parameters

structure
[str] The name of the structure that contains `variable`.

variable
[str] The name of the column to check for enum type.

Returns

:class:'bool'
`True` if the variable is enum type.

list_of_dicts(*table*)
Construct a list of dictionaries.

Takes a table from the yanny object and constructs a list object containing one row per entry. Each item in the list is a dictionary keyed by the struct value names.

If the yanny object instance is set up for NumPy record arrays, then the same functionality can be obtained with:

```
foo = par['TABLE'][0]['column']
```

Parameters

table
[str] The table to convert

Returns

:class:'list'
A list containing the rows of `table` converted to `dict`.

new_dict_from_pairs()
Returns a new dictionary of keyword/value pairs.

The new dictionary (*i.e.*, not a yanny object) contains the keys that `pairs()` returns. There are two reasons this is convenient:

- the key 'symbols' that is part of the yanny object will not be present
- a simple yanny file can be read with no further processing

Returns

:class:‘~collections.OrderedDict‘
A dictionary of the keyword-value pairs that remembers the order in which they were defined in the file.

Examples

Read a yanny file and return only the pairs:

```
>>> from os.path import dirname
>>> from pydl.pydlutils.yanny import yanny
>>> new_dict = yanny(dirname(__file__)+'/tests/t/test.par').new_dict_from_pairs()
>>> new_dict['mjd']
'54579'
>>> new_dict['alpha']
'beta gamma delta'
```

added: Demitri Muna, NYU 2009-04-28

pairs()
Returns a list of keys to keyword/value pairs.

Equivalent to doing `self.keys()`, but with all the data tables & other control structures stripped out.

static protect(*x*)
Used to appropriately quote string that might contain whitespace.

This method is mostly for internal use by the yanny object.

Parameters

x
[`str`] The data to protect.

Returns

:class:‘str‘
The data with white space protected by quotes.

Examples

```
>>> from pydl.pydlutils.yanny import yanny
>>> yanny.protect('This string contains whitespace.')
'"This string contains whitespace."'
>>> yanny.protect('This string contains a #hashtag.')
'"This string contains a #hashtag."'
```

row(*table*, *index*)
Returns a list containing a single row from a specified table in column order.

If index is out of range, it returns an empty list.

If the yanny object instance is set up for NumPy record arrays, then a single row can be obtained with:

```
row0 = par['TABLE'][0]
```

Parameters

table
[str] The table whose row is desired.

index
[int] The number of the row to return.

Returns

:class:'list'
A row from table.

size(*table*)
Returns the number of rows in a table.

Parameters

table
[str] The table whose size desired.

Returns

:class:'int'
The number of rows in table.

tables()
Returns a list of all the defined structures.

This is just the list of keys of the object with the 'internal' keys removed.

static trailing_comment(*line*)
Identify a trailing comment and strip it.

This routine works on the theory that a properly quoted comment mark will be surrounted by an odd number of double quotes, & we can skip to searching for the last one in the line.

Parameters

line
[str] A line from a yanny file potentially containing trailing comments.

Returns

:class:'str'
The line with any trailing comment and any residual white space trimmed off.

Notes

This may fail in certain pathological cases, for example if a real trailing comment contains a single double-quote:

```
# a 'pathological" trailing comment
```

or if someone is over-enthusiastically commenting:

```
# # # # # I like # characters.
```

Examples

```
>>> from pydl.pydlutils.yanny import yanny
>>> yanny.trailing_comment('mystruct 1234 "#hashtag" # a comment.')
'mystruct 1234 "#hashtag"'
>>> yanny.trailing_comment('mystruct 1234 "#hashtag" # a "comment".')
'mystruct 1234 "#hashtag"'
```

type(*structure*, *variable*)

Returns the type of a variable defined in a structure.

Returns None if the structure or the variable is undefined.

Parameters

structure
[str] The name of the structure that contains variable.

variable
[str] The name of the column whose type you want.

Returns

:class:'str'
The type of the variable.

write(*newfile=None*, *comments=None*)

Write a yanny object to a file.

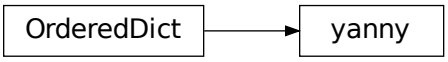
This assumes that the filename used to create the object was not that of a pre-existing file. If a file of the same name is detected, this method will *not* attempt to overwrite it, but will print a warning. This also assumes that the special 'symbols' key has been properly created. This will not necessarily make the file very human-readable, especially if the data lines are long. If the name of a new file is given, it will write to the new file (assuming it doesn't exist). If the writing is successful, the data in the object will be updated.

Parameters

newfile
[str, optional] The name of the file to write.

comments
[str or list of str, optional] Comments that will be placed at the head of the file. If a single string is passed, it will be written out verbatim, although a '#' character will be added if it does not already have one. If a list of strings is passed, comment characters will be added and the strings will be joined together.

Class Inheritance Diagram



CHAPTER 2

Goddard Utilities (`pydl.goddard`)

2.1 Introduction

This package provides functionality similar to the The IDL® Astronomy User's Libary, sometimes called the "Goddard Utilities", maintained by Wayne Landsman and distributed with idlutils.

In general, functions that are needed by `pydl.pydlutils` or `pydl.pydlspec2d` are implemented, while functions that are *not* needed have much lower priority.

The Goddard package is itself divided into a number of subpackages. Below we list the subpackages and the usability of the PyDL equivalent. The readiness levels are defined as:

Obsolete
: No point in implementing because the purpose of the code lapsed many years ago.

Not Applicable (NA)
: No point in implementing because another built-in or numpy/scipy/astropy package completely replaces this.

None
: Not (yet) implemented at all.

Rudimentary
: Only a few functions are implemented.

Fair
: Enough functions are implemented to be useful, but some are missing.

Good
: Pretty much anything you could do with the Goddard code you can do with the equivalent here.

Subpackage	Readiness Level	Comments
astro	Rudimentary	General astronomical utility functions.
astrom	NA	Tools for manipulating WCS data in FITS headers. Use `astropy.io.fits` and `astropy.wcs`.
coyote	NA	The Coyote library for plotting and graphics developed by David Fanning.
database	None	Allows access to IDL-specific databases.
disk_io	None	Provides access to IRAF image (`.imh`) files and AJ/ApJ-style tables.
fits	NA	Use `astropy.io.fits`.
fits_bintable	NA	Use `astropy.io.fits`.
fits_table	NA	Use `astropy.io.fits`.
idlphot	None	Adapted from an early version of DAOPHOT.
image	None	Generic image processing functions, including convolution/deconvolution.
jhuapl	None	Functions from the JHU Applied Physics Lab.
markwardt	None	Levenberg-Marquardt least-squares minimization.
math	Rudimentary	Generic mathematical functions. Many are implemented in numpy or scipy.
misc	Rudimentary	General utility functions that do not involve astronomy specifically.
plot	NA	Functions that supplement the built-in IDL plotting capabilities.
robust	None	Robust statistical fitting procedures.
sdas	None	Provides access to STDAS/GEIS image files.
sockets	NA	Functions for performing web queries in IDL. Use astroquery.
structure	NA	Tools for manipulating IDL data structures. Use `numpy.recarray`.
tv	NA	Functions for manipulating IDL image displays.

2.2 API

2.2.1 pydl.goddard Package

This subpackage contains the Goddard utilities.

2.2.2 pydl.goddard.astro Module

This module corresponds to the goddard/astro directory in idlutils.

Functions

`airtovac`(air)	Convert air wavelengths to wavelengths in vacuum.
`gcirc`(ra1, dec1, ra2, dec2[, units])	Computes rigorous great circle arc distances.
`get_juldate`([seconds])	Returns the current Julian date.
`get_juldate_main`()	Entry point for the get_juldate command-line script.
`vactoair`(vacuum)	Convert vacuum wavelengths to wavelengths in air.

airtovac

pydl.goddard.astro.**airtovac**(*air*)

Convert air wavelengths to wavelengths in vacuum.

Parameters

air
[array-like] Values of wavelength in air in Angstroms. `Quantity` objects with valid length dimensions will be internally converted to Angstrom.

Returns

array-like
Values of wavelength in vacuum in Angstroms. If a `Quantity` object was passed in, the output will be converted to the same units as the input.

Notes

- Formula from P. E. Ciddor, Applied Optics, 35, 1566 (1996).
- Values of wavelength below 2000 Å are not converted.

gcirc

pydl.goddard.astro.**gcirc**(*ra1*, *dec1*, *ra2*, *dec2*, *units=2*)
Computes rigorous great circle arc distances.

Parameters

ra1, dec1, ra2, dec2
[`float` or array-like] RA and Dec of two points.

units
[{ 0, 1, 2 }, optional]

- units = 0: everything is already in radians
- units = 1: RA in hours, dec in degrees, distance in arcsec.
- units = 2: RA, dec in degrees, distance in arcsec (default)

Returns

:class:'float' or array-like
The angular distance. Units of the value returned depend on the input value of `units`.

Notes

The formula below is the one best suited to handling small angular separations. See: http://en.wikipedia.org/wiki/Great-circle_distance

get_juldate

pydl.goddard.astro.**get_juldate**(*seconds=None*)
Returns the current Julian date.

Uses the MJD trick & adds the offset to get JD.

Parameters

seconds
[int or float, optional] Time in seconds since the UNIX epoch. This should only be used for testing.

Returns

:class:'float'
The Julian Day number as a floating point number.

Notes

Do not use this function if high precision is required, or if you are concerned about the distinction between UTC & TAI.

get_juldate_main

pydl.goddard.astro.**get_juldate_main**()
Entry point for the get_juldate command-line script.

vactoair

pydl.goddard.astro.**vactoair**(*vacuum*)
Convert vacuum wavelengths to wavelengths in air.

Parameters

vacuum
[array-like] Values of wavelength in vacuum in Angstroms. Quantity objects with valid length dimensions will be internally converted to Angstrom.

Returns

array-like
Values of wavelength in air in Angstroms. Quantity object was passed in, the output will be converted to the same units as the input.

Notes

- Formula from P. E. Ciddor, Applied Optics, 35, 1566 (1996).
- Values of wavelength below 2000 Å are not converted.

2.2.3 pydl.goddard.math Module

This module corresponds to the goddard/math directory in idlutils.

Functions

flegendre(x, m)	Compute the first m Legendre polynomials.

flegendre

pydl.goddard.math.**flegendre**(*x*, *m*)

Compute the first m Legendre polynomials.

Parameters

x
[array-like] Compute the Legendre polynomials at these abscissa values.

m
[int] The number of Legendre polynomials to compute. For example, if $m = 3$, $P_0(x)$, $P_1(x)$ and $P_2(x)$ will be computed.

Returns

:class:‘numpy.ndarray‘
The values of the Legendre functions.

2.2.4 pydl.goddard.misc Module

This module corresponds to the goddard/misc directory in idlutils.

Functions

cirrange(ang[, radians])	Convert an angle larger than 360 degrees to one less than 360 degrees.

cirrange

pydl.goddard.misc.**cirrange**(*ang*, *radians=False*)

Convert an angle larger than 360 degrees to one less than 360 degrees.

Parameters

ang
[float or array-like] Angle to convert. If the angle is in radians, the radians argument should be set.

radians
[class:bool, optional] If True, the input angle is in radians, and the output will be between zero and 2π.

Returns

:class:‘float‘ or array-like

Angle in the restricted range.

Examples

```
>>> from pydl.goddard.misc import cirrange
>>> cirrange(-270.0)
90.0
```

CHAPTER 3

SDSS Spectroscopic Data (`pydl.pydlspec2d`)

3.1 Introduction

This package provides functionality in the SDSS idlspec2d package. This package is used for processing and analyzing data from the SDSS optical spectrographs. The code is thus relevant to the SDSS Legacy, BOSS and eBOSS surveys. This package does *not* work with any infrared spectrograph data associated with the APOGEE-2 survey.

The primary *technical* focus of this particular implementation is the function `combine1fiber()`. This function is responsible for resampling 1D spectra onto a new wavelength solution. This allows for:

1. Shifting a spectrum from observed redshift to rest frame.
2. Coaddition of spectra of the same object, after resampling all spectra onto the same wavelength solution.

The primary *scientific* motivation of implementing `combine1fiber()` is to create template spectra based on curated spectra of, *e.g.*, luminous red galaxies (LRGs). Principal Component Analysis (PCA) or other techniques may be used to construct template spectra, but putting all spectra on the same rest-frame wavelength solution is the first step.

The idlspec2d package is itself divided into a number of subpackages. Below we list the subpackages and the usability of the PyDL equivalent. The readiness levels are defined as:

Obsolete
: No point in implementing because the purpose of the code lapsed many years ago.

Not Applicable (NA)
: No point in implementing because another built-in or numpy/scipy/astropy package completely replaces this.

None
: Not (yet) implemented at all.

Rudimentary
: Only a few functions are implemented.

Fair
: Enough functions are implemented to be useful, but some are missing.

Good
: Pretty much anything you could do with the idlspec2d code you can do with the equivalent here.

Subpackage	Readiness Level	Comments
apo2d	None	Quick extraction code for quality assurance at observation time.
config	None	Extraction pipeline configuration parameters in object-oriented IDL.
fluxfix	None	Flux calibration
guider	None	Interface to guider camera.
inspect	None	Tools for manual inspection of spectra.
photoz	Obsolete	Photometric redshifts for SDSS objects using spectral templates.
plan	None	Tools for planning exposures and recordings summaries of exposures.
plate	Obsolete	Tools for designing SDSS spectroscopic plates, especially for star clusters.
science	None	Code for science analysis of sets of 1D spectra.
spec1d	Fair	Tools for processing 1D spectra, including redshift fitting.
spec2d	Fair	Tools for extracting spectra from 2D images.
specdb	Obsolete	Tests on storing spectroscopic results in SQL databases.
specflat	None	Flat-fielding of spectroscopic 2D images.
templates	None	Tools for constructing spectroscopic templates.
testsuite	None	Tools for high-level quality assurance, *e.g.* comparing two reductions of the same data.

3.2 API

3.2.1 pydl.pydlspec2d Package

This subpackage implements functions from the idlspec2d package.

Classes

`Pydlspec2dException`	Exceptions raised by `pydl.pydlspec2d` that don't fit into a standard exception class like `ValueError`.
`Pydlspec2dUserWarning`	Class for warnings issued by `pydl.pydlspec2d`.

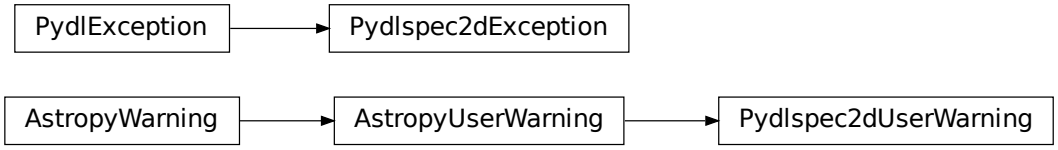
Pydlspec2dException

exception pydl.pydlspec2d.**Pydlspec2dException**
Exceptions raised by `pydl.pydlspec2d` that don't fit into a standard exception class like `ValueError`.

Pydlspec2dUserWarning

exception pydl.pydlspec2d.**Pydlspec2dUserWarning**
Class for warnings issued by `pydl.pydlspec2d`.

Class Inheritance Diagram



3.2.2 pydl.pydlspec2d.spec1d Module

This module corresponds to the spec1d directory in idlspec2d.

Functions

findspec(*args, **kwargs)	Find SDSS/BOSS spectra that match a given RA, Dec.
latest_mjd(plate, **kwargs)	Find the most recent MJD associated with a plate.
number_of_fibers(plate, **kwargs)	Returns the total number of fibers per plate.
pca_solve(newflux, newivar[, maxiter, ...])	Replacement for idlspec2d pca_solve.pro.
plot_eig(filename[, title])	Plot spectra from an eigenspectra/template file.
preprocess_spectra(flux, ivar[, loglam, ...])	Handle the processing of input spectra through the combine1fiber() stage.
readspec(platein[, mjd, fiber])	Read SDSS/BOSS spec2d & spec1d files.
skymask(invvar, andmask[, ormask, ngrow])	Mask regions where sky-subtraction errors are expected to dominate.
spec_append(spec1, spec2[, pixshift])	Append the array spec2 to the array spec1 & return a new array.
spec_path(plate[, path, topdir, run2d])	Return the directory containing spPlate files.
template_input(inputfile, dumpfile[, flux, ...])	Collect spectra and pass them to PCA or HMF solvers to compute spectral templates.
template_input_main()	Entry point for the compute_templates script.
template_metadata(inputfile[, verbose])	Read template metadata from file.
template_qso(metadata, newflux, newivar[, ...])	Run PCA or HMF on QSO spectra.
template_star(metadata, newloglam, newflux, ...)	Run PCA or HMF on stellar spectra of various classes.
wavevector(minfullwave, maxfullwave[, ...])	Return an array of wavelengths.

findspec

pydl.pydlspec2d.spec1d.**findspec**(**args*, ***kwargs*)

Find SDSS/BOSS spectra that match a given RA, Dec.

Parameters

ra, dec

[array-like, optional] If set, the first two positional arguments will be interpreted as RA,

Dec.

best
[bool, optional] If set, return only the best match for each input RA, Dec.

infile
[str, optional] If set, read RA, Dec data from this file.

outfile
[str, optional] If set, print match data to this file.

print
[bool, optional] If set, print the match data to the console.

run1d
[str, optional] Override the value of RUN1D.

run2d
[str, optional] Override the value of RUN2D.

sdss
[bool, optional] If set, search for SDSS-I/II spectra instead of BOSS spectra.

searchrad
[float, optional] Search for spectra in this radius around given RA, Dec. Default is 3 arcsec.

topdir
[str, optional] If set, override the value of SPECTRO_REDUX or BOSS_SPECTRO_REDUX.

Returns

:class:'dict'
A dictionary containing plate, MJD, fiber, etc.

latest_mjd

pydl.pydlspec2d.spec1d.**latest_mjd**(*plate*, ***kwargs*)
Find the most recent MJD associated with a plate.

Parameters

plate
[int or numpy.ndarray] The plate(s) to examine.

Returns

:class:'numpy.ndarray'
An array of MJD values for each plate.

number_of_fibers

pydl.pydlspec2d.spec1d.**number_of_fibers**(*plate*, ***kwargs*)
Returns the total number of fibers per plate.

Parameters

plate
[int or numpy.ndarray] The plate(s) to examine.

Returns

:class:‘numpy.ndarray‘
The number of fibers on each plate.

pca_solve

pydl.pydlspec2d.spec1d.**pca_solve**(*newflux*, *newivar*, *maxiter=0*, *niter=10*, *nkeep=3*, *nreturn=None*, *verbose=False*)
Replacement for idlspec2d pca_solve.pro.

Parameters

newflux
[array-like] The input spectral flux, assumed to have a common wavelength and redshift system.

newivar
[array-like] The inverse variance of the spectral flux.

maxiter
[int, optional] Stop PCA+reject iterations after this number.

niter
[int, optional] Stop PCA iterations after this number.

nkeep
[int, optional] Number of PCA components to keep.

nreturn
[int, optional] Number of PCA components to return, usually the same as nkeep.

verbose
[bool, optional] If True, print extra information.

Returns

:class:‘dict‘
The PCA solution.

plot_eig

pydl.pydlspec2d.spec1d.**plot_eig**(*filename*, *title='Unknown'*)
Plot spectra from an eigenspectra/template file.

Parameters

filename
[str] Name of a FITS file containing eigenspectra/templates.

title
[str, optional] Title to put on the plot.

Raises

:exc:‘ValueError‘
If an unknown template type was input in `filename`.

preprocess_spectra

pydl.pydlspec2d.spec1d.**preprocess_spectra**(*flux*, *ivar*, *loglam=None*, *zfit=None*, *aesthetics='mean'*, *newloglam=None*, *wavemin=None*, *wavemax=None*, *verbose=False*)

Handle the processing of input spectra through the `combine1fiber()` stage.

Parameters

flux
[array-like] The input spectral flux.

ivar
[array-like] The inverse variance of the spectral flux.

loglam
[array-like, optional] The input wavelength solution.

zfit
[array-like, optional] The redshift of each input spectrum.

aesthetics
[`str`, optional] This parameter will be passed to `combine1fiber()`.

newloglam
[array-like, optional] The output wavelength solution.

wavemin
[`float`, optional] Minimum wavelength if `newloglam` is not specified.

wavemax
[`float`, optional] Maximum wavelength if `newloglam` is not specified.

verbose
[`bool`, optional] If `True`, print extra information.

Returns

:func:‘tuple‘ of :class:‘numpy.ndarray‘
The resampled flux, inverse variance and wavelength solution, respectively.

readspec

pydl.pydlspec2d.spec1d.**readspec**(*platein*, *mjd=None*, *fiber=None*, ***kwargs*)

Read SDSS/BOSS spec2d & spec1d files.

Parameters

platein
[`int` or `numpy.ndarray`] Plate number(s).

mjd
[int or numpy.ndarray, optional] MJD numbers. If not provided, they will be calculated by latest_mjd().

fiber
[array-like, optional] Fibers to read. If not set, all fibers from all plates will be returned.

topdir
[str, optional] Override the value of BOSS_SPECTRO_REDUX.

run2d
[str, optional] Override the value of RUN2D.

run1d
[str, optional] Override the value of RUN1D.

path
[str, optional] Override all path information with this directory name.

align
[bool, optional] If set, align all the spectra in wavelength.

znum
[int, optional] If set, return the znum-th best fit reshift fit, instead of the best.

Returns

:class:'dict'
A dictionary containing the data read.

skymask

pydl.pydlspec2d.spec1d.**skymask**(*invvar*, *andmask*, *ormask=None*, *ngrow=2*)
Mask regions where sky-subtraction errors are expected to dominate.

Parameters

invvar
[numpy.ndarray] Inverse variance.

andmask
[numpy.ndarray] An "and" mask. For historical reasons, this input is ignored.

ormask
[numpy.ndarray, optional] An "or" mask. Although technically this is optional, if it is not supplied, this function will have no effect.

ngrow
[int, optional] Expand bad areas by this number of pixels.

Returns

:class:'numpy.ndarray'
The invvar multiplied by the bad areas.

spec_append

pydl.pydlspec2d.spec1d.**spec_append**(*spec1*, *spec2*, *pixshift=0*)

Append the array spec2 to the array spec1 & return a new array.

If the dimension of these arrays is the same, then append as [spec1,spec2]. If not, increase the size of the smaller array & fill with zeros.

Parameters

spec1, spec2
[numpy.ndarray] Append `spec2` to `spec1`.

pixshift
[int, optional] If `pixshift` is set to a positive integer, `spec2` will be padded with `pixshift` zeros on the left side. If `pixshift` is set to a negative integer, `spec1` will be padded with `abs(pixshift)` zeros on the left side. If not set, all zeros will be padded on the right side.

Returns

:class:'numpy.ndarray'
A new array containing both `spec1` and `spec2`.

spec_path

pydl.pydlspec2d.spec1d.**spec_path**(*plate*, *path=None*, *topdir=None*, *run2d=None*)

Return the directory containing spPlate files.

Parameters

plate
[int or numpy.ndarray] The plate(s) to examine.

path
[str, optional] If set, `path` becomes the full path for every plate. In other words, it completely short-circuits this function.

topdir
[str, optional] Used to override the value of `BOSS_SPECTRO_REDUX`.

run2d
[str, optional] Used to override the value of `RUN2D`.

Returns

:class:'list'
A list of directories, one for each plate.

Raises

:exc:'KeyError'
If environment variables are not supplied.

template_input

pydl.pydlspec2d.spec1d.**template_input**(*inputfile*, *dumpfile*, *flux=False*, *verbose=False*)
Collect spectra and pass them to PCA or HMF solvers to compute spectral templates.

This function replaces the various `PCA_GAL()`, `PCA_STAR()`, etc., functions from idlspec2d.

Parameters

inputfile
[`str`] Name of a Parameter file containing the input data and metadata.

dumpfile
[`str`] Name of a Pickle file used to store intermediate data.

flux
[`bool`, optional] If `True`, plot the individual input spectra.

verbose
[`bool`, optional] If `True`, print lots of extra information.

template_input_main

pydl.pydlspec2d.spec1d.**template_input_main**()
Entry point for the compute_templates script.

Returns

:class:'int'
An integer suitable for passing to `sys.exit()`.

template_metadata

pydl.pydlspec2d.spec1d.**template_metadata**(*inputfile*, *verbose=False*)
Read template metadata from file.

Parameters

inputfile
[`str`] Name of a Parameter file containing the input data and metadata.

verbose
[`bool`, optional] If `True`, print lots of extra information.

Returns

:func:'tuple'
A tuple containing the list of input spectra and a dictionary containing other metadata.

template_qso

pydl.pydlspec2d.spec1d.**template_qso**(*metadata*, *newflux*, *newivar*, *verbose=False*)
Run PCA or HMF on QSO spectra.

Historically, QSO templates were comptuted one at a time instead of all at once.

Parameters

metadata
[dict] Dictionary containing metadata about the spectra.

newflux
[ndarray] Flux shifted onto common wavelength.

newivar
[ndarray] Inverse variances of the fluxes.

verbose
[bool, optional] If True, print lots of extra information.

Returns

:class:'dict'
A dictonary containing flux, eigenvalues, etc.

template_star

pydl.pydlspec2d.spec1d.**template_star**(*metadata*, *newloglam*, *newflux*, *newivar*, *slist*, *outfile*, *verbose=False*)

Run PCA or HMF on stellar spectra of various classes.

Parameters

metadata
[dict] Dictionary containing metadata about the spectra.

newloglam
[ndarray] The wavelength array, used only for plots.

newflux
[ndarray] Flux shifted onto common wavelength.

newivar
[ndarray] Inverse variances of the fluxes.

slist
[recarray] The list of objects, containing stellar class information.

outfile
[str] The base name of output file, used for plots.

verbose
[bool, optional] If True, print lots of extra information.

Returns

:class:'dict'
A dictonary containing flux, eigenvalues, etc.

wavevector

pydl.pydlspec2d.spec1d.**wavevector**(*minfullwave*, *maxfullwave*, *zeropoint=3.5*, *binsz=0.0001*, *wavemin=None*)

Return an array of wavelengths.

Parameters

minfullwave
[float] Minimum wavelength.

maxfullwave
[float] Maximum wavelength.

zeropoint
[float, optional] Offset of the input wavelength values.

binsz
[float, optional] Separation between wavelength values.

wavemin
[float, optional] If this is set the values of `minfullwave` and `zeropoint` are ignored.

Returns

:class:'numpy.ndarray'
Depending on the values of `minfullwave`, `binsz`, etc., the resulting array could be interpreted as an array of wavelengths or an array of log(wavelength).

Classes

HMF(spectra, invvar[, K, n_iter, seed, ...])	Class used to manage data for Heteroscedastic Matrix Factorization (HMF).

HMF

class pydl.pydlspec2d.spec1d.**HMF**(*spectra*, *invvar*, *K=4*, *n_iter=None*, *seed=None*, *nonnegative=False*, *epsilon=None*, *verbose=False*)

Bases: object

Class used to manage data for Heteroscedastic Matrix Factorization (HMF).

This is a replacement for pca_solve(). It can be called with:

```
hmf = HMF(spectra, invvar)
output = hmf.solve()
```

The input spectra should be pre-processed through combine1fiber().

Parameters

spectra
[array-like] The input spectral flux, assumed to have a common wavelength and redshift system.

invvar
[array-like] The inverse variance of the spectral flux.

K
[int, optional] The number of dimensions of the factorization (default 4).

n_iter
[int, optional] Number of iterations.

seed
[int, optional.] If set, pass this value to numpy.random.seed().

nonnegative
[bool, optional] Set this to True to perform nonnegative HMF.

epsilon
[float, optional] Regularization parameter. Set to any non-negative float value to turn it on.

verbose
[bool, optional] If True, print extra information.

Notes

See [1] and [2] for the original derivation of this method.

The HMF iteration is initialized using kmeans(), which itself uses random numbers to initialize its state. If you need to ensure reproducibility, call numpy.random.seed() before initializing HMF.

The current algorithm cannot handle input data that contain *columns* of zeros. Columns of this type need to be *carefully* removed from the input data. This could also result in the output data having a different size compared to the input data.

References

[1], [2]

Methods Summary

astep()	Update for coefficients at fixed component spectra.
astepnn()	Non-negative update for coefficients at fixed component spectra.
badness()	Compute χ^2, including possible non-smoothness penalty.
chi()	Compute χ, the scaled residual.
gstep()	Update for component spectra at fixed coefficients.
gstepnn()	Non-negative update for component spectra at fixed coefficients.
iterate()	Handle the HMF iteration.
model()	Compute the model.
normbase()	Apply standard component normalization.
penalty()	Compute penalty for non-smoothness.
reorder()	Reorder and rotate basis analogous to PCA.
resid()	Compute residuals.
solve()	Process the inputs.

Methods Documentation

astep()
Update for coefficients at fixed component spectra.

astepnn()
Non-negative update for coefficients at fixed component spectra.

badness()
Compute χ^2, including possible non-smoothness penalty.

chi()
Compute χ, the scaled residual.

gstep()
Update for component spectra at fixed coefficients.

gstepnn()
Non-negative update for component spectra at fixed coefficients.

iterate()
Handle the HMF iteration.

Returns

:func:‘tuple‘ of :class:‘numpy.ndarray‘
The fitting coefficients and fitted functions, respectively.

model()
Compute the model.

normbase()
Apply standard component normalization.

penalty()
Compute penalty for non-smoothness.

reorder()
Reorder and rotate basis analogous to PCA.

resid()
Compute residuals.

solve()
Process the inputs.

Returns

:class:‘dict‘
The HMF solution.

Class Inheritance Diagram

3.2.3 pydl.pydlspec2d.spec2d Module

This module corresponds to the spec2d directory in idlspec2d.

Functions

aesthetics(flux, invvar[, method])	Add nice values to a spectrum where it is masked.
combine1fiber(inloglam, objflux, newloglam)	Combine several spectra of the same object, or resample a single spectrum.
filter_thru(flux[, waveimg, wset, mask, ...])	Compute throughput in SDSS filters.

aesthetics

pydl.pydlspec2d.spec2d.**aesthetics**(*flux*, *invvar*, *method='traditional'*)

Add nice values to a spectrum where it is masked.

Parameters

flux
[numpy.ndarray] The spectrum to clean up.

invvar
[numpy.ndarray] Inverse variance of the spectrum.

method
[{ 'traditional', 'noconst', 'mean', 'damp', 'nothing' }, optional] Apply this method to clean up the spectrum. Default is 'traditional'.

Returns

:class:'numpy.ndarray'
A cleaned-up spectrum.

combine1fiber

pydl.pydlspec2d.spec2d.**combine1fiber**(*inloglam*, *objflux*, *newloglam*, *objivar=None*, *verbose=False*, ***kwargs*)

Combine several spectra of the same object, or resample a single spectrum.

Parameters

inloglam
[numpy.ndarray] Vector of log wavelength.

objflux
[numpy.ndarray] Input flux.

newloglam
[numpy.ndarray] Output wavelength pixels, vector of log wavelength.

objivar
[numpy.ndarray, optional] Inverse variance of the flux.

verbose
[bool, optional] If True, set log level to DEBUG.

Returns

:func:'tuple' of :class:'numpy.ndarray'
The resulting flux and inverse variance.

Raises

:exc:'ValueError'
If input dimensions don't match.

filter_thru

pydl.pydlspec2d.spec2d.**filter_thru**(*flux*, *waveimg=None*, *wset=None*, *mask=None*, *filter_prefix='sdss_jun2001'*, *toair=False*)

Compute throughput in SDSS filters.

Parameters

flux
[array-like] Spectral flux.

waveimg
[array-like, optional] Full wavelength solution, with the same shape as flux.

wset
[TraceSet, optional] A trace set containing the wavelength solution. Must be specified if waveimg is not specified.

mask
[array-like, optional] Interpolate over pixels where mask is non-zero.

filter_prefix
[str, optional] Specifies a set of filter curves.

toair
[bool, optional] If True, convert the wavelengths to air from vacuum before computing.

Returns

array-like
Integrated flux in the filter bands.

Raises

:exc:‘ValueError‘
If neither `waveimg` nor `wset` are set.

CHAPTER 4

SDSS Imaging Data (`pydl.photoop`)

4.1 Introduction

The photoop package is used to process SDSS imaging data. This package is used to reduce, resolve and flux-calibrate the SDSS raw data, resulting in both flux-calibrated images and catalogs.

SDSS ceased taking imaging data in 2009, and there has only been one full processing of the imaging data since then, although adjustments have been made to the astrometry and flux calibration.

The primary emphasis of this implementation is:

1. Functions related to the SDSS photometric `objID`, which is a unique integer used in SDSS databases, constructed from quantities that specify a particular astronomical object on a particular image.
2. Functions related to finding SDSS photometric data on disk.
3. The SDSS "window" function, which defines what parts of the sky are covered by SDSS images. This can be used in conjunction with the `mangle` module to find points and regions that have SDSS imaging.
4. In general, functions that work with existing imaging data, rather than functions to reduce the data.

The photoop package is itself divided into a number of subpackages. Below we list the subpackages and the usability of the PyDL equivalent. The readiness levels are defined as:

Obsolete
: No point in implementing because the purpose of the code lapsed many years ago.

Not Applicable (NA)
: No point in implementing because another built-in or numpy/scipy/astropy package completely replaces this.

None
: Not (yet) implemented at all.

Rudimentary
: Only a few functions are implemented.

Fair
: Enough functions are implemented to be useful, but some are missing.

Good
Pretty much anything you could do with the photoop code you can do with the equivalent here.

Sub-package	Readiness Level	Comments
apache	Obsolete	Processing of "Apache Wheel" images, used for calibration.
astrom	None	Astrometry for SDSS images.
atlas	None	Construction of "atlas" images, small cutouts of individual objects.
bluetip	None	Tools for "blue-tip" photometry and extinction estimation.
compare	Obsolete	Compare the same images in two different data reductions.
database	Obsolete	Experimental database loading code.
flats	None	Analysis of flat-field files.
hoggpipe	Obsolete	Another version of "Apache Wheel" processing code.
image	None	Tools for creating "corrected frame" images. These are flux-calibrated and sky-subtracted images with physical flux units.
ircam	Obsolete	Tools for processing all-sky "cloud camera" images, used to establish photometricity.
match	None	Tools for matching SDSS spectra to corresponding photometric objects.
misc	None	Code with no obvious home in any other category.
pcalib	None	Tools related to "ubercalibration".
photoobj	Rudimentary	Tools for creating calibrated catalogs from images.
plan	Obsolete	Tools for planning photometric reductions.
plots	None	Plots for high-level quality assurance.
psf	None	Analysis of point-spread functions.
ptcalib	Obsolete	Processing of "photometric telescope" data, an obsolete technique for flux-calibration.
resolve	None	Code for the resolve stage of image processing.
sdss3_runqa	Obsolete	Quality assurance tests from the most recent photometric reduction.
sdssio	Rudimentary	Tools for reading and writing various data files produced by the photometric reductions.
window	Rudimentary	Tools for determining the sky coverage of the survey.

4.2 API

4.2.1 pydl.photoop Package

This subpackage implements functions from the photoop package.

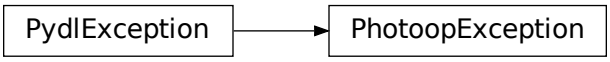
Classes

PhotoopException	Exceptions raised by pydl.photoop that don't fit into a standard exception class like ValueError.

PhotoopException

exception pydl.photoop.**PhotoopException**
Exceptions raised by pydl.photoop that don't fit into a standard exception class like ValueError.

Class Inheritance Diagram



4.2.2 pydl.photoop.photoobj Module

This module corresponds to the photoobj directory in photoop.

Functions

sdss_calibv()	Return calibration for velocities from pix/frame to deg/day.
unwrap_objid(objid)	Unwrap CAS-style objID into run, camcol, field, id, rerun.

sdss_calibv

pydl.photoop.photoobj.**sdss_calibv**()
Return calibration for velocities from pix/frame to deg/day.

Returns

:class:‘~astropy.units.quanity.Quantity‘
The conversion from pixels per frame to degrees per day

Notes

Assumes frame time difference of 71.72 seconds and pixel scale of 0.396 arcsec, both fixed. Also note that observations of the same part of sky from adjacent bands are separated by *two* frame numbers, so we multiply by a factor two.

unwrap_objid

pydl.photoop.photoobj.**unwrap_objid**(*objid*)
Unwrap CAS-style objID into run, camcol, field, id, rerun.

See sdss_objid() for details on how the bits within an objID are assigned.

Parameters

objid
[numpy.ndarray] An array containing 64-bit integers or strings. If strings are passed, they will be converted to integers internally.

Returns

:class:‘numpy.recarray‘
A record array with the same length as objid, with the columns ‘skyversion’, ‘rerun’, ‘run’, ‘camcol’, ‘firstfield’, ‘frame’, ‘id’.

Raises

:exc:‘ValueError‘
If the input objID has a type that can’t be converted into 64-bit integer.

Notes

For historical reasons, the inverse of this function, sdss_objid() is not in the same namespace as this function.

‘frame’ is used instead of ‘field’ because record arrays have a method of the same name.

Examples

```
>>> from numpy import array
>>> from pydl.photoop.photoobj import unwrap_objid
>>> unwrap_objid(array([1237661382772195474]))
rec.array([(2, 301, 3704, 3, 0, 91, 146)],
      dtype=[('skyversion', '<i4'), ('rerun', '<i4'), ('run', '<i4'), ('camcol', '<i4'), (
↪'firstfield', '<i4'), ('frame', '<i4'), ('id', '<i4')])
```

4.2.3 pydl.photoop.sdssio Module

This module corresponds to the sdssio directory of photoop.

Functions

filtername(f)	Return the name of a filter given its number.
filternum([filt])	Return index number for SDSS filters either from a number or name.
sdss_calib(run, camcol, field[, rerun])	Read photometric calibration solutions from calibPhotom or calibPhotomGlobal files.
sdss_name(ftype, run, camcol, field[, ...])	Return the name of an SDSS data file including path.
sdss_path(ftype, run[, camcol, rerun])	Return the path name for SDSS data assuming SAS directory structure.
sdssflux2ab(flux[, magnitude, ivar])	Convert the SDSS calibrated fluxes (magnitudes) into AB fluxes (magnitudes).

filtername

pydl.photoop.sdssio.**filtername**(*f*)
Return the name of a filter given its number.

Parameters

f
[int] The filter number.

Returns

:class:'str'
The corresponding filter name.

Examples

```
>>> filtername(0)
'u'
```

filternum

pydl.photoop.sdssio.**filternum**(*filt='foo'*)
Return index number for SDSS filters either from a number or name.

Parameters

filt
[str] The filter name.

Returns

:class:'int'
The corresponding filter number

Raises

:exc:'KeyError'
If `filt` is not a valid filter name.

Examples

```
>>> filternum('g')
1
```

sdss_calib

pydl.photoop.sdssio.**sdss_calib**(*run*, *camcol*, *field*, *rerun=''*, ***kwargs*)
Read photometric calibration solutions from calibPhotom or calibPhotomGlobal files.

Parameters

run
[int] Photo run number

camcol
[int] Camcol number

field
[int] Field number

rerun
[str, optional] Photometric reduction number, as a string.

Returns

:class:'dict'
A dictionary containing the 'NMGYPERCOUNT' keyword.

Notes

Currently, this is just a placeholder.

sdss_name

pydl.photoop.sdssio.**sdss_name**(*ftype*, *run*, *camcol*, *field*, *rerun=''*, *thisfilter='r'*, *no_path=False*)
Return the name of an SDSS data file including path.

Parameters

ftype
[str] The general type of the file, for example `'reObj'`

run
[int] The run number.

camcol
[int] The camcol number.

field
[int] The field number

rerun
[str, optional] If necessary, set the rerun name using this argument.

thisfilter
[int or str, optional] If necessary, set the filter using this argument.

no_path
[bool, optional] Normally, sdss_name returns the full path. If `no_path` is `True`, only the basename of the file is returned.

Returns

:class:'str'
The full file name, normally including the full path.

Raises

:exc:‘KeyError‘
If the file type is unknown.

sdss_path

pydl.photoop.sdssio.**sdss_path**(*ftype*, *run*, *camcol=0*, *rerun=”*)
Return the path name for SDSS data assuming SAS directory structure.

Parameters

ftype
[str] The general type of the file, for example `'reObj'`

run
[int] The run number.

camcol
[int, optional] If necessary, set the camcol number using this argument.

rerun
[str, optional] If necessary, set the rerun name using this argument.

Returns

:class:‘str‘
The directory in which file `ftype` lives.

Raises

:exc:‘KeyError‘
If the file type is unknown.

sdssflux2ab

pydl.photoop.sdssio.**sdssflux2ab**(*flux*, *magnitude=False*, *ivar=False*)
Convert the SDSS calibrated fluxes (magnitudes) into AB fluxes (magnitudes).

Parameters

flux
[numpy.ndarray] Array of calibrated fluxes or SDSS magnitudes with 5 columns, corresponding to the 5 filters *u*, *g*, *r*, *i*, *z*.

magnitude
[bool, optional] If set to `True`, then assume `flux` are SDSS magnitudes instead of linear flux units.

ivar
[numpy.ndarray, optional] If set, the input fluxes are actually inverse variances.

Returns

:class:‘numpy.ndarray‘
Array of fluxes or magnitudes on the AB system.

Notes

Uses the conversions posted by D.Hogg (sdss-calib/845):

```
u(AB,2.5m) = u(2.5m) - 0.042
g(AB,2.5m) = g(2.5m) + 0.036
r(AB,2.5m) = r(2.5m) + 0.015
i(AB,2.5m) = i(2.5m) + 0.013
z(AB,2.5m) = z(2.5m) - 0.002
```

4.2.4 pydl.photoop.window Module

This module corresponds to the window directory in photoop.

Functions

sdss_score(flist[, silent])	Score a list of imaging fields from zero to one.
window_read(**kwargs)	Read window files in $PHOTO_RESOLVE.
window_score(**kwargs)	For uber-resolve, score all the fields from zero to one.

sdss_score

pydl.photoop.window.**sdss_score**(*flist*, *silent=True*)
Score a list of imaging fields from zero to one.

Parameters

flist
[HDUList] Opened FITS file.

silent
[bool, optional] If False, print extra information.

Returns

:class:‘numpy.ndarray‘
A vector of scores, one for each row of the FITS file.

window_read

pydl.photoop.window.**window_read**(***kwargs*)
Read window files in $PHOTO_RESOLVE.

window_score

pydl.photoop.window.**window_score**(***kwargs*)

For uber-resolve, score all the fields from zero to one.

If 'rescore' is set, then write a new file 'window_flist_rescore.fits' rather than over-writing the file 'window_flist.fits'

Part III

Other Notes

CHAPTER 5

SDSS Spectroscopic Templates with PyDL

One of the original motivations for creating the PyDL package was to reproduce and, ultimately, improve the method for generating spectroscopic templates for the Sloan Digital Sky Survey (SDSS). The spectroscopic templates are used by the "1D" portion of the SDSS spectroscopic pipeline (Bolton *et al.* 2012[1]; See also `pydl.pydlspec2d`). Historically, those were generated by:

1. Selecting individual objects (galaxies, QSOs, stars, white dwarfs, etc.) with good SDSS spectra and reliable classifications.
2. For a given set of objects, resample the spectra onto a common wavelength grid and redshift to rest frame.
3. Perform an iterative PCA analysis on the flux versus wavelength matrix.
4. Select the most significant eigenspectra as the templates. Typically the first four eigenspectra were chosen.

For SDSS-I/II, the input spectra were all high signal-to-noise, but as the BOSS survey in SDSS-III started to target more distant galaxies, there was interest in improving the robustness of the PCA fitting to handle low signal-to-noise inputs. In addition, full alternatives to PCA, such has Heteroscedastic Matrix Factorization (HMF)[2],[3] could be compared.

Here are the algorithmic steps taken in the code to implement the procedure described above.

1. `template_input()` reads a configuration file that includes a list of individual spectra. There is one configuration file for each type of object (galaxy, QSO, etc.).
2. `readspec()` reads each spectrum. Masked areas are identified by setting the per-spectrum inverse variance to zero as needed.
3. A new wavelength grid is created. SDSS typically uses wavelength grids that are evenly spaced in $\log_{10}$ wavelength.
4. `preprocess_spectra()` calls `combine1fiber()` to perform the resampling and rest frame shift.
5. `pca_solve()` or `HMF` are used to obtain the eigenspectra.
6. The eigenspectra are written to a FITS file.

[1] Bolton, Adam, et al. 2012 AJ 144, 144.
[2] Tsalmantza, P., Decarli, R., Dotti, M., Hogg, D. W., 2011 ApJ 738, 20
[3] Tsalmantza, P., Hogg, D. W., 2012 ApJ 753, 122

At least historically, there was no test data set on which to (unit) test these algorithms. The best one could do was to take the inputs and compare them, function-by-function, to the equivalent IDL® code, ensuring that the results were the same, to some numerical precision.

CHAPTER 6

PyDL Changelog

6.1 1.0.0 (unreleased)

This version will only support Python 3 and Astropy 3. This release is planned for early 2019.

6.2 0.7.0 (2019-02-22)

This version is planned to be the last version with Python 2 support.

- Support the `firstField` bit in ObjIDs from DR7 and earlier (Issue #37).
- Change tests of Astropy development version from Python 2 to Python 3.
- Update to astropy_helpers/v2.0.6 (PR #40).
- Add `astropy.units` support to `airtovac()` and `vactoair()` (PR #41).
- Change Exelis to Harris Geospatial (PR #42).
- Fix `FutureWarning` in `re` in Python 3.7 due to nested sets (PR #44).
- Use `six` instead of `astropy.extern.six` (PR #48).
- Update `_astropy_init.py` (PR #47 via PR #48).
- Update astropy_helpers/v2.0.8 (PR #45 via PR #48).

6.3 0.6.0 (2017-09-19)

- This release is compatible with Astropy 2.0, and may be backwards incompatible with astropy v1.x.
- Update to astropy_helpers/v2.0.1.
- Use standard library `argparse` (Issue #31).

- Use the new `astropy.coordinates.Attribute` class.
- Fix typo (PR #26).

6.4 0.5.4 (2017-05-04)

- Added `sdss_specobjid()` to compute SDSS specObjIDs, and its inverse function `unwrap_specobjid()`.
- Update to astropy_helpers/v1.3.1.
- Refactor HMF code into an object to contain the data and methods.
- Use functions from `astropy.utils.data` where possible.
- Fix an integer division error encountered when using Numpy 1.12 (Issue #19).
- Fixed tests that were failing on 32-bit platforms *and* Python 3.5 (Issue #20).

6.5 0.5.3 (2016-12-03)

- Fixed formatting of TODO document.
- Fixed tests that were failing on 32-bit platforms (Issue #14).
- Use temporary files so that tests can run when astropy is installed read-only (*e.g.*, with **`pip`**; Issue #16)

6.6 0.5.2 (2016-08-04)

- Changes in how Mangle-polygon containing FITS files are handled, related to Issue #11.
- Fixed memory leak in `combine1fiber()`, see Issue #12.
- Added `is_in_window()`.
- Allow polygon area functions to deal with negative caps and `use_caps`.
- Update `docs/conf.py` for Python 3.5 compatibility (PR #13).

6.7 0.5.1 (2016-06-22)

- Removed unnecessary `from __future__` import in `pydl.pydlspec2d.spec1d`.
- Ongoing documentation upgrades.
- Update some links that needed to be transitioned from SDSS-III to SDSS-IV.
- Upgrade to astropy_helpers/v1.2.
- Update to latest version of package-template.
- Disabled tests on Python 3.3; enabled tests on Python 3.5
- Fix Issue #8; Issue #9.
- Add warnings about incomplete Mangle functions.

6.8 0.5.0 (2016-05-01)

- Dropped support for Python 2.6. Python 2.6 does not contain `collections.OrderedDict`, which is needed to support `yanny` objects, and at this point it is not worth going to the trouble to support this with an external package.
- Ongoing review and upgrade of docstrings.
- Yanny files can now be converted into *genuine* NumPy `record arrays`; previously, the conversion was only to `numpy.ndarray` with named columns, which is a slightly different thing.
- Added additional tests on `yanny` objects.
- Experimental support for interconversion of `yanny` objects and `Table` objects.
- Improving PEP 8 compliance
- Restructuing sub-packages to reduce the number of files.
- Improvements to spectral template processing code, deduplicated some code.
- Support platform-independent home directory (PR #7).
- Uppercase the package name (in documentation only).
- Upgrade to astropy_helpers/v1.1.1.
- Add functions from the idlutils rgbcolor directory.
- `spec_path()` can now find SDSS spectra, not just BOSS.

6.9 0.4.1 (2015-09-22)

- No changes at all from 0.4.0. This tag only exists because of a botched PyPI upload.

6.10 0.4.0 (2015-09-22)

- Use astropy_helpers/v1.0.3, package-template/v1.0.
- Remove some old FITS code that `astropy.io.fits` makes moot.
- Remove code for command-line scripts. These are now auto-generated by the "entry_point" method.
- Remove Python/3.2 tests.
- Improved test coverage.
- Fixed problem with the `spheregroup` code.
- Removed some code that is 100% redundant with astropy (*e.g.* `angles_to_xyz()`).
- Fixed bug in `set_use_caps()` that was discovered on the IDL side.
- Updated documentation of `read_fits_polygons()`.
- Added cross-references to classes, functions, etc.

6.11 0.3.0 (2015-02-20)

- Use astropy_helpers/v0.4.3, package-template/v0.4.1.
- Avoided (but did not fix) a bug in `chunks` that occurs when operating on a list of coordinates of length 1.
- Fixed a typo in `bspline`, added documentation.
- Simplify documentation files.
- `sdss_flagname()` now accepts more types of numeric input.
- Added *Authors and Credits* file.

6.12 0.2.3 (2014-07-22)

- Added `pydl.photoop.window`.
- Added stub `sdss_calib()`, updated `sdss_score()`.
- Added `unwrap_objid()`.
- Merged pull request #4, fixing some Python3 issues.

6.13 0.2.2 (2014-05-07)

- Updated to latest package-template version.
- Added ability to write multiple ndarray to yanny files.
- Fixed `struct_print()` test for older Numpy versions.
- Fixed failing yanny file test.
- Improve test infrastructure, including Travis builds.
- Allow comment characters inside quoted strings in yanny files.

6.14 0.2.1 (2013-10-06)

- Added `sdss_sweep_circle()`.
- Added first few `pydl.photoop` functions.
- Clean up some import statements.

6.15 0.2.0 (2013-04-22)

- Using the astropy package-template to bring pydl into astropy-compatible form.
- Some but not all tests are re-implemented.

6.16 0.1.1 (2013-03-06)

- Creating a tag representing the state immediately after creation of the git repository.

6.17 0.1 (2010-11-10)

- Initial tag (made in svn, not visible in git). Visible at http://www.sdss3.org/svn/repo/pydl/tags/0.1 .

CHAPTER 7

TODO

- Increase test coverage.
- `pydl.pydlutils.mangle` needs more work.
  - Area (solid angle) calculation.
  - Area calculation needs to account for the `use_caps` attribute.
  - Intersection of caps with other caps.
- Use numpy/scipy Cholesky tools
  - https://trac.sdss.org/browser/repo/sdss/idlutils/trunk/pro/bspline/cholesky_band.pro
  - https://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.cholesky.html
  - https://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.cholesky.html
- Update `astropy_helpers` to v2.0.8.
- Check `groupdim`, `groupsize` in `djs_reject()`. Make sure integer division works.
- Document `bspline`.

CHAPTER 8

Authors and Credits

Thank you to everyone who has contributed to PyDL. If we have neglected to include you, please let us know right away!

- Kyle Barbary
- Larry Bradley
- Matt Craig
- Christoph Deil
- Mike DiPompeo
- igoldste
- Duncan Macleod
- Demitri Muna
- Thomas Robitaille
- José Sánchez-Gallego
- Brigitta Sipocz
- Erik Tollerud
- ViviCoder
- Benjamin Alan Weaver
- Kyle Westfall

CHAPTER 9

Licenses

9.1 PyDL License

PyDL is licensed under a 3-clause BSD style license:

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the Astropy Team nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

Part IV

Base API

CHAPTER 10

pydl Package

Python replacements for functions that are part of the IDL built-in library, or part of astronomical IDL libraries. The emphasis is on reproducing results of the astronomical library functions. Only the bare minimum of IDL built-in functions are implemented to support this.

10.1 Functions

file_lines(path[, compress])	Replicates the IDL FILE_LINES() function.
median(array[, width, axis, even])	Replicate the IDL MEDIAN() function.
rebin(x, d[, sample])	Resize x to new dimensions given by d.
smooth(signal, owidth[, edge_truncate])	Replicates the IDL SMOOTH() function.
uniq(x[, index])	Replicates the IDL UNIQ() function.

10.1.1 file_lines

pydl.**file_lines**(*path*, *compress=False*)

Replicates the IDL FILE_LINES() function.

Given a path to a file name or a list of such paths, returns the number of lines in the file(s).

Parameters

path
[str or list of str] Path to a file. Can be a list of paths.

compress
[bool, optional] If set to True, assumes that all files in path are GZIP compressed.

Returns

:class:‘int‘ or :class:‘list‘ of :class:‘int‘
The number of lines in `path`. Returns a list of lengths if a list of files is supplied.

Notes

The `/NOEXPAND_PATH` option in IDL's `FILE_LINES()` is not implemented.

References

http://www.harrisgeospatial.com/docs/file_lines.html

Examples

```
>>> from pydl import file_lines
>>> from os.path import dirname, join
>>> file_lines(join(dirname(__file__),'tests','t','this-file-contains-42-lines.txt'))
42
```

10.1.2 median

pydl.**median**(*array*, *width=None*, *axis=None*, *even=False*)
Replicate the IDL `MEDIAN()` function.

Parameters

array
[array-like] Compute the median of this array.

width
[`int`, optional] Size of the neighborhood in which to compute the median (*i.e.*, perform median filtering). If omitted, the median of the whole array is returned.

axis
[`int`, optional] Compute the median over this axis for a multi-dimensional array. If ommitted, the median over the entire array will be returned. If set, this function will behave as though `even` is `True`.

even
[`bool`, optional] If set to `True`, the median of arrays with an even number of elements will be the average of the middle two values.

Returns

array-like
The median of the array.

Raises

:exc:‘ValueError‘
If `width` is set, and the input `array` is not 1 or 2 dimensional.

Notes

- For arrays with an even number of elements, the `numpy.median()` function behaves like `MEDIAN(array, /EVEN)`, so the absence of the `even` keyword has to turn *off* that behavior.
- For median filtering, this uses `scipy.signal.medfilt()` and `scipy.signal.medfilt2d()` under the hood, but patches up the values on the array boundaries to match the return values of the IDL `MEDIAN()` function.

10.1.3 rebin

pydl.**rebin**(*x*, *d*, *sample=False*)

Resize `x` to new dimensions given by `d`. The new dimensions must be integer multiples or factors of the original dimensions.

Although there are some elegant solutions out there for rebinning, this function is intended to replace the IDL `REBIN()` function, which has a number of special properties:

- It refuses to perform extrapolation when rebinning to a larger size in a particular dimension.
- It can simultaneously rebin to a larger size in one dimension while rebinning to a smaller size in another dimension.

Parameters

x
[`ndarray`] The array to resample.

d
[`tuple()`] The new shape of the array.

sample
[`bool`, optional] If `True`, nearest-neighbor techniques will be used instead of interpolation.

Returns

:class:‘~numpy.ndarray‘
The resampled array.

Raises

:exc:‘ValueError‘
If the new dimensions are incompatible with the algorithm.

References

http://www.harrisgeospatial.com/docs/rebin.html

Examples

```
>>> from numpy import arange, float
>>> from pydl import rebin
>>> rebin(arange(10, dtype=float), (5,)) # doctest: +NORMALIZE_WHITESPACE
```

(continues on next page)

(continued from previous page)

```
array([ 0.5,  2.5,  4.5,  6.5,  8.5])
>>> rebin(arange(5, dtype=float), (10,)) # doctest: +NORMALIZE_WHITESPACE
array([ 0. ,  0.5,  1. ,  1.5,  2. ,  2.5,  3. ,  3.5,  4. ,  4. ])
```

10.1.4 smooth

pydl.**smooth**(*signal*, *owidth*, *edge_truncate=False*)

Replicates the IDL `SMOOTH()` function.

Parameters

signal
[array-like] The array to be smoothed.

owidth
[`int` or array-like] Width of the smoothing window. Can be a scalar or an array with length equal to the number of dimensions of `signal`.

edge_truncate
[`bool`, optional] Set `edge_truncate` to `True` to apply smoothing to all points. Points near the edge are normally excluded from smoothing.

Returns

array-like
A smoothed array with the same dimesions and type as `signal`.

References

http://www.harrisgeospatial.com/docs/smooth.html

10.1.5 uniq

pydl.**uniq**(*x*, *index=None*)

Replicates the IDL `UNIQ()` function.

Returns the *subscripts* of the unique elements of an array. The elements must actually be *sorted* before being passed to this function. This can be done by sorting `x` explicitly or by passing the array subscripts that sort `x` as a second parameter.

Parameters

x
[array-like] Search this array for unique items.

index
[array-like, optional] This array provides the array subscripts that sort `x`.

Returns

array-like
The subscripts of `x` that are the unique elements of `x`.

Notes

Given a sorted array, and assuming that there is a set of adjacent identical items, `uniq()` will return the subscript of the *last* unique item. This charming feature is retained for reproducibility.

References

http://www.harrisgeospatial.com/docs/uniq.html

Examples

```
>>> import numpy as np
>>> from pydl import uniq
>>> data = np.array([ 1, 2, 3, 1, 5, 6, 1, 7, 3, 2, 5, 9, 11, 1 ])
>>> print(uniq(np.sort(data)))
[ 3  5  7  9 10 11 12 13]
```

10.2 Classes

pcomp(x[, standardize, covariance])	Replicates the IDL `PCOMP()` function.
PydlException	Base class for exceptions raised in PyDL functions.

10.2.1 pcomp

class pydl.**pcomp**(*x*, *standardize=False*, *covariance=False*)

Bases: object

Replicates the IDL `PCOMP()` function.

The attributes of this class are all read-only properties, implemented with `lazyproperty`.

Parameters

x
: [array-like] A 2-D array with N rows and M columns.

standardize
: [bool, optional] If set to `True`, the input data will have its mean subtracted off and will be scaled to unit variance.

covariance
: [bool, optional.] If set to `True`, the covariance matrix of the data will be used for the computation. Otherwise the correlation matrix will be used.

References

http://www.harrisgeospatial.com/docs/pcomp.html

Attributes Summary

coefficients	(ndarray) The principal components.
derived	(ndarray) The derived variables.
eigenvalues	(ndarray) The eigenvalues.
variance	(ndarray) The variances of each derived variable.

Attributes Documentation

coefficients
(ndarray) The principal components. These are the coefficients of derived. Basically, they are a re-scaling of the eigenvectors.

derived
(ndarray) The derived variables.

eigenvalues
(ndarray) The eigenvalues. There is one eigenvalue for each principal component.

variance
(ndarray) The variances of each derived variable.

10.2.2 PydlException

exception pydl.**PydlException**
Base class for exceptions raised in PyDL functions.

Bibliography

[1] Tsalmantza, P., Decarli, R., Dotti, M., Hogg, D. W., 2011 ApJ 738, 20

[2] Tsalmantza, P., Hogg, D. W., 2012 ApJ 753, 122

Python Module Index

p

Index

R

S

T

U

V

W

X

Y