
Whisky Documentation

Release 0.0.1

Interferometric Monkeys

November 07, 2014

CONTENTS

1	libs Package	3
1.1	libs Package	3
1.2	astro Module	3
1.3	flux Module	4
1.4	font Module	4
1.5	funcs Module	5
1.6	getoption Module	12
1.7	img Module	14
1.8	progbar Module	17
2	write_docstrings Module	19
	Python Module Index	21
	Index	23

LIBS PACKAGE

1.1 libs Package

A package that includes miscellaneous functions of various utility.

1.2 astro Module

`libs.astro.cal2JD` (*time=None*)

`libs.astro.fr_to_nu` (*fr, eccentricity, prec=1e-10, degrees=False*)

input: *fr* [% of period] (can be vector), *eccentricity* [0-1]; output: *nu* [deg] (same vector-size as *fr*)

`libs.astro.julian_to_calendar` (*julian*)

Calculates the calendar date from a julian date.

Parameters *julian* (*real*) – the data to pad, can be 1 to 3 dimensions

Returns anti-padded data as ndarray

```
>>> dvsef
```

`libs.astro.next_date` (*initdate, period, date_after=False*)

ads ‘period’ to ‘initdate’ as many times as required to have the new date right before the current gm time (or right after if ‘date_after’ is True)

`libs.astro.now` ()

Returns the current UT time

`libs.astro.nu_to_fr` (*nu, eccentricity, degrees=False*)

input: *nu* N-dim, *eccentricity* [0-1]; output: *fr* [% of period] (same shape as *nu*)

`libs.astro.orbital_pos` (*nu, e, i, w, a=149597871000.0, o=0.0, degrees=False*)

Calculates the orbital positions of a body with Keplerian orbit

Parameters

- **nu** (*N-dim array*) – true anomaly of the body (position on its orbit)
- **e** (*real*) – eccentricity of the orbit, must be [0,1)
- **i** (*real, radian*) – inclination of the orbit, must be [0,180). *i*>90 means retrograde body
- **w** (*real, radian*) – argument at periapsis of the orbit, should be [0,360)

- **a** (*real*) – semi-major axis of the orbit
- **o** (*real, radian*) – longitude of the ascending node of the orbit, should be [0,360)
- **degrees** (*bool*) – if True, i, w, o will be expected in degrees

Returns (radius, projected radius, north angle, phase angle), each having the same shape as nu. radius and projected radius have the same unit as a; north angle and phase angle are radians unless degrees is True.

Raises

- Exception, if e is outside [0,1)
- Exception, if i is outside [0,180]

```
>>> import misc.funcs as funcs
>>> print funcs.orbital_pos([10,20,30], 0.1, 89.6, 56, 1, 90, degrees=True)
(array([ 0.90124472,  0.90496144,  0.91109671]),
 array([ 0.36661431,  0.21901579,  0.06387085]),
 array([ 90.89833379,  91.60388027,  95.70132773]),
 array([ 24.00313585,  14.00559899,  4.01991791]))
```

1.3 flux Module

`libs.flux.blackbody_spectral_irr` (*teff, wl*)

Give an effective temerature Returns the emitted (at surface!) flux in W/m2/m assuming black body behaviour

UNITS: teff: K wl: m - can be vector output: W/m2/m

EXAMPLE: >>> blackbody_spectral_irr(5778, np.linspace(550-44, 550+44)*e-9)

`class libs.flux.c_filter` (*band, data*)

`libs.flux.import_bands` (*filename='data/UBVRIJHK.txt', skiprows=2*)

Reads filename skiping the skiprows first rows Returns a dictionary of filters

EXAMPLE: filters=import_bands('UBVRIJHK.txt', 2) filters['V'] <enter> shows all properties

`libs.flux.magref2bbflux` (*ref_mag, ref_band, band_or_wl, teff, photometry_file='data/UBVRIJHK.txt', unit='W'*)

Give a reference magnitude, a reference band and wavelength or band for which you want to know the flux given an effective temperature of the star and the black body assumption. Returns a magnitude in the band_or_wl band, or a flux in W/m2/m for each of the wavelength in band_or_wl.

UNITS: ref_mag/ref_band: U, B, V, R, I, J, H, K band_or_wl: band or meter - can be a vector teff: K output: depending on unit 'Phm': Ph/s/m2/m (m being the walength grain) 'Wm': W/m2/m (m being the walength grain) 'Ph': Ph/s/m2 'W': W/m2 if band_or_wl is a photometric band, unit is disregarded and a mag. is returned

EXAMPLE: >>> magref2bbflux(4.83, 'V', 'R', 5778)

http://www.stsci.edu/hst/nicmos/tools/conversion_form.html

1.4 font Module

This module provides the user with a set of string constants to be concatenated, in order to add colors and/or formatting to terminal display.


```
>>> import misc.font as font
>>> print 'Hello, '+font.red+font.underlined+' I am in underlined red'+font.normal

>>> font. <tab>
Will give you the list of available colors and formatting options
```

Colors and font attributes taken from: http://misc.flogisoft.com/bash/tip_colors_and_formatting

1.5 funcs Module

```
libs.funcs.apodizer(sizeframe, no_power_radius, full_power_radius=None, order=2.0,
                    no_power_limit=0.01, full_power_limit=0.99)
```

```
libs.funcs.array_to_bins(arr)
```

TBD

Parameters **data** (*array*) – the data to pad, can be 1 to 3 dimensions

Returns anti-padded data as ndarray

```
>>> import misc.funcs as funcs
>>> print func.array_to_bins([1,4,5,7])
[-0.5  2.5  4.5  6.   8. ]
```

```
libs.funcs.array_to_binsize(arr)
```

TBD

Give the bin size between elements of arr so that the whole range from min(arr) to max(arr) is filled.
the i-th binsize = (arr[i+1]-arr[i-1])/2

```
>>> import misc.funcs as funcs
>>> print funcs.array_to_binsize([1,4,5,7])
[ 3.   2.   1.5  2. ]
```

```
libs.funcs.aslist(data, numpy=False, integer=False)
```

Transforms data into a 1 dimensional list

Parameters

- **data** – the data to transform into a list, can be any dimension and size
- **numpy** (*bool*) – returns the list as ndarray
- **integer** (*bool*) – returns the list as integer

Returns a 1 dimensional list of data

```
>>> import misc.funcs as funcs
>>> print funcs.aslist(1)
[1]
>>> print funcs.aslist([(1.0,4.0),(2.0,3.0)], numpy=True, integer=True)
[1, 4, 2, 3]
```

```
libs.funcs.bin_array(data, binning, mode=<function sum at 0x9c9d534>)
```

Groups together adjacent cells of an array

Parameters

- **data** – the N-dim array to process
- **binning** (*int or tuple of int*) – the shape of the N-dim cube that defines adjacent cells

- **mode** (*callable function*) – the function to apply on the adjacent cells

Returns the data, where adjacent cells in a cube of shape `binning` got collapsed together following `mode` method. The shape of the new data is `_np.asarray(data.shape)/_np.asarray(binning)`

Raises Exception, if `binning` data.shape is not a multiple of `binning`, or if their size disagree

```
>>> import misc.funcs as funcs
>>> vector = _np.array(_np.matrix(_np.arange(4)).T*_np.arange(6))
>>> data=_np.array([vector, vector+1])
>>> print data
[[[ 0  0  0  0  0  0]
  [ 0  1  2  3  4  5]
  [ 0  2  4  6  8 10]
  [ 0  3  6  9 12 15]]
 [[ 1  1  1  1  1  1]
  [ 1  2  3  4  5  6]
  [ 1  3  5  7  9 11]
  [ 1  4  7 10 13 16]]]
>>> print data.shape
(2, 4, 6)
>>> binned_data = funcs.bin_array(data, (2, 2, 1))
>>> print binned_data
[[[ 2  4  6  8 10 12]
  [ 2 12 22 32 42 52]]]
>>> print binned_data.shape
(1, 2, 6)
```

Note: `mode` must be a callable function that collapses 2D data along its axis 1 to a 1D data.

This callable function must take:

- some 2D data as first parameter
- a parameter `axis` that will be set to 1

You may use `_np.sum`, `_np.mean`, `_np.median`, `_np.max`, `_np.min`, ... functions.

`libs.funcs.cat_cond(cond, iftrue='', iffalse='')`

Returns `iftrue` as string if `cond` is True or not None, or `iffalse` as string if `cond` is false or None

`libs.funcs.center_array(data, newshape)`

Extracts the central part of an array

Parameters

- **data** – the N-dim array to process
- **newshape** (*int or tuple of int*) – the shape of the final subarray to extract

Returns central portion of the array, with shape `newshape`

```
>>> import misc.funcs as funcs
>>> print funcs.center_array([0,1,1,0], 2)
[1 1]
>>> print funcs.center_array([[0,1,1,0],[0,2,2,0],[0,3,3,0]], (1,2))
[[2 2]]
```

`libs.funcs.check_str(text, rem_char=None, auth_char=None, rem_words=None, auth_words=None, ignore_case=False)`

```
libs.funcs.cmd_line(command, ret=True, clean=True)
```

Executes a command line in shell environment

Parameters

- **command** (*string*) – the command to execute
- **ret** (*bool.*) – if True, the function will wait until command is executed. Else, the python script will go on while command is executed in the background
- **clean** (*bool.*) – if True, the ‘empty lines’ and the ‘end of line characters’ will be deleted from the output list

Returns None if ret is False, or if it is True a list of the standard output lines given back by command

```
>>> import misc.funcs as funcs
>>> print funcs.cmd_line('sleep 1')
[] (after 1 sec wait)
>>> print funcs.cmd_line('sleep 1', False)
will output nothing and jump to the next python instruction
>>> print funcs.cmd_line('sleep 1;ls *.dum', True, False)
['dummy.dum\n', 'dummy2.dum\n'] (after 1 sec wait)
```

```
libs.funcs.colorbar(cmap='jet', cm_min=0, cm_max=1)
```

```
libs.funcs.euler_rot(x, y, z, z1, x2, z3, degrees=False)
```

Calculates the Euler rotation (z1, x2, z3) of the input coordinates

Parameters

- **x-y-z** (*N-dim array*) – the coordinates to rotate
- **z1-x2-z3** (*real, radian*) – the Euler angles
- **degrees** (*bool*) – if True, the Euler angles z1, x2, z3 will be expected in degrees

Returns (x', y', z'), the Euler angle rotated values of (x, y, z)

```
>>> import misc.funcs as funcs
>>> print funcs.euler_rot(1, 2, 3, 0.1, 0.2, -0.6)
(1.898096374053891, 0.66116903171326336, 3.1559603397867377)
```

```
libs.funcs.gauss(x, a=1.0, x0=0.0, sigma=1.0, foot=0.0)
```

```
libs.funcs.gauss2D(x, y, a=1.0, x0=0.0, y0=0.0, sigma=1.0, foot=0.0)
```

```
libs.funcs.gauss2D_sup(x, y, a=1.0, x0=0.0, y0=0.0, sigma=1.0, foot=0.0, n=2)
```

```
libs.funcs.gauss_fit(x, y, a=None, x0=None, sigma=None, foot=None)
```

```
libs.funcs.gauss_sup(x, a=1.0, x0=0.0, sigma=1.0, foot=0.0, n=2)
```

```
libs.funcs.gen_generator(seed=None)
```

Generates a random generator with a seed based on the micro second time and the process id. This makes it multi-threading safe.

Returns a random number generator object

```
>>> import misc.funcs as funcs
>>> rnd = funcs.gen_generator()
>>> print rnd.uniform()
0.472645
```

Note: This function calls `gen_seed()` if seed is not given.

`libs.funcs.gen_seed()`

Generates a seed based on the micro second time and the process id. This makes it multi-threading safe.

Returns an integer seed

```
>>> import misc.funcs as funcs
>>> print funcs.gen_seed()
1825296981
```

`class libs.funcs.idlvar_read(filename)`

Reads an IDLVAR file and converts it to a python object

Parameters `filename` (*string*) – (path+)filename to the IDLVAR file

Returns an object whose attributes are the different variables found in the IDLVAR file

```
>>> import misc.funcs as funcs
>>> myvar = funcs.idlvar('/folder/folder/file.idlvar')
>>> myvar. <tab>
This will display all variables stored in the IDLVAR file
```

`libs.funcs.integrate_array(x, y, axis=0)`

Calculates the trapezoidal integral $Y(x)$ of $y(x)$ such as $Y[i] = \text{integral of } y \text{ from } x[0] \text{ to } x[i]$, along one specific axis of y if it is multi-dimensional

Parameters

- **x-y** (*real*) – arrays that define a function f such as $y=f(x)$. x and y along `axis` must have the same dimension
- **axis** (*int*) – if y is multi-dimensional, it tells the axis along which the integral must be carried

Returns $Y(x)$, a numpy array having the same dimension as y

```
>>> import misc.funcs as funcs
>>> y = _np.tile(_np.array(_np.matrix(_np.ones(3)).T*_np.arange(4)), (2,1,1))
>>> print y
[[[ 0.  1.  2.  3.]
  [ 0.  1.  2.  3.]
  [ 0.  1.  2.  3.]]
 [[ 0.  1.  2.  3.]
  [ 0.  1.  2.  3.]
  [ 0.  1.  2.  3.]]]
>>> print y.shape
(2, 3, 4)
>>> print funcs.integrate_array(_np.arange(4), y, axis=2)
[[[ 0.  0.5  2.  4.5]
  [ 0.  0.5  2.  4.5]
  [ 0.  0.5  2.  4.5]]
 [[ 0.  0.5  2.  4.5]
  [ 0.  0.5  2.  4.5]
  [ 0.  0.5  2.  4.5]]]
```

`libs.funcs.nextp2(data)`

`libs.funcs.nextpn(data, power)`

`libs.funcs.pad_array(data, fill_value=0, axes=None, newshape=None)`

Pads data to the double of its size (unless specified) in each specified axis

Parameters

- **data** – the N-dim array to pad
- **fill_value** (*real*) – the value to fill the padding with
- **axes** (*int or tuple of int*) – the axes along which to perform anti-padding, default is all axes
- **newshape** (*“fft”, int or tuple of int*) –
 - if set to “fft”, the data will be padded to the next power of 2 along each specified axis
 - if array.shape-like, axes is ignored and data returned as the center of a an array with shape newshape

Returns padded data as ndarray

Raises Exception, if dimension of newshape does not agree with data.shape

```
>>> import misc.funcs as funcs
>>> print funcs.pad_array([1,1])
[ 0.  1.  1.  0. ]
>>> print funcs.pad_array([[1,1],[2,2]],3, axes=-1)
[[ 3.  1.  1.  3. ]
 [ 3.  2.  2.  3. ]]
```

Note: If newshape is set to “fft”, all specified axes will be returned with the same size, which will be determined by the greatest next power of 2

If newshape is not specified, the size of each specified axis will be multiplied by 2

```
>>> print funcs.pad_array([[1,2]], 0, axes=(1), newshape="fft")
[[ 0.  1.  2.  0.]]
>>> print funcs.pad_array([[1,2]], 0, axes=(0,1), newshape="fft")
[[ 0.  0.  0.  0.]
 [ 0.  1.  2.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]]
```

`libs.funcs.padrev_array(data, axes=None, newshape=None)`

Anti-pads data to the half of its size in each specified axis, or to newshape is specified

Parameters

- **data** – the N-dim array to unpad
- **axes** (*int or tuple of int*) – the axes along which to perform anti-padding, default is all axes
- **newshape** (*int or tuple of int*) – if not None, axes is ignored and the inner region of data is returned, with shape newshape

Returns unpadded data as ndarray

```
>>> import misc.funcs as funcs
>>> print funcs.padrev_data([0,1,1,0])
[ 1.  1. ]
```

```
>>> print funcs.padrev_data([[0,1,1,0],[0,2,2,0]], axes=1)
[[1 1]
 [2 2]]
>>> print funcs.padrev_data([[0,1,1,0],[0,2,2,0]], newshape=(2,2))
[[1 1]
 [2 2]]
```

Note: This function calls `center_array()`.

If `newshape` is not specified, the size of each specified axis will be divided by 2

`libs.funcs.percentile_view(data, bins=None, log=False, plot=None)`
Calculates the percentile of a data and plots the result if required

Parameters

- **data** (*bool*) – the data, any dimension possible as it will be flattened
- **bins** (*int or array*) – the number of percentile values to be calculated, they are linearly spaced between 0 to 100 (included). If `bins` is an array, the percentile values are calculated for each of these values
- **log** (*bool*) – if true, shows the plot with log scale on y
- **plot** (*int or False*) – the number of the figure on which the curve is (over)plotted; if False, the function returns a vector; else or None, a new figure is created

Returns a vector of the calculated percentile if `plot` is False, otherwise None

```
>>> import misc.funcs as funcs
>>> print funcs.percentile_view(_np.random.random(1000), [25, 50, 75])
[0.23487384236763059, 0.48188344917304932, 0.76106477406241557]
```

`libs.funcs.random_custom_cdf(x, cdf, size=1, renorm=False)`
Generates random numbers (or the generating function) following a custom cdf

Parameters

- **x** (*array*) – the x data relative to the pdf
- **cdf** (*array*) – the cdf, must be the same size as x
- **size** (*bool, int, tuple*) – the size of the output: * if False, this function returns a function * if int, this function returns an array of size `size` * if array, this function returns an array whose shape is `size`
- **renorm** (*bool*) – forces first and last element of the cdf (obtained from the pdf) to be 0 and 1

Returns randomly generated numbers (or the generating function) following the pdf distribution

Raises

- Exception, if cdf is not monotonic increasing
- Exception, if first and last element of cdf are not 0 and 1 (and `renorm` is False)

```
>>> import misc.funcs as funcs
>>> x = _np.r_[_np.linspace(0,0.5,100), _np.linspace(0.5,1,101)[1:]]
>>> y = _np.r_[_np.linspace(0,0.5,100), 0.5-_np.linspace(0,0.5,100)]
>>> y = integrate_array(x, y)
```

```
>>> print func.random_custom_cdf(x, y, size=(3,2), renorm=True)
[[ 0.34601846  0.57638203]
 [ 0.46625474  0.52558282]
 [ 0.28020879  0.14224155]]
```

Note: This function calls `gen_generator()`.

```
libs.funcs.random_custom_pdf(x, pdf, size=1, renorm=False)
```

Generates random numbers (or the generating function) following a custom pdf

Parameters

- **x** (*array*) – the x data relative to the pdf
- **pdf** (*array*) – the pdf, must be the same size as x
- **size** (*bool, int or tuple of int*) – the size of the output: * if False, this function returns a function * if int, this function returns an array of size `size` * if array, this function returns an array whose shape is `size`
- **renorm** (*bool*) – forces first and last element of the cdf (obtained from the pdf) to be 0 and 1

Returns randomly generated numbers (or the generating function) following the pdf distribution

Raises Exception, if any value of the pdf is negative

```
>>> import misc.funcs as funcs
>>> x = _np.r_[_np.linspace(0,0.5,100), _np.linspace(0.5,1,101)[1:]]
>>> y = _np.r_[_np.linspace(0,0.5,100), 0.5-_np.linspace(0,0.5,100)]
>>> print funcs.random_custom_pdf(x, y, size=(3,2), renorm=True)
[[ 0.48273149  0.76731952]
 [ 0.8438356  0.32353295]
 [ 0.44305302  0.5361907 ]]
```

Note: This function calls `random_custom_cdf()`, which calls `gen_generator()`.

```
libs.funcs.rebin_2D(x, y, data, bin_x, bin_y, offset_x=0.0, offset_y=0.0, halfway=False)
```

```
libs.funcs.rebin_2D_smart(x, y, data, bin_x, bin_y)
```

```
libs.funcs.replace_multi(text, reps, ignore_case=False)
```

Return the string obtained by replacing the leftmost non-overlapping occurrences of pattern in string by the replacement repl.

```
libs.funcs.resample_1d(data, pts, norm=True)
```

```
libs.funcs.resample_2d(data, shape, norm=True)
```

```
libs.funcs.setval(cond, iftrue, iffelse, toobject=None, attr='')
```

Returns (or adds) a value (or an attribute to an object) depending on a condition

Parameters

- **cond** (*bool or None*) – the condition. If `cond` is not already a boolean, then `cond` is checked to be None: `cond = cond is not None`
- **iftrue** (*any*) – the value that will be returned or added to the object if `cond` is True

- **iffalse** (*any*) – the value that will be returned or added to the object if `cond` is `False`
- **toobject** (*non-write protected object*) – if not `None`, `setval` will not return a value, but add it to this object
- **attr** (*string*) – the name of the attribute under which the value will be added to `toobject`. Ignored if `toobject` is `None`

Returns `iftrue` value if `cond` is `True` or not `None`, `iffalse` value if `cond` is `False` or `None`, nothing if the value is added to `toobject`

```
>>> import misc.funcs as funcs
>>> a = 12
>>> print funcs.setval(a>10, "Hello", "Goodbye")
Hello
```

```
libs.funcs.shiftmicro_1d(data, delta)
```

```
libs.funcs.shiftmicro_2d(data, dx, dy)
```

```
libs.funcs.timestamp(gm=True, fmt='%Y%m%dT%H%M%S')
```

```
libs.funcs.timestamp_name(filename, ext=None, gm=True,
                          fmt='%Y%m%dT%H%M%S', rem_char='sp',
                          auth_char=None)
```

```
libs.funcs.unique_timeid(unicity=None, freq=10, len_output=None)
```

Generates a short unique ID based on the UT timestamp

Parameters

- **unicity** (*real, in years or None*) – the duration over which the unique ID will be unique. Leave `None` for truly unique ID
- **freq** (*real, in Hz*) – the unique ID update frequency. Systems usually provide timestamps down to the micro-second (`freq=1e6`)
- **len_output** (*int or None*) – the length of the string output, which will be padded with 0 if necessary. Leave `None` for no padding.

Returns a string representing the unique ID based on the UT timestamp

```
>>> import misc.funcs as funcs
>>> print funcs.unique_timeid(10, 1)
1zxtk
```

Note: The alphabet used for the string generation is
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789.

1.6 getopt Module

This module wraps `getopt` library to return a convenient object whose properties are lists containing the attributes and options provided in the command-line call.

```
class libs.getoption.getoption(sysargv, opt='', optval='', param=[], paramval=[],
                               man='')
    Wraps getopt library to return a convenient object whose properties are lists containing the attributes
    and options provided in the command-line call.
```


Parameters

- **sysargv** (*list of string*) – the command-line call, as it is provided by sys.argv
- **opt** (*string*) – the concatenation of all authorized one-char options i.e. “-e”
- **optval** (*string*) – the concatenation of all authorized one-char options with input value i.e. “-e 3”
- **param** (*list of string*) – the list of all authorized multi-char options i.e. “-check”
- **paramval** (*list of string*) – the list of all authorized multi-char options with input value i.e. “-name=test”
- **man** (*string*) – the manual page of the script

Returns a python object whose attributes are lists of options and attributes, see note below for details

Raises Exception, if the getopt library cannot parse properly the command-line call

Note:

The returned object possesses as attributes:

- **command**: a string that shows the full command-line
 - **nargs**: an integer showing the total number of arguments provided
 - **optkey**: a list of all options provided in the command-line call, in the order of the command-line call
 - **opt**: a dictionary of the option provided and their associated value. Value-less options return True
 - **attrs**: a list of all the attributes provided after the options
 - **man**: a string that contains the manual of how to use this script
-

```
>>> import getopt, sys
>>> scriptcall = getopt.getoption(sys.argv, opt='wes', optval='r',
>>>                               man='This is the manual page. Thanks you for using getopt')
>>> if scriptcall['r'] is not None: print "r option was given"
>>> if 'h' in scriptcall.optkey:
>>>     print scriptcall.man
>>>     sys.exit()
```

Note: outputObject['option_name'] will return the value of the option option_name (or True is the option has no value), or None if the option_name was not provided in the command line call

The option -h is automatically added to the list of one-char authorized options

The syntax of the script is automatically generated from authorized input given to getopt and added at the beginning of the provided manual man

This module uses misc.font and misc.funcs.asList()

```
>>> from getopt import getopt
>>> thecall = "conex -w --logfile=~/.log.txt 1 TP 234".split(' ')
>>> # the .split(' ') is use to emulate the parsing of sys.argv
>>> scriptcall = getopt(thecall, opt='wes', paramval=['logfile'],
```

```
man='Manual page of that script')
>>> scriptcall. <tab>
a.attrs    a.command  a.man      a.nargs    a.opt      a.optkey
>>> print scriptcall.nargs
5
>>> print scriptcall.optkey
['w', 'logfile']
>>> print scriptcall.attrs
['1', 'TP', '234']
>>> print scriptcall['w']
True
>>> print scriptcall['logfile']
~/log.txt
>>> print scriptcall['e']
None
```

1.7 img Module

`libs.img.add_poisson_noise(data)`

Adds poisson noise to a N-dim array. Replaces each value of the array with a random new value following poisson distribution whose mean is the initial value of the given array.

Parameters *data* (*real*) – N-dim array

Returns the same N-dim array where each value has been replaced with a random poisson distribution value with same mean (integer values)

```
>>> import misc.funcs as funcs
>>> funcs.add_poisson_noise([[4.9,5.3],[5.1,5.2]])
array([[ 3.,  4.],
       [ 6.,  7.]])
```

`libs.img.azimuthal(image, center=None, zone_radius=None, bin_size=None)`

Calculate the azimuthally averaged radial profile.

image - The 2D image center - The [x,y] pixel coordinates used as the center. The default is

None, which then uses the center of the image (including fracitonal pixels).

TODO : implémenter *bin_size*

`libs.img.blur_image(input, sigma, order=0, output=None, mode='reflect', cval=0.0)`

Multi-dimensional Gaussian filter.

Parameters

- **input** – input array to filter
- **sigma** (*scalar or sequence of scalars*) – standard deviation for Gaussian kernel. The standard deviations of the Gaussian filter are given for each axis as a sequence, or as a single number, in which case it is equal for all axes.
- **order** (*0, 1, 2, 3 or sequence from same set*) – The order of the filter along each axis is given as a sequence of integers, or as a single number. An order of 0 corresponds to convolution with a Gaussian kernel. An order of 1, 2, or 3 corresponds to convolution with the first, second or third derivatives of a Gaussian. Higher order derivatives are not implemented

- **output** (*array*) – The output parameter passes an array in which to store the filter output
- **mode** (*'reflect','constant','nearest','mirror', 'wrap'*) – The mode parameter determines how the array borders are handled, where *cval* is the value when mode is equal to *'constant'*. Default is *'reflect'*
- **cval** (*scalar*) – Value to fill past edges of input if mode is *'constant'*

Note: The multi-dimensional filter is implemented as a sequence of one-dimensional convolution filters. The intermediate arrays are stored in the same data type as the output. Therefore, for output types with a limited precision, the results may be imprecise because intermediate results may be stored with insufficient precision.

This function is an alias of `scipy.ndimage.gaussian_filter`

```
libs.img.disk_add2array(arr, x0, y0, radius, max_amplitude=1.0)
```

Draws and adds a disk patterns to an input array.

Parameters

- **arr** – The 2D array to which the disk will be added. If `dim(arr)>2`, only the 2 last dimensions will be considered
- **x0-y0** (*real, in pixels*) – The pixel coordinates of the disk center
- **radius** (*real, in pixels*) – The radius of the disk
- **max_amplitude** (*real*) – The maximum amplitude of the disk

Returns 2D numpy array – the input array to which the disk was added. Values are real between 0 and `max_amplitude`.

Note: The addition of the disk pattern to the input array is genuinely an addition of the disk pattern values, not a replacement. Overlapping patterns will give values greater than `max_amplitude`

If the radius is lower than 0.5 px, the function will just add `max_amplitude` at `x0` and `y0` coordinates

```
>>> import misc.funcs as funcs
>>> import matplotlib.pyplot as plt
>>> myarray = funcs.disk_add2array(np.zeros([512,512]), 80, 250, 30)
>>> plt.matshow(myarray)
```

```
libs.img.find_boundaries(center, subframesize, framesize=None, mode='fit', integer=True, fuckingvariablewhichnobodyknowswhatisit='hui')
```

Returns the coordinates of a subframe in a frame, given its center and shape.

Parameters

- **center** (*real or array of real*) – the N-dim array of the subframe center's coordinates in the frame
- **subframesize** (*int or array of int*) – the N-dim array of the subframe shape
- **framesize** (*int or array of int*) – the N-dim array of the frame shape, None means “infinite” which is equivalent to mode ```=``simple`
- **mode** (*keyword*) –
 - *simple*: no checks on the frame borders are performed

- *fit*: in the case of a subframe centered too close to a frame border, the subframe center is shifted so the subframe fits inside the frame with the requested sub-frame size
- *cut*: in the case of a subframe centered too close to a frame border, the subframe is truncated so it doesn't go outside the frame. The sub-frame size is not conserved
- **integer** (*bool*) – forces the output to be integers

Returns a N-tuple of 2-element arrays ((xmin, xmax), (ymin, ymax), ...) of the subframe coordinates in the frame

Raises Exception, if the subframe is smaller than the frame in at least 1 dimension

```
>>> import misc.funcs as funcs
>>> funcs.find_boundaries([5.32, 16.76], subframesize=14, framesize=20,
mode="fit")
(array([ 0, 14]), array([ 6, 20]))
```

```
libs.img.get_blob_centers(im, n=2, blob_radius=10, sigma_blurring=None, re-
turn_sigma=False, stop_if_error=False, silent=False)
```

Returns the coordinates of the n blobs of the image im after having performed a blur if specified

TODO: - blob mode sub ou flat rond ou carré, - auto find size of blob from 3 px then sigma (or 9 px etc) - silent option - opt return cleaned frame - auto blur sub pixel

```
libs.img.subframe(data, center, subframesize, mode='fit', fill_value=0.0, by_ref=False)
```

Selects a N-dim subframe out of a N-dim frame, given its center and shape.

Parameters

- **data** – the N-dim array from which the subframe will be taken
- **center** (*real or array of real*) – the N-dim array of the subframe center's coordinates in the frame
- **subframesize** (*real or array of real*) – the N-dim array of the subframe shape
- **mode** (*keyword*) –
 - *fill*: in case the subframe goes over the edge of the frame, conserves the center location in the frame and the subfram size by filling the output with value `fill_value`
 - *fit*: in the case of a subframe centered too close to a frame border, the subframe is shifted so the subframe fits inside the frame with the requested sub-frame shape
 - *cut*: in the case of a subframe centered too close to a frame border, the subframe is truncated so it doesn't go outside the frame. The sub-frame shape is not conserved
- **fill_value** (*real or int*) – the fill value that is used if `mode` is set to `fill`
- **by_ref** (*bool*) – if True, the function will return a subframe of the frame that points to the corresponding slice of the frame. If False, the function returns a copy of it

Returns a N-dim array of shape `subframesize`

Raises

- Exception, if the subframe selection is done by reference with `mode` set to “fill”
- Exception, if the subframe selection is done by reference but data is not a nparray

```
>>> import misc.funcs as funcs
>>> funcs.subframe(np.array(np.matrix(np.arange(10)).T*np.arange(10)),
    center=[3.4, 1.7], subframesize=3, mode="fit")
[[ 2  4  6]
 [ 3  6  9]
 [ 4  8 12]]
```

Note: This function calls `find_boundaries()`.

1.8 progbar Module

class `libs.progbar.progbar` (*valmax=100, barsize=None, title='Be patient...'*)

Progress bar *valmax* is the last value of the iteration. Default is 100. *barsize* is the size of the bar in the opened window. If empty, the bar will automatically fit the window *title* is the title.

reset (*valmax=None, barsize=None, title=None*)

Progress bar *valmax* is the last value of the iteration. Default is 100. *barsize* is the size of the bar in the opened window. If empty, the bar will automatically fit the window *title* is the title.

update (*val='add'*)

Calling `.update()` adds 1 to the progress of the bar. Calling `.update(i)` puts the progress of the bar to *i* value.

WRITE_DOCSTRINGS MODULE

This is some introductory text that is here for the whole module.

This module shows examples how to write good docstrings and make links between modules and functions

```
write_docstrings.another_func()  
    Just for fun
```

```
write_docstrings.write_docstrings(param1, param2='haha')  
    This function does something.
```

*I have a docstring, but **won't** be imported if you don't imported it.*

Parameters

- **param1** (*str.*) – A first parameter.
- **param2** (*bool.*) – Another parameter, depending on param1

Returns int – the return code.

Raises AttributeError, KeyError

You can use that magic function if:

- you are awesome
- your are really awesome
- you hate IDL

You never call this class before calling `another_func()`.

Note: An example of intersphinx is this: you **cannot** use `misc` on this class.

```
>>> print 'this is some code sample followed by the result'  
'this is some code sample followed by the result'
```

source: <http://codeandchaos.wordpress.com/2012/07/30/sphinx-autodoc-tutorial-for-dummies/>

and: https://pythonhosted.org/an_example_pypi_project/sphinx.html#full-code-example

PYTHON MODULE INDEX

I

- `libs`, [3](#)
- `libs.astro`, [3](#)
- `libs.flux`, [4](#)
- `libs.font`, [4](#)
- `libs.funcs`, [5](#)
- `libs.getoption`, [12](#)
- `libs.img`, [14](#)
- `libs.progbar`, [17](#)

W

- `write_docstrings`, [19](#)

INDEX

A

`add_poisson_noise()` (in module `libs.img`), 14
`another_func()` (in module `write_docstrings`), 19
`apodizer()` (in module `libs.funcs`), 5
`array_to_bins()` (in module `libs.funcs`), 5
`array_to_binsize()` (in module `libs.funcs`), 5
`aslist()` (in module `libs.funcs`), 5
`azimuthal()` (in module `libs.img`), 14

B

`bin_array()` (in module `libs.funcs`), 5
`blackbody_spectral_irr()` (in module `libs.flux`), 4
`blur_image()` (in module `libs.img`), 14

C

`c_filter` (class in `libs.flux`), 4
`cal2JD()` (in module `libs.astro`), 3
`cat_cond()` (in module `libs.funcs`), 6
`center_array()` (in module `libs.funcs`), 6
`check_str()` (in module `libs.funcs`), 6
`cmd_line()` (in module `libs.funcs`), 7
`colorbar()` (in module `libs.funcs`), 7

D

`disk_add2array()` (in module `libs.img`), 15

E

`euler_rot()` (in module `libs.funcs`), 7

F

`find_boundaries()` (in module `libs.img`), 15
`fr_to_nu()` (in module `libs.astro`), 3

G

`gauss()` (in module `libs.funcs`), 7
`gauss2D()` (in module `libs.funcs`), 7
`gauss2D_sup()` (in module `libs.funcs`), 7
`gauss_fit()` (in module `libs.funcs`), 7
`gauss_sup()` (in module `libs.funcs`), 7
`gen_generator()` (in module `libs.funcs`), 7
`gen_seed()` (in module `libs.funcs`), 8

`get_blob_centers()` (in module `libs.img`), 16
`getoption` (class in `libs.getoption`), 12

I

`idlvar_read` (class in `libs.funcs`), 8
`import_bands()` (in module `libs.flux`), 4
`integrate_array()` (in module `libs.funcs`), 8

J

`julian_to_calendar()` (in module `libs.astro`), 3

L

`libs` (module), 3
`libs.astro` (module), 3
`libs.flux` (module), 4
`libs.font` (module), 4
`libs.funcs` (module), 5
`libs.getoption` (module), 12
`libs.img` (module), 14
`libs.progbar` (module), 17

M

`magref2bbflux()` (in module `libs.flux`), 4

N

`next_date()` (in module `libs.astro`), 3
`nextp2()` (in module `libs.funcs`), 8
`nextpnp()` (in module `libs.funcs`), 8
`now()` (in module `libs.astro`), 3
`nu_to_fr()` (in module `libs.astro`), 3

O

`orbital_pos()` (in module `libs.astro`), 3

P

`pad_array()` (in module `libs.funcs`), 8
`padrev_array()` (in module `libs.funcs`), 9
`percentile_view()` (in module `libs.funcs`), 10
`progbar` (class in `libs.progbar`), 17

R

`random_custom_cdf()` (in module `libs.funcs`), [10](#)
`random_custom_pdf()` (in module `libs.funcs`), [11](#)
`rebin_2D()` (in module `libs.funcs`), [11](#)
`rebin_2D_smart()` (in module `libs.funcs`), [11](#)
`replace_multi()` (in module `libs.funcs`), [11](#)
`resample_1d()` (in module `libs.funcs`), [11](#)
`resample_2d()` (in module `libs.funcs`), [11](#)
`reset()` (`libs.progbar.progbar` method), [17](#)

S

`setval()` (in module `libs.funcs`), [11](#)
`shiftmicro_1d()` (in module `libs.funcs`), [12](#)
`shiftmicro_2d()` (in module `libs.funcs`), [12](#)
`subframe()` (in module `libs.img`), [16](#)

T

`timestamp()` (in module `libs.funcs`), [12](#)
`timestamp_name()` (in module `libs.funcs`), [12](#)

U

`unique_timeid()` (in module `libs.funcs`), [12](#)
`update()` (`libs.progbar.progbar` method), [17](#)

W

`write_docstrings` (module), [19](#)
`write_docstrings()` (in module `write_docstrings`), [19](#)