

---

# healpy Documentation

*Release 1.2*

**C. Rosset**

August 14, 2012



# CONTENTS



# TUTORIAL

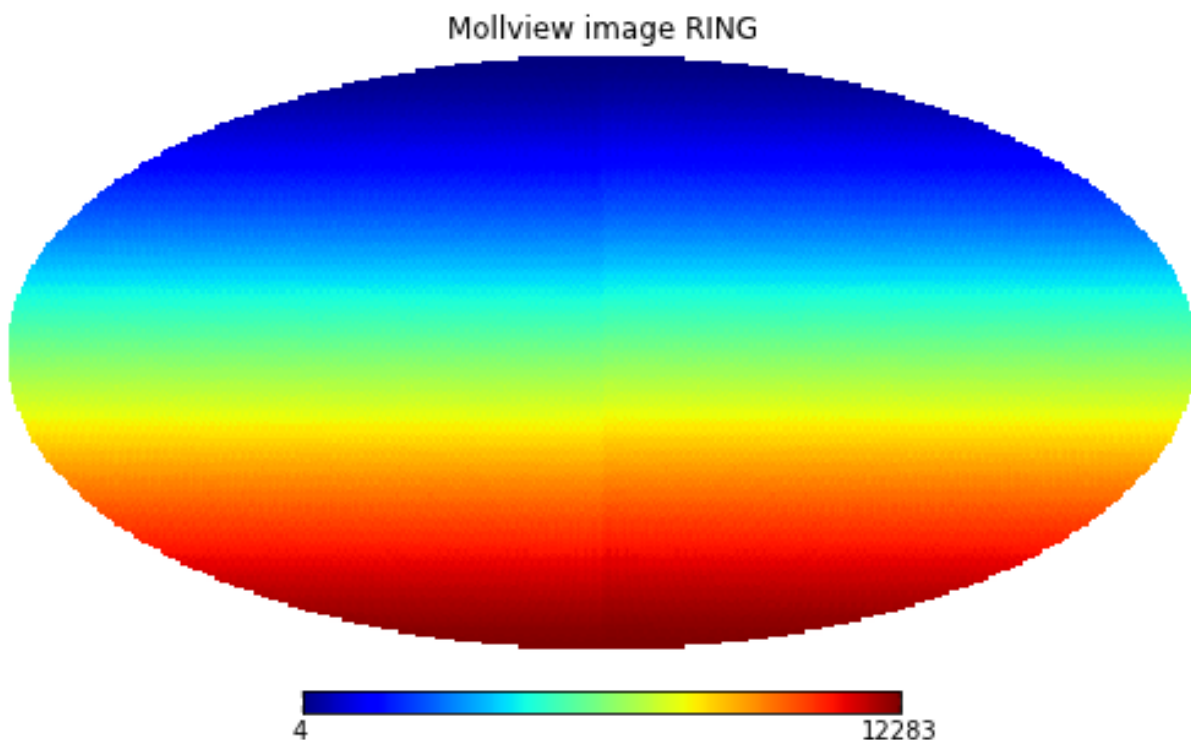
## 1.1 Healpy tutorial

### 1.1.1 Creating and manipulating maps

Maps are simply numpy arrays, each array element refers to a location in the sky as defined by the Healpix pixelization schemes, see the [healpix website](#).

The resolution of the map is defined by the *NSIDE* parameter, the `nside2npix()` function gives the number of pixel *NPIX* of the map:

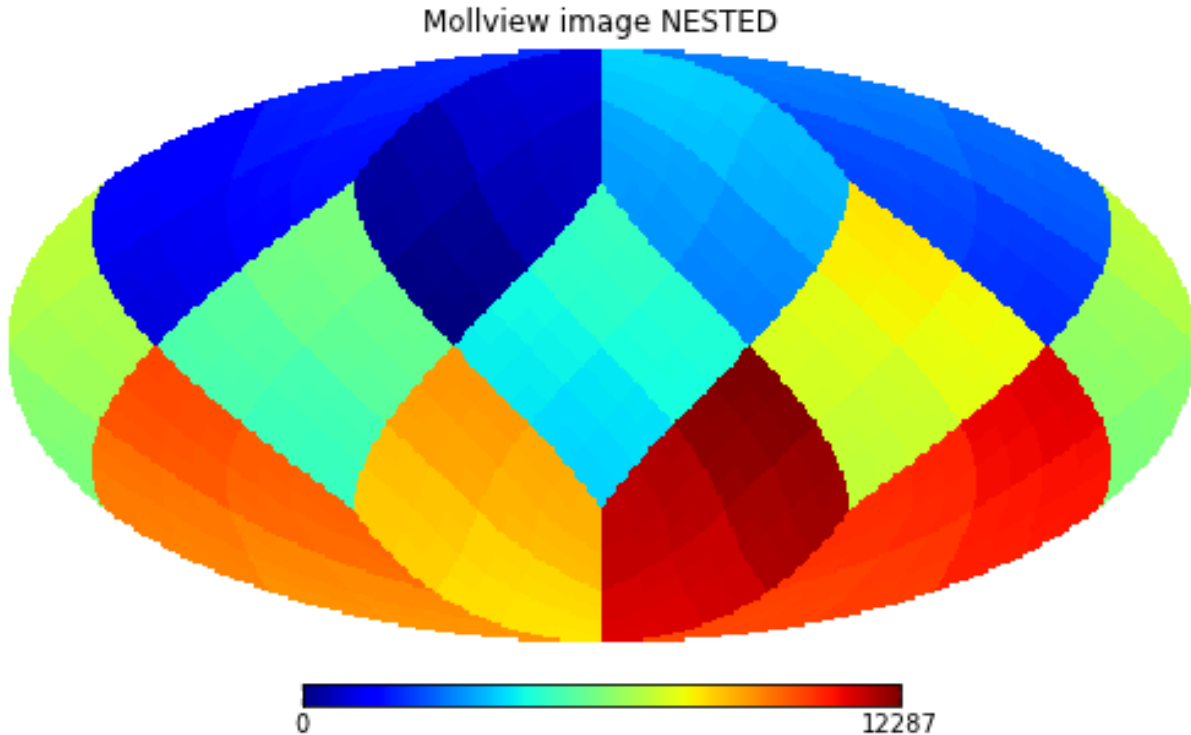
```
>>> import numpy as np
>>> import healpy as hp
>>> NSIDE = 32
>>> m = np.arange(hp.nside2npix(NSIDE))
>>> hp.mollview(m, title="Mollview image RING")
```



Healpix supports two different ordering schemes, *RING* or *NESTED*, by default healpy maps are in *\*RING\** ordering.

In order to work with *NESTED* ordering, all map related functions support the *nest* keyword, for example:

```
>>> hp.mollview(m, nest=True, title="Mollview image NESTED")
```



### 1.1.2 Reading and writing maps to file

Maps are read with the `read_map()` function:

```
>>> wmap_map_I = hp.read_map('../healpy/test/data/wmap_band_imap_r9_7yr_W_v4.fits')
```

by default input maps are **converted to *\*RING\** ordering**, if they are in *NESTED* ordering. You can otherwise specify *nest=True* to retrieve a map is *NESTED* ordering, or *nest=None* to keep the ordering unchanged.

By default `read_map()` loads the first column, for reading other columns you can specify the *field* keyword.

`write_map()` writes a map to disk in FITS format, if the input map is a list of 3 maps, they are written to a single file as I,Q,U polarization components:

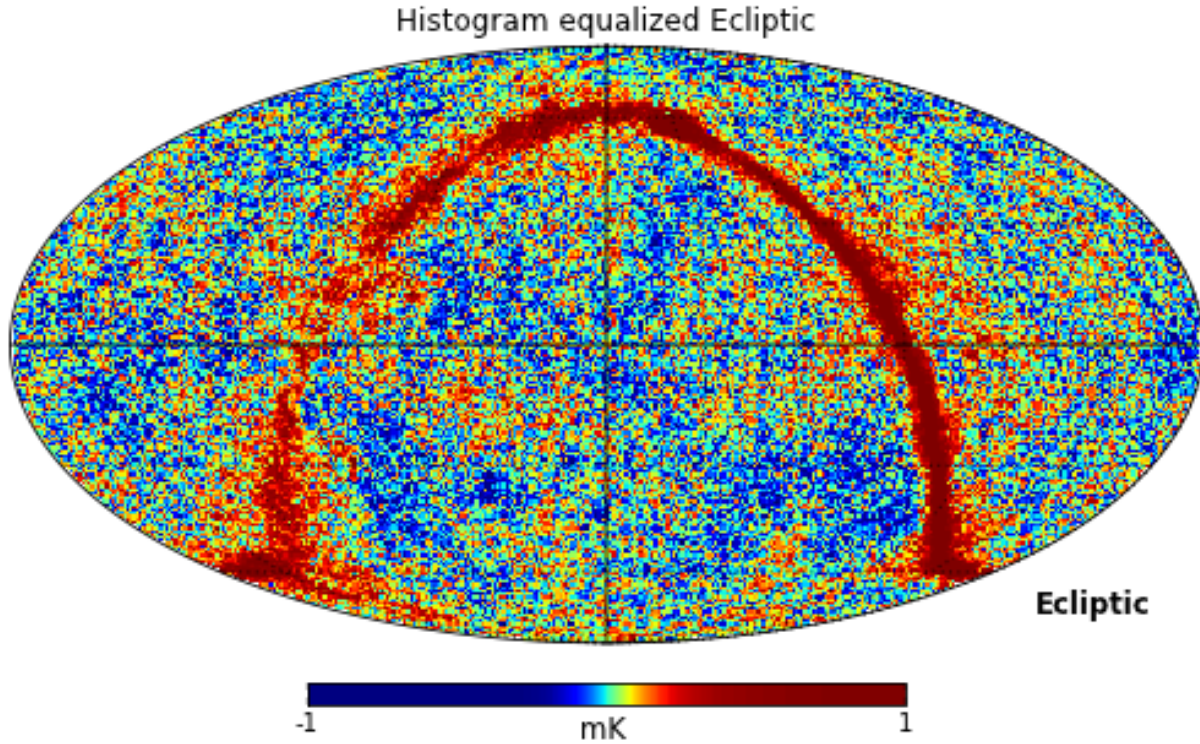
```
>>> hp.write_map("my_map.fits", wmap_map_I)
```

### 1.1.3 Visualization

Mollweide projection with `mollview()` is the most common visualization tool for HEALPIX maps, it supports also coordinate transformation:

```
>>> hp.mollview(wmap_map_I, coord=['G','E'], title='Histogram equalized Ecliptic', unit='mK', norm='I')
>>> hp.graticule()
```

`coord` does galactic to ecliptic coordinate transformation, `norm='hist'` sets a histogram equalized color scale and `xsize` increases the size of the image. `graticule()` adds meridians and parallels.



`gnomview()` instead provides gnomonic projection around a position specified by `rot`:

```
>>> hp.gnomview(wmap_map_I, rot=[0,0.3], title='GnomView', unit='mK', format='%.2g')
```

shows a projection of the galactic center, `xsize` and `ysize` change the dimension of the sky patch.

`mollzoom()` is a powerful tool for interactive inspection of a map, it provides a mollweide projection where you can click to set the center of the adjacent gnomview panel.

### 1.1.4 Masked map, partial maps

By convention HEALPIX uses  $-1.6375 \times 10^{30}$  to mark invalid or unseen pixels, this is stored in healpy as the constant `UNSEEN()`.

All healpy functions automatically deal with maps with UNSEEN pixels, for example `mollview()` marks in grey that sections of a map.

There is an alternative way of dealing with UNSEEN pixel based on the numpy MaskedArray class, `ma()` loads a map as a masked array:

```
>>> mask = hp.read_map('../healpy/test/data/wmap_temperature_analysis_mask_r9_7yr_v4.fits').astype(np.int)
>>> wmap_map_I_masked = hp.ma(wmap_map_I)
>>> wmap_map_I_masked.mask = np.logical_not(mask)
```

by convention the mask is 0 where the data are masked, while numpy defines data masked when the mask is True, so it is necessary to flip the mask.

```
>>> hp.mollview(wmap_map_I_masked.filled())
```

filling a masked array fills the *UNSEEN* value in and return a standard array that can be used by *mollview.compressed()* instead removes all the masked pixels and returns a standard array that can be used for examples by the matplotlib *hist()* function:

```
>>> import matplotlib.pyplot as plt
>>> plt.hist(wmap_map_I_masked.compressed(), bins = 1000)
```

## 1.1.5 Spherical harmonic transforms

healpy provides bindings to the C++ HEALPIX library for performing spherical harmonic transforms. *anafast()* computes the angular power spectrum of a map:

```
>>> LMAX = 1024
>>> cl = hp.anafast(wmap_map_I_masked.filled(), lmax=LMAX)
```

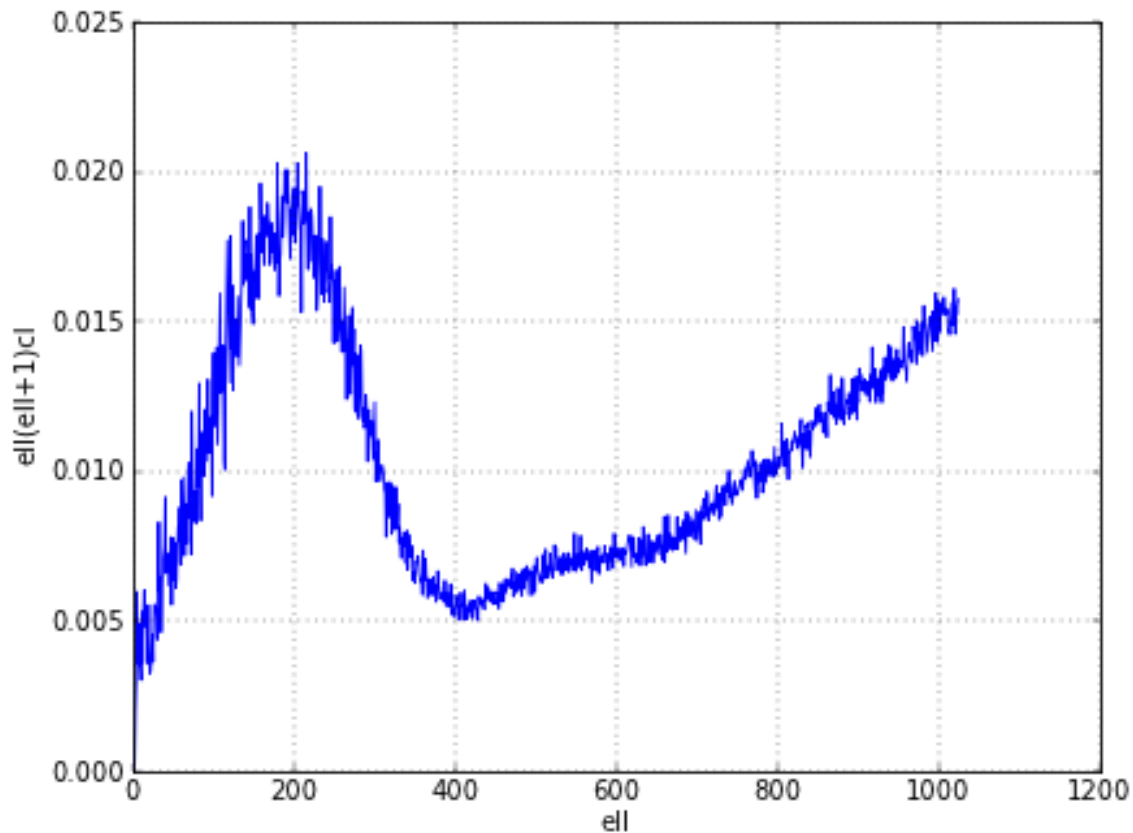
the relative *ell* array is just:

```
>>> ell = np.arange(len(cl))
```

therefore we can plot a normalized CMB spectrum and write it to disk:

```
>>> plt.figure()
>>> plt.plot(ell, ell * (ell+1) * cl)
>>> plt.xlabel('ell'); plt.ylabel('ell(ell+1)cl'); plt.grid()
>>> hp.write_cl('cl.fits', cl)
```





Gaussian beam map smoothing is provided by `smoothing()`:

```
>>> wmap_map_I_smoothed = hp.smoothing(wmap_map_I, fwhm=60, arcmin=True)
>>> hp.mollview(wmap_map_I_smoothed, min=-1, max=1, title='Map smoothed 1 deg')
```



# INSTALLATION

## 2.1 Installation procedure for Healpy

- Note: Healpix is included, so you don't need to get it separately

### 2.1.1 Requirements

- Healpix needs cfitsio. You can either:
  - use an existing installation : Either define the environment variable CFITSIO\_EXT\_PREFIX where to find the cfitsio library and include file (eg /usr/local, so that /usr/local/include/fitsio.h and /usr/local/lib/libcfitsio.a exists), or define CFITSIO\_EXT\_INC (with the cfitsio include directory) and CFITSIO\_EXT\_LIB (with the cfitsio library file name) Please do not define all three of these environment variables simultaneously.
  - compile a specific cfitsio lib: Define EXTERNAL\_CFITSIO=no, place the cfitsioXXXX.tar.gz in hp-beta/libcfitsio before installing. The cfitsio version XXXX must match the version in hpbeta/planck.make (or you need to modify it there).
- the psht spherical transform library is integrated into healpy with the

pshyt cython wrapper. a pregenerated c code is included in the repository, but if you have cython installed it will be run in the build phase.

### 2.1.2 Installation

```
$ cd healpy $ python setup.py build
```

OR, if you do not want OpenMP support (sometimes, it causes SegFault)

```
$ python setup.py build --without-openmp (alternatively, you can define the environment variable HEALPY_WITHOUT_OPENMP)
```

If you do not want the “-march=native” flag (if your g++ is too old)

```
$ python setup.py build --without-native (alternatively, you can define the environment variable HEALPY_WITHOUT_NATIVE)
```

If everything goes fine, you can give a test: 

```
$ cd build/lib* $ ipython -pylab >>> import healpy as H >>> H.mollview(arange(12)) >>> pylab.show()
```

If the plot looks good, you can install: 

```
$ sudo python setup.py install
```

 # install in default location, need root rights or 

```
$ python setup.py install --install-lib=~/.Softs/Python
```

 # will install healpy in directory ~/.Softs/Python, which then must

be in your PYTHONPATH or \$ python setup.py install --user # will install it in your User python directory (python >= 2.6)

### 2.1.3 Compile on OSX

Suggested compilation on OSX Lion is installing pyfits, cython and cfitsio using mac ports and run: >>> python setup.py --without-openmp

### 2.1.4 Development install

the best way to install healpy if you plan to develop is to build the C++ extensions in place with:

```
python setup.py build_ext --inplace
```

the add the healpy/healpy folder to your PYTHONPATH

# REFERENCE

## 3.1 pixelfunc – Pixelisation related functions

### 3.1.1 conversion from/to sky coordinates

<code>pix2ang(nside, ipix[, nest])</code>	<code>pix2ang</code> : <code>nside, ipix, nest=False</code> -> <code>theta[rad], phi[rad]</code> (default RING)
<code>pix2vec(nside, ipix[, nest])</code>	<code>pix2vec</code> : <code>nside, ipix, nest=False</code> -> <code>x, y, z</code> (default RING)
<code>ang2pix(nside, theta, phi[, nest])</code>	<code>ang2pix</code> : <code>nside, theta[rad], phi[rad], nest=False</code> -> <code>ipix</code> (default:RING)
<code>vec2pix(nside, x, y, z[, nest])</code>	<code>vec2pix</code> : <code>nside, x, y, z, nest=False</code> -> <code>ipix</code> (default:RING)
<code>vec2ang(vectors)</code>	<code>vec2ang</code> : <code>vectors [x, y, z]</code> -> <code>theta[rad], phi[rad]</code>
<code>ang2vec(theta, phi)</code>	<code>ang2vec</code> : convert angles to 3D position vector
<code>get_neighbours(nside, theta[, phi, nest])</code>	Return the 4 nearest pixels and corresponding weights.
<code>get_all_neighbours(nside, theta[, phi, nest])</code>	Return the 8 nearest pixels.

#### healpy.pixelfunc.pix2ang

`healpy.pixelfunc.pix2ang(nside, ipix, nest=False)`  
`pix2ang` : `nside, ipix, nest=False` -> `theta[rad], phi[rad]` (default RING)

**Parameters** `nside` : int or array-like

The healpix `nside` parameter, must be a power of 2

`ipix` : int or array-like

Angular coordinates of a point on the sphere

`nest` : bool, optional

if True, assume NESTED pixel ordering, otherwise, RING pixel ordering

**Returns** `theta, phi` : float, scalar or array-like

The angular coordinates corresponding to `ipix`. Scalar if all input are scalar, array otherwise. Usual numpy broadcasting rules apply.

**See Also:**

`ang2pix`, `vec2pix`, `pix2vec`

## Examples

```
>>> import healpy as hp
>>> hp.pix2ang(16, 1440)
(1.5291175943723188, 0.0)

>>> hp.pix2ang(16, [1440, 427, 1520, 0, 3068])
(array([ 1.52911759,  0.78550497,  1.57079633,  0.05103658,  3.09055608]), array([ 0.
, 0.05103658, 0.05103658, 0.05103658, 0.05103658]), array([ 0.05103658, 0.05103658, 0.05103658, 0.05103658, 0.05103658]))

>>> hp.pix2ang([1, 2, 4, 8], 11)
(array([ 2.30052398,  0.84106867,  0.41113786,  0.2044802 ]), array([ 5.49778714,  5.89048623,  6.28318531,  6.67577144]))
```

## healpy.pixelfunc.pix2vec

healpy.pixelfunc.**pix2vec** (*nside*, *ipix*, *nest=False*)  
 pix2vec : nside,ipix,nest=False -> x,y,z (default RING)

**Parameters** **nside** : int, scalar or array-like

The healpix nside parameter, must be a power of 2

**ipix** : int, scalar or array-like

Healpix pixel number

**nest** : bool, optional

if True, assume NESTED pixel ordering, otherwise, RING pixel ordering

**Returns** **x, y, z** : floats, scalar or array-like

The coordinates of vector corresponding to input pixels. Scalar if all input are scalar, array otherwise. Usual numpy broadcasting rules apply.

**See Also:**

[ang2pix](#), [pix2ang](#), [vec2pix](#)

## Examples

```
>>> import healpy as hp
>>> hp.pix2vec(16, 1504)
(0.99879545620517241, 0.049067674327418015, 0.0)

>>> hp.pix2vec(16, [1440, 427])
(array([ 0.99913157,  0.5000534 ]), array([ 0.05103658,  0.5000534]), array([ 0.04166667,  0.70710678]))

>>> hp.pix2vec([1, 2], 11)
(array([ 0.52704628,  0.68861915]), array([-0.52704628, -0.28523539]), array([-0.66666667,  0.66666667]))
```

## healpy.pixelfunc.ang2pix

healpy.pixelfunc.**ang2pix** (*nside*, *theta*, *phi*, *nest=False*)  
 ang2pix : nside,theta[rad],phi[rad],nest=False -> ipix (default:RING)

**Parameters** **nside** : int, scalar or array-like

The healpix nside parameter, must be a power of 2

**theta, phi** : float, scalars or array-like

Angular coordinates of a point on the sphere

**nest** : bool, optional

if True, assume NESTED pixel ordering, otherwise, RING pixel ordering

**Returns** **pix** : int or array of int

The healpix pixel numbers. Scalar if all input are scalar, array otherwise. Usual numpy broadcasting rules apply.

**See Also:**

`pix2ang`, `pix2vec`, `vec2pix`

### Examples

```
>>> import healpy as hp
>>> hp.ang2pix(16, np.pi/2, 0)
1440

>>> hp.ang2pix(16, [np.pi/2, np.pi/4, np.pi/2, 0, np.pi], [0., np.pi/4, np.pi/2, 0, 0])
array([1440,  427, 1520,    0, 3068])

>>> hp.ang2pix(16, np.pi/2, [0, np.pi/2])
array([1440, 1520])

>>> hp.ang2pix([1, 2, 4, 8, 16], np.pi/2, 0)
array([  4,  12,  72, 336, 1440])
```

### healpy.pixelfunc.vec2pix

`healpy.pixelfunc.vec2pix` (*nside*, *x*, *y*, *z*, *nest=False*)

`vec2pix` : *nside*, *x*, *y*, *z*, *nest=False* -> *ipix* (default: RING)

**Parameters** **nside** : int or array-like

The healpix *nside* parameter, must be a power of 2

**x,y,z** : floats or array-like

vector coordinates defining point on the sphere

**nest** : bool, optional

if True, assume NESTED pixel ordering, otherwise, RING pixel ordering

**Returns** **ipix** : int, scalar or array-like

The healpix pixel number corresponding to input vector. Scalar if all input are scalar, array otherwise. Usual numpy broadcasting rules apply.

**See Also:**

`ang2pix`, `pix2ang`, `pix2vec`

## Examples

```
>>> import healpy as hp
>>> hp.vec2pix(16, 1, 0, 0)
1504

>>> hp.vec2pix(16, [1, 0], [0, 1], [0, 0])
array([1504, 1520])

>>> hp.vec2pix([1, 2, 4, 8], 1, 0, 0)
array([ 4, 20, 88, 368])
```

## healpy.pixelfunc.vec2ang

healpy.pixelfunc.**vec2ang**(*vectors*)  
 vec2ang: vectors [x, y, z] -> theta[rad], phi[rad]

**Parameters** **vectors** : float, array-like  
 the vector(s) to convert, shape is (3,) or (N, 3)

**Returns** **theta, phi** : float, tuple of two arrays  
 the colatitude and longitude in radians

### See Also:

[ang2vec](#), [rotator.vec2dir](#), [rotator.dir2vec](#)

## healpy.pixelfunc.ang2vec

healpy.pixelfunc.**ang2vec**(*theta, phi*)  
 ang2vec : convert angles to 3D position vector

**Parameters** **theta** : float, scalar or array-like  
 colatitude in radians measured southward from north pole (in [0,pi]).

**phi** : float, scalar or array-like  
 longitude in radians measured eastward (in [0, 2\*pi]).

**Returns** **vec** : float, array  
 if theta and phi are vectors, the result is a 2D array with a vector per row otherwise, it is a 1D array of shape (3,)

### See Also:

[vec2ang](#), [rotator.dir2vec](#), [rotator.vec2dir](#)

## healpy.pixelfunc.get\_neighbours

healpy.pixelfunc.**get\_neighbours**(*nside, theta, phi=None, nest=False*)  
 Return the 4 nearest pixels and corresponding weights.

**Parameters** **nside** : int  
 the healpix nside



**theta, phi** : float, scalar or array-like

if phi is not given, theta is interpreted as pixel number, otherwise theta[rad], phi[rad] are angular coordinates

**nest** : bool

if True, NESTED ordering, otherwise RING ordering.

**Returns** **res** : tuple of length 2

contains pixel numbers in res[0] and weights in res[1]. Usual numpy broadcasting rules apply.

**See Also:**

`get_interp_val`, `get_all_neighbours`

### Examples

```
>>> import healpy as hp
>>> hp.get_neighbours(1, 0)
(array([0, 1, 4, 5]), array([ 1.,  0.,  0.,  0.]))

>>> hp.get_neighbours(1, 0, 0)
(array([1, 2, 3, 0]), array([ 0.25,  0.25,  0.25,  0.25]))

>>> hp.get_neighbours(1, [0, np.pi/2], 0)
(array([[ 1,  4],
        [ 2,  5],
        [ 3, 11],
        [ 0,  8]]), array([[ 0.25,  1.  ],
        [ 0.25,  0.  ],
        [ 0.25,  0.  ],
        [ 0.25,  0.  ]]))
```

### healpy.pixelfunc.get\_all\_neighbours

`healpy.pixelfunc.get_all_neighbours` (*nside*, *theta*, *phi=None*, *nest=False*)

Return the 8 nearest pixels.

**Parameters** **nside** : int

the nside to work with

**theta, phi** : scalar or array-like

if phi is not given or None, theta is interpreted as pixel number, otherwise, theta[rad], phi[rad] are angular coordinates

**nest** : bool

if True, pixel number will be NESTED ordering, otherwise RING ordering.

**Returns** **ipix** : int, array

pixel number of the SW, W, NW, N, NE, E, SE and S neighbours, shape is (8,) if input is scalar, otherwise shape is (8, N) if input is of length N. If a neighbor does not exist (it can be the case for W, N, E and S) the corresponding pixel number will be -1.

**See Also:**

`get_neighbours`, `get_interp_val`

**Examples**

```
>>> import healpy as hp
>>> hp.get_all_neighbours(1, 4)
array([11,  7,  3, -1,  0,  5,  8, -1])

>>> hp.get_all_neighbours(1, np.pi/2, np.pi/2)
array([ 8,  4,  0, -1,  1,  6,  9, -1])
```

### 3.1.2 conversion between NESTED and RING schemes

<code>nest2ring(nside, ipix)</code>	Convert pixel number from NESTED ordering to RING ordering.
<code>ring2nest(nside, ipix)</code>	Convert pixel number from RING ordering to NESTED ordering.
<code>reorder(map_in[, inp, out, r2n, n2r])</code>	Reorder an healpix map from RING/NESTED ordering to NESTED/RING

#### healpy.pixelfunc.nest2ring

`healpy.pixelfunc.nest2ring(nside, ipix)`  
 Convert pixel number from NESTED ordering to RING ordering.

**Parameters** **nside** : int, scalar or array-like

the healpix nside parameter

**ipix** : int, scalar or array-like

the pixel number in NESTED scheme

**Returns** **ipix** : int, scalar or array-like

the pixel number in RING scheme

**See Also:**

`ring2nest`, `reorder`

**Examples**

```
>>> import healpy as hp
>>> hp.nest2ring(16, 1130)
1504

>>> hp.nest2ring(2, range(10))
array([13,  5,  4,  0, 15,  7,  6,  1, 17,  9])

>>> hp.nest2ring([1, 2, 4, 8], 11)
array([ 11,   2,  12, 211])
```

### healpy.pixelfunc.ring2nest

healpy.pixelfunc.**ring2nest** (*nside*, *ipix*)

Convert pixel number from RING ordering to NESTED ordering.

**Parameters** **nside** : int, scalar or array-like

the healpix nside parameter

**ipix** : int, scalar or array-like

the pixel number in RING scheme

**Returns** **ipix** : int, scalar or array-like

the pixel number in NESTED scheme

**See Also:**

`nest2ring`, `reorder`

#### Examples

```
>>> import healpy as hp
>>> hp.ring2nest(16, 1504)
1130

>>> hp.ring2nest(2, range(10))
array([ 3,  7, 11, 15,  2,  1,  6,  5, 10,  9])

>>> hp.ring2nest([1, 2, 4, 8], 11)
array([ 11,  13,  61, 253])
```

### healpy.pixelfunc.reorder

healpy.pixelfunc.**reorder** (*map\_in*, *inp=None*, *out=None*, *r2n=None*, *n2r=None*)

Reorder an healpix map from RING/NESTED ordering to NESTED/RING

**Parameters** **map\_in** : array-like

the input map to reorder

**inp, out** : 'RING' or 'NESTED'

define the input and output ordering

**r2n** : bool

if True, reorder from RING to NESTED

**n2r** : bool

if True, reorder from NESTED to RING

**Returns** **map\_out** : array-like

the reordered map

**See Also:**

`nest2ring`, `ring2nest`

## Notes

if `r2n` or `n2r` is defined, override `inp` and `out`.

## Examples

```
>>> import healpy as hp
>>> hp.reorder(np.arange(48), r2n = True)
array([13,  5,  4,  0, 15,  7,  6,  1, 17,  9,  8,  2, 19, 11, 10,  3, 28,
       20, 27, 12, 30, 22, 21, 14, 32, 24, 23, 16, 34, 26, 25, 18, 44, 37,
       36, 29, 45, 39, 38, 31, 46, 41, 40, 33, 47, 43, 42, 35])
>>> hp.reorder(range(12), n2r = True)
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

### 3.1.3 nside/npix/resolution

<code>nside2npix(nside)</code>	Give the number of pixel for the given nside.
<code>npix2nside(npix)</code>	Give the nside parameter for the given number of pixels.
<code>nside2resol(nside[, arcmin])</code>	Give approximate resolution for nside.
<code>nside2pixarea(nside[, degrees])</code>	Give pixel area given nside.
<code>isnsideok(nside)</code>	Returns <code>True</code> if nside is a valid nside parameter, <code>False</code> otherwise.
<code>isnpixok(npix)</code>	Return <code>True</code> if npix is a valid value for healpix map size, <code>False</code> otherwise.
<code>get_map_size(m)</code>	Returns the npix of a given map (implicit or explicit pixelization).
<code>get_min_valid_nside(npix)</code>	Returns the minimum acceptable nside so that <code>npix &lt;= nside2npix(nside)</code> .
<code>get_nside(m)</code>	Return the nside of the given map.
<code>maptype(m)</code>	Describe the type of the map (valid, single, sequence of maps).
<code>ud_grade(map_in, nside_out[, pess, ...])</code>	Upgrade or degrade resolution of a map (or list of maps).

### healpy.pixelfunc.nside2npix

`healpy.pixelfunc.nside2npix(nside)`  
Give the number of pixel for the given nside.

**Parameters** `nside` : int

healpix nside parameter; an exception is raised if nside is not valid (nside must be a power of 2)

**Returns** `npix` : int

corresponding number of pixels

## Notes

Raise a `ValueError` exception if nside is not valid.

## Examples

```
>>> import healpy as hp
>>> hp.nside2npix(8)
768

>>> np.all([hp.nside2npix(nside) == 12 * nside**2 for nside in [2**n for n in range(12)]])
True

>>> hp.nside2npix(7)
Traceback (most recent call last):
...
ValueError: Given number is not a valid nside parameter (must be a power of 2)
```

### healpy.pixelfunc.npix2nside

healpy.pixelfunc.npix2nside(*npix*)

Give the nside parameter for the given number of pixels.

**Parameters** *npix* : int

the number of pixels

**Returns** *nside* : int

the nside parameter corresponding to npix

#### Notes

Raise a ValueError exception if number of pixel does not correspond to the number of pixel of an healpix map.

#### Examples

```
>>> import healpy as hp
>>> hp.npix2nside(768)
8

>>> np.all([hp.npix2nside(12 * nside**2) == nside for nside in [2**n for n in range(12)]])
True

>>> hp.npix2nside(1000)
Traceback (most recent call last):
...
ValueError: Wrong pixel number (it is not 12*nside**2)
```

### healpy.pixelfunc.nside2resol

healpy.pixelfunc.nside2resol(*nside*, *arcmin=False*)

Give approximate resolution for nside.

Resolution is just the square root of the pixel area, which is a gross approximation given the different pixel shapes

**Parameters** *nside* : int

healpix nside parameter, must be a power of 2

**arcmin** : bool

if True, return resolution in arcmin, otherwise in radian

**Returns** **resol** : float

approximate pixel size in radians or arcmin

### Notes

Raise a `ValueError` exception if `nside` is not valid.

### Examples

```
>>> import healpy as hp
>>> hp.nside2resol(128, arcmin = True)
27.483891294539248

>>> hp.nside2resol(256)
0.0039973699529159707

>>> hp.nside2resol(7)
Traceback (most recent call last):
...
ValueError: Given number is not a valid nside parameter (must be a power of 2)
```

## healpy.pixelfunc.nside2pixarea

`healpy.pixelfunc.nside2pixarea` (*nside*, *degrees=False*)

Give pixel area given `nside`.

**Parameters** **nside** : int

healpix `nside` parameter, must be a power of 2

**degrees** : bool

if True, returns pixel area in square degrees, in square radians otherwise

**Returns** **pixarea** : float

pixel area in square radian or square degree

### Notes

Raise a `ValueError` exception if `nside` is not valid.

### Examples

```
>>> import healpy as hp
>>> hp.nside2pixarea(128, degrees = True)
0.2098234113027917

>>> hp.nside2pixarea(256)
1.5978966540475428e-05
```

```
>>> hp.nside2pixarea(7)
Traceback (most recent call last):
...
ValueError: Given number is not a valid nside parameter (must be a power of 2)
```

### healpy.pixelfunc.isnsideok

healpy.pixelfunc.**isnsideok** (*nside*)

Returns True if nside is a valid nside parameter, False otherwise.

**Parameters** **nside** : int, scalar or array-like

integer value to be tested

**Returns** **ok** : bool, scalar or array-like

True if given value is a valid nside, False otherwise.

#### Examples

```
>>> import healpy as hp
>>> hp.isnsideok(13)
False

>>> hp.isnsideok(32)
True

>>> hp.isnsideok([1, 2, 3, 4, 8, 16])
array([ True,  True, False,  True,  True,  True], dtype=bool)
```

### healpy.pixelfunc.isnpixok

healpy.pixelfunc.**isnpixok** (*npix*)

Return True if npix is a valid value for healpix map size, False otherwise.

**Parameters** **npix** : int, scalar or array-like

integer value to be tested

**Returns** **ok** : bool, scalar or array-like

True if given value is a valid number of pixel, False otherwise

#### Examples

```
>>> import healpy as hp
>>> hp.isnpixok(12)
True

>>> hp.isnpixok(768)
True

>>> hp.isnpixok([12, 768, 1002])
array([ True,  True, False], dtype=bool)
```

## healpy.pixelfunc.get\_map\_size

healpy.pixelfunc.get\_map\_size(*m*)

Returns the npix of a given map (implicit or explicit pixelization).

If map is a dict type, assumes explicit pixelization: use nside key if present, or use nside attribute if present, otherwise use the smallest valid npix given the maximum key value. otherwise assumes implicit pixelization and returns len(m).

**Parameters** *m* : array-like or dict-like

a map with implicit (array-like) or explicit (dict-like) pixelization

**Returns** npix : int

a valid number of pixel

### Notes

In implicit pixelization, raise a ValueError exception if the size of the input is not a valid pixel number.

### Examples

```
>>> import healpy as hp
>>> m = {0: 1, 1: 1, 2: 1, 'nside': 1}
>>> print hp.get_map_size(m)
12

>>> m = {0: 1, 767: 1}
>>> print hp.get_map_size(m)
768

>>> print hp.get_map_size(np.zeros(12 * 8 ** 2))
768
```

## healpy.pixelfunc.get\_min\_valid\_nside

healpy.pixelfunc.get\_min\_valid\_nside(*npix*)

Returns the minimum acceptable nside so that npix <= nside2npix(nside).

**Parameters** npix : int

a minimal number of pixel

**Returns** nside : int

a valid healpix nside so that  $12 * nside ** 2 \geq npix$

### Examples

```
>>> import healpy as hp
>>> hp.pixelfunc.get_min_valid_nside(355)
8
>>> hp.pixelfunc.get_min_valid_nside(768)
8
```



### healpy.pixelfunc.get\_nside

healpy.pixelfunc.get\_nside(*m*)

Return the nside of the given map.

**Parameters** *m* : sequence

the map to get the nside from.

**Returns** *nside* : int

the healpix nside parameter of the map (or sequence of maps)

#### Notes

If the input is a sequence of maps, all of them must have same size. If the input is not a valid map (not a sequence, unvalid number of pixels), a `TypeError` exception is raised.

### healpy.pixelfunc.maptypes

healpy.pixelfunc.maptypes(*m*)

Describe the type of the map (valid, single, sequence of maps). Checks : the number of maps, that all maps have same length and that this length is a valid map size (using `isnpixok()`).

**Parameters** *m* : sequence

the map to get info from

**Returns** *info* : int

-1 if the given object is not a valid map, 0 if it is a single map, *info* > 0 if it is a sequence of maps (*info* is then the number of maps)

#### Examples

```

>>> import healpy as hp
>>> hp.pixelfunc.maptypes(np.arange(12))
0
>>> hp.pixelfunc.maptypes(2)
-1
>>> hp.pixelfunc.maptypes([np.arange(12), np.arange(12)])
2
>>> hp.pixelfunc.maptypes([np.arange(12), np.arange(768)])
-1

```

### healpy.pixelfunc.ud\_grade

healpy.pixelfunc.ud\_grade(*map\_in*, *nside\_out*, *power=False*, *order\_in='RING'*, *order\_out=None*,  
*power=None*, *dtype=None*)

Upgrade or degrade resolution of a map (or list of maps).

**Parameters** *map\_in* : array-like or sequence of array-like

the input map(s) (if a sequence of maps, all must have same size)

*nside\_out* : int

the desired nside of the output map(s)

**pass** : bool

if `True`, in degrading, reject pixels which contains a bad sub\_pixel. Otherwise, estimate average with good pixels

**order\_in, order\_out** : str

pixel ordering of input and output ('RING' or 'NESTED')

**power** : float

if non-zero, multiply the result by  $(\text{nside\_in}/\text{nside\_out})^{**\text{power}}$  For example, `power=-2` keep the sum of the map invariant

**dtype** : type

the type of the output map

**Returns** **map\_out** : array-like or sequence of array-like

the upgraded or degraded map(s)

### Examples

```
>>> import healpy as hp
>>> hp.ud_grade(np.arange(48.), 1)
array([[ 5.5 ,  7.25,  9.  , 10.75, 21.75, 21.75, 23.75, 25.75,
        36.5 , 38.25, 40.  , 41.75])
```

## 3.1.4 Masking pixels

<code>UNSEEN</code>	Special Healpix values for masked pixels.
<code>mask_bad(m[, badval, rtol, atol])</code>	Returns a bool array with <code>True</code> where <code>m</code> is close to <code>badval</code> .
<code>mask_good(m[, badval, rtol, atol])</code>	Returns a bool array with <code>False</code> where <code>m</code> is close to <code>badval</code> .
<code>ma(m[, badval, rtol, atol, copy])</code>	Return map as a masked array, with <code>badval</code> pixels masked.

### healpy.pixelfunc.UNSEEN

`healpy.pixelfunc.UNSEEN`  
alias of `Mock`

### healpy.pixelfunc.mask\_bad

`healpy.pixelfunc.mask_bad(m, badval=<class 'Mock'>, rtol=1e-05, atol=1e-08)`

Returns a bool array with `True` where `m` is close to `badval`.

**Parameters** **m** : a map (may be a sequence of maps)

**badval** : float, optional

The value of the pixel considered as bad (`UNSEEN` by default)

**rtol** : float, optional

The relative tolerance

**atol** : float, optional

The absolute tolerance

**Returns** **mask** :

a bool array with the same shape as the input map, True where input map is close to badval, and False elsewhere.

**See Also:**

`mask_good`, `ma`

### Examples

```
>>> import healpy as hp
>>> m = np.arange(12.)
>>> m[3] = hp.UNSEEN
>>> hp.mask_bad(m)
array([False, False, False,  True, False, False, False, False, False,
        False, False, False], dtype=bool)
```

## healpy.pixelfunc.mask\_good

`healpy.pixelfunc.mask_good(m, badval=<class 'Mock'>, rtol=1e-05, atol=1e-08)`

Returns a bool array with False where m is close to badval.

**Parameters** **m** : a map (may be a sequence of maps)

**badval** : float, optional

The value of the pixel considered as bad (`UNSEEN` by default)

**rtol** : float, optional

The relative tolerance

**atol** : float, optional

The absolute tolerance

**Returns** a bool array with the same shape as the input map, “False“ where input map is :  
close to badval, and “True“ elsewhere. :

**See Also:**

`mask_bad`, `ma`

### Examples

```
>>> import healpy as hp
>>> m = np.arange(12.)
>>> m[3] = hp.UNSEEN
>>> hp.mask_good(m)
array([ True,  True,  True, False,  True,  True,  True,  True,  True,
        True,  True,  True], dtype=bool)
```

## healpy.pixelfunc.ma

`healpy.pixelfunc.ma(m, badval=<class 'Mock'>, rtol=1e-05, atol=1e-08, copy=True)`

Return map as a masked array, with badval pixels masked.

**Parameters** **m** : a map (may be a sequence of maps)

**badval** : float, optional

The value of the pixel considered as bad (UNSEEN by default)

**rtol** : float, optional

The relative tolerance

**atol** : float, optional

The absolute tolerance

**copy** : bool, optional

If True, a copy of the input map is made.

**Returns** a masked array with the same shape as the input map, masked where input map is :

close to badval. :

**See Also:**

`mask_good`, `mask_bad`, `numpy.ma.masked_values`

### Examples

```
>>> import healpy as hp
>>> m = np.arange(12.)
>>> m[3] = hp.UNSEEN
>>> hp.ma(m)
masked_array(data = [0.0 1.0 2.0 -- 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0],
              mask = [False False False  True False False False False False False],
              fill_value = -1.6375e+30)
```

## 3.1.5 Map data manipulation

<code>fit_dipole(m[, nest, bad, gal_cut])</code>	Fit a dipole and a monopole to the map, excluding bad pixels.
<code>fit_monopole(m[, nest, bad, gal_cut])</code>	Fit a monopole to the map, excluding unseen pixels.
<code>remove_dipole(m[, nest, bad, gal_cut, ...])</code>	Fit and subtract the dipole and the monopole from the given map m.
<code>remove_monopole(m[, nest, bad, gal_cut, ...])</code>	Fit and subtract the monopole from the given map m.
<code>get_interp_val(m, theta, phi[, nest])</code>	Return the bi-linear interpolation value of a map using 4 nearest neighbours.

## healpy.pixelfunc.fit\_dipole

`healpy.pixelfunc.fit_dipole(m, nest=False, bad=<class 'Mock'>, gal_cut=0)`

Fit a dipole and a monopole to the map, excluding bad pixels.

**Parameters** **m** : float, array-like

the map to which a dipole is fitted and subtracted

**nest** : bool

if `False` `m` is assumed in RING scheme, otherwise map is NESTED

**bad** : float

bad values of pixel, default to `UNSEEN`.

**gal\_cut** : float

pixels at latitude in `[-gal_cut;+gal_cut]` degrees are not taken into account

**Returns** **res** : tuple of length 2

the monopole value in `res[0]` and the dipole vector (as array) in `res[1]`

**See Also:**

`remove_dipole`, `fit_monopole`, `remove_monopole`

### healpy.pixelfunc.fit\_monopole

`healpy.pixelfunc.fit_monopole(m, nest=False, bad=<class 'Mock'>, gal_cut=0)`

Fit a monopole to the map, excluding unseen pixels.

**Parameters** **m** : float, array-like

the map to which a dipole is fitted and subtracted

**nest** : bool

if `False` `m` is assumed in RING scheme, otherwise map is NESTED

**bad** : float

bad values of pixel, default to `UNSEEN`.

**gal\_cut** : float

pixels at latitude in `[-gal_cut;+gal_cut]` degrees are not taken into account

**Returns** **res**: float :

fitted monopole value

**See Also:**

`fit_dipole`, `remove_monopole`, `remove_monopole`

### healpy.pixelfunc.remove\_dipole

`healpy.pixelfunc.remove_dipole(m, nest=False, bad=<class 'Mock'>, gal_cut=0, fitval=False, copy=True, verbose=False)`

Fit and subtract the dipole and the monopole from the given map `m`.

**Parameters** **m** : float, array-like

the map to which a dipole is fitted and subtracted

**nest** : bool

if `False` `m` is assumed in RING scheme, otherwise map is NESTED

**bad** : float

bad values of pixel, default to `UNSEEN`.

**gal\_cut** : float

pixels at latitude in `[-gal_cut;+gal_cut]` are not taken into account

**fitval** : bool

whether to return or not the fitted values of monopole and dipole

**copy** : bool

whether to modify input map or not (by default, make a copy)

**verbose** : bool

print values of monopole and dipole

**Returns** **res** : array or tuple of length 3

if `fitval` is `False`, returns map with monopole and dipole subtracted, otherwise, returns map (array, in `res[0]`), monopole (float, in `res[1]`), dipole\_vector (array, in `res[2]`)

**See Also:**

`fit_dipole`, `fit_monopole`, `remove_monopole`

### healpy.pixelfunc.remove\_monopole

`healpy.pixelfunc.remove_monopole(m, nest=False, bad=<class 'Mock'>, gal_cut=0, fitval=False, copy=True, verbose=False)`

Fit and subtract the monopole from the given map `m`.

**Parameters** **m** : float, array-like

the map to which a monopole is fitted and subtracted

**nest** : bool

if `False` `m` is assumed in RING scheme, otherwise map is NESTED

**bad** : float

bad values of pixel, default to `UNSEEN`.

**gal\_cut** : float

pixels at latitude in `[-gal_cut;+gal_cut]` are not taken into account

**fitval** : bool

whether to return or not the fitted value of monopole

**copy** : bool

whether to modify input map or not (by default, make a copy)

**verbose: bool** :

whether to print values of monopole

**Returns** **res** : array or tuple of length 3

if `fitval` is `False`, returns map with monopole subtracted, otherwise, returns map (array, in `res[0]`) and monopole (float, in `res[1]`)

**See Also:**

`fit_dipole`, `fit_monopole`, `remove_dipole`

## healpy.pixelfunc.get\_interp\_val

`healpy.pixelfunc.get_interp_val(m, theta, phi, nest=False)`  
 Return the bi-linear interpolation value of a map using 4 nearest neighbours.

**Parameters** `m` : array-like

an healpix map

**theta, phi** : float, scalar or array-like

angular coordinates of point at which to interpolate the map

**nest** : bool

if True, the is assumed to be in NESTED ordering.

**Returns** `val` : float, scalar or array-like

the interpolated value(s), usual numpy broadcasting rules apply.

**See Also:**

`get_neighbours`, `get_all_neighbours`

### Examples

```
>>> import healpy as hp
>>> hp.get_interp_val(np.arange(12.), np.pi/2, 0)
4.0
>>> hp.get_interp_val(np.arange(12.), np.linspace(0, np.pi, 10), 0)
array([ 1.5          ,  1.5          ,  1.5          ,  2.20618428,  3.40206143,
        5.31546486,  7.94639458,  9.5          ,  9.5          ,  9.5          ])
```

## 3.2 sphtfunc – Spherical harmonic transforms

### 3.2.1 From map to spherical harmonics

<code>synfast(cls, nside[, lmax, mmax, alm, pol, ...])</code>	Create a map(s) from cl(s).
<code>map2alm(maps[, lmax, mmax, iter, pol, ...])</code>	Computes the alm of an Healpix map.

## healpy.sphtfunc.synfast

`healpy.sphtfunc.synfast(cls, nside, lmax=None, mmax=None, alm=False, pol=True, pixwin=False, fwhm=0.0, sigma=None, new=False)`  
 Create a map(s) from cl(s).

**Parameters** `cls` : array or tuple of array

A cl or a list of cl (either 4 or 6, see `synalm()`)

**nside** : int, scalar

The nside of the output map(s)

**lmax** : int, scalar, optional

Maximum l for alm. Default: 3\*nside-1

**mmax** : int, scalar, optional

Maximum m for alm. Default: 3\*nside-1

**alm** : bool, scalar, optional

If True, return also alm(s). Default: False.

**pol** : bool, optional

If True, assumes input cls are TEB and correlation. Output will be TQU maps. (input must be 1, 4 or 6 cl's) If False, fields are assumed to be described by spin 0 spherical harmonics. (input can be any number of cl's) If there is only one input cl, it has no effect. Default: True.

**pixwin** : bool, scalar, optional

If True, convolve the alm by the pixel window function. Default: False.

**fwhm** : float, scalar, optional

The fwhm of the Gaussian used to smooth the map (applied on alm) [in radians]

**sigma** : float, scalar, optional

The sigma of the Gaussian used to smooth the map (applied on alm) [in radians]

**Returns** **maps** : array or tuple of arrays

The output map (possibly list of maps if polarized input). or, if alm is True, a tuple of (map,alm) (alm possibly a list of alm if polarized input)

## Notes

The order of the spectra will change in a future release. The new= parameter help to make the transition smoother. You can start using the new order by setting new=True. In the next version of healpy, the default will be new=True. This change is done for consistency between the different tools (alm2cl, synfast, anafast). In the new order, the spectra are ordered by diagonal of the correlation matrix. Eg, if fields are T, E, B, the spectra are TT, EE, BB, TE, EB, TB with new=True, and TT, TE, TB, EE, EB, BB if new=False.

## healpy.sphtfunc.map2alm

`healpy.sphtfunc.map2alm(maps, lmax=None, mmax=None, iter=3, pol=True, use_weights=False, regression=True, datapath=None)`

Computes the alm of an Healpix map.

**Parameters** **maps** : array-like, shape (Npix,) or (n, Npix)

The input map or a list of n input maps.

**lmax** : int, scalar, optional

Maximum l of the power spectrum. Default: 3\*nside-1

**mmax** : int, scalar, optional

Maximum m of the alm. Default: lmax

**iter** : int, scalar, optional

Number of iteration (default: 3)

**pol** : bool, optional



If True, assumes input maps are TQU. Output will be TEB alm's. (input must be 1 or 3 maps) If False, apply spin 0 harmonic transform to each map. (input can be any number of maps) If there is only one input map, it has no effect. Default: True.

**use\_weights:** bool, scalar, optional :

If True, use the ring weighting. Default: False.

**regression:** bool, scalar, optional :

If True, subtract map average before computing alm. Default: True.

**datapath :** None or str, optional

If given, the directory where to find the weights data.

**Returns** **alms** : array or tuple of array

alm or a tuple of 3 alm (almT, almE, almB) if polarized input.

### Notes

The pixels which have the special *UNSEEN* value are replaced by zeros before spherical harmonic transform. They are converted back to *UNSEEN* value, so that the input maps are not modified. Each map have its own, independent mask.

## 3.2.2 From spherical harmonics to map

<code>anafast(map1[, map2, nspec, lmax, mmax, ...])</code>	Computes the power spectrum of an Healpix map, or the cross-spectrum
<code>alm2map(alms, nside[, lmax, mmax, pixwin, ...])</code>	Computes an Healpix map given the alm.
<code>alm2map_der1(alm, nside[, lmax, mmax])</code>	Computes an Healpix map and its first derivatives given the alm.

### healpy.sphtfunc.anafast

`healpy.sphtfunc.anafast` (*map1*, *map2=None*, *nspec=None*, *lmax=None*, *mmax=None*, *iter=3*, *alm=False*, *pol=True*, *use\_weights=False*, *regression=True*, *datapath=None*)

Computes the power spectrum of an Healpix map, or the cross-spectrum between two maps if *map2* is given.

**Parameters** **map1** : float, array-like shape (Npix,) or (3, Npix)

Either an array representing a map, or a sequence of 3 arrays representing I, Q, U maps

**map2** : float, array-like shape (Npix,) or (3, Npix)

Either an array representing a map, or a sequence of 3 arrays representing I, Q, U maps

**nspec** : None or int, optional

The number of spectra to return. If None, returns all, otherwise returns `cls[:nspec]`

**lmax** : int, scalar, optional

Maximum l of the power spectrum (default: 3\*nside-1)

**mmax** : int, scalar, optional

Maximum m of the alm (default: lmax)

**iter** : int, scalar, optional

Number of iteration (default: 3)

**alm** : bool, scalar, optional

If True, returns both cl and alm, otherwise only cl is returned

**pol** : bool, optional

If True, assumes input maps are TQU. Output will be TEB cl's and correlations (input must be 1 or 3 maps). If False, maps are assumed to be described by spin 0 spherical harmonics. (input can be any number of maps) If there is only one input map, it has no effect. Default: True.

**regression** : bool, scalar, optional

If True, map average is removed before computing alm. Default: True.

**datapath** : None or str, optional

If given, the directory where to find the weights data.

**Returns** **res** : array or sequence of arrays

If *alm* is False, returns cl or a list of cl's (TT, EE, BB, TE, EB, TB for polarized input map) Otherwise, returns a tuple (cl, alm), where cl is as above and alm is the spherical harmonic transform or a list of almT, almE, almB for polarized input

## healpy.sphtfunc.alm2map

`healpy.sphtfunc.alm2map(alm, nside, lmax=None, mmax=None, pixwin=False, fwhm=0.0, sigma=None, invert=False, pol=True, inplace=False)`

Computes an Healpix map given the alm.

The alm are given as a complex array. You can specify lmax and mmax, or they will be computed from array size (assuming lmax==mmax).

**Parameters** **alms** : complex, array or sequence of arrays

A complex array or a sequence of complex arrays. Each array must have a size of the form:  $mmax * (2 * lmax + 1 - mmax) / 2 + lmax + 1$

**nside** : int, scalar

The nside of the output map.

**lmax** : None or int, scalar, optional

Explicitly define lmax (needed if mmax!=lmax)

**mmax** : None or int, scalar, optional

Explicitly define mmax (needed if mmax!=lmax)

**pixwin** : bool, optional

Smooth the alm using the pixel window functions. Default: False.

**fwhm** : float, scalar, optional

The fwhm of the Gaussian used to smooth the map (applied on alm) [in radians]

**sigma** : float, scalar, optional

The sigma of the Gaussian used to smooth the map (applied on alm) [in radians]

**invert** : bool, optional

If True, alms are divided by Gaussian beam function (un-smooth). Otherwise, alms are multiplied by Gaussian beam function (smooth). Default: False.

**pol** : bool, optional

If True, assumes input alms are TEB. Output will be TQU maps. (input must be 1 or 3 alms) If False, apply spin 0 harmonic transform to each alm. (input can be any number of alms) If there is only one input alm, it has no effect. Default: True.

**inplace** : bool, optional

If True, input alms may be modified by pixel window function and beam smoothing (if alm(s) are complex128 contiguous arrays). Otherwise, input alms are not modified. A copy is made if needed to apply beam smoothing or pixel window.

**Returns** **maps** : array or list of arrays

An Healpix map in RING scheme at nside or a list of T,Q,U maps (if polarized input)

### healpy.sphtfunc.alm2map\_der1

`healpy.sphtfunc.alm2map_der1(alm, nside, lmax=None, mmax=None)`

Computes an Healpix map and its first derivatives given the alm.

The alm are given as a complex array. You can specify lmax and mmax, or they will be computed from array size (assuming lmax==mmax).

**Parameters** **alm** : array, complex

A complex array of alm. Size must be of the form mmax(lmax-mmax+1)/2+lmax

**nside** : int

The nside of the output map.

**lmax** : None or int, optional

Explicitly define lmax (needed if mmax!=lmax)

**mmax** : None or int, optional

Explicitly define mmax (needed if mmax!=lmax)

**Returns** **m, d\_theta, d\_phi** : tuple of arrays

The maps corresponding to alm, and its derivatives with respect to theta and phi.

## 3.2.3 Spherical harmonic transform tools

<code>smoothing(maps[, fwhm, sigma, invert, pol, ...])</code>	Smooth a map with a Gaussian symmetric beam.
<code>smoothalm(alms[, fwhm, sigma, invert, pol, ...])</code>	Smooth alm with a Gaussian symmetric beam function.
<code>alm2cl(alms1[, alms2, lmax, mmax, lmax_out, ...])</code>	Computes (cross-)spectra from alm(s).
<code>synalm(cls[, lmax, mmax, new])</code>	Generate a set of alm given cl.
<code>almxfl(alm, fl[, mmax, inplace])</code>	Multiply alm by a function of l.
<code>pixwin(nside[, pol])</code>	Return the pixel window function for the given nside.
<code>Alm()</code>	This class provides some static methods for alm index computation.

## healpy.sphtfunc.smoothing

`healpy.sphtfunc.smoothing`(*maps*, *fwhm*=0.0, *sigma*=None, *invert*=False, *pol*=True, *iter*=3, *lmax*=None, *mmax*=None, *use\_weights*=False, *regression*=True, *datapath*=None)

Smooth a map with a Gaussian symmetric beam.

**Parameters** **maps** : array or sequence of 3 arrays

Either an array representing one map, or a sequence of 3 arrays representing 3 maps

**fwhm** : float, optional

The full width half max parameter of the Gaussian. Default:0.0

**sigma** : float, optional

The sigma of the Gaussian. Override fwhm.

**invert** : bool, optional

If True, alms are divided by Gaussian beam function (un-smooth). Otherwise, alms are multiplied by Gaussian beam function (smooth). Default: False.

**pol** : bool, optional

If True, assumes input maps are TQU. Output will be TQU maps. (input must be 1 or 3 alms) If False, each map is assumed to be a spin 0 map and is treated independently (input can be any number of alms). If there is only one input map, it has no effect. Default: True.

**iter** : int, scalar, optional

Number of iteration (default: 3)

**lmax** : int, scalar, optional

Maximum l of the power spectrum. Default: 3\*nside-1

**mmax** : int, scalar, optional

Maximum m of the alm. Default: lmax

**use\_weights**: bool, scalar, optional :

If True, use the ring weighting. Default: False.

**regression**: bool, scalar, optional :

If True, subtract map average before computing alm. Default: True.

**datapath** : None or str, optional

If given, the directory where to find the weights data.

**Returns** **maps** : array or list of 3 arrays

The smoothed map(s)

## healpy.sphtfunc.smoothalm

`healpy.sphtfunc.smoothalm`(*alms*, *fwhm*=0.0, *sigma*=None, *invert*=False, *pol*=True, *mmax*=None, *verbose*=False, *inplace*=True)

Smooth alm with a Gaussian symmetric beam function.

**Parameters** **alms** : array or sequence of 3 arrays

Either an array representing one alm, or a sequence of arrays. See *pol* parameter.

**fwhm** : float, optional

The full width half max parameter of the Gaussian. Default:0.0 [in radians]

**sigma** : float, optional

The sigma of the Gaussian. Override fwhm. [in radians]

**invert** : bool, optional

If True, alms are divided by Gaussian beam function (un-smooth). Otherwise, alms are multiplied by Gaussian beam function (smooth). Default: False.

**pol** : bool, optional

If True, assumes input alms are TEB. Output will be TQU maps. (input must be 1 or 3 alms) If False, apply spin 0 harmonic transform to each alm. (input can be any number of alms) If there is only one input alm, it has no effect. Default: True.

**mmax** : None or int, optional

The maximum m for alm. Default: mmax=lmax

**inplace** : bool, optional

If True, the alm's are modified inplace if they are contiguous arrays of type complex128. Otherwise, a copy of alm is made. Default: True.

**verbose** : bool, optional

If True prints diagnostic information. Default: False

**Returns** **alms** : array or sequence of 3 arrays

The smoothed alm. If alm[i] is a contiguous array of type complex128, and *inplace* is True the smoothing is applied inplace. Otherwise, a copy is made.

## healpy.sphtfunc.alm2cl

`healpy.sphtfunc.alm2cl(alm1, alms2=None, lmax=None, mmax=None, lmax_out=None, nspec=None)`

Computes (cross-)spectra from alm(s). If alm2 is given, cross-spectra between alm and alm2 are computed. If alm (and alm2 if provided) contains n alm, then n(n+1)/2 auto and cross-spectra are returned.

**Parameters** **alm** : complex, array or sequence of arrays

The alm from which to compute the power spectrum. If n>=2 arrays are given, computes both auto- and cross-spectra.

**alms2** : complex, array or sequence of 3 arrays, optional

If provided, computes cross-spectra between alm and alm2. Default: alm2=alm, so auto-spectra are computed.

**lmax** : None or int, optional

The maximum l of the input alm. Default: computed from size of alm and mmax\_in

**mmax** : None or int, optional

The maximum m of the input alm. Default: assume mmax\_in = lmax\_in

**lmax\_out** : None or int, optional

The maximum  $l$  of the returned spectra. By default: the  $l_{\max}$  of the given  $alm(s)$ .

**nspec** : None or int, optional

The number of spectra to return. None means all, otherwise returns  $cl[:nspec]$

**Returns** **cl** : array or tuple of  $n(n+1)/2$  arrays

the spectrum  $\langle alm \times alm2 \rangle$  if  $alm$  (and  $alm2$ ) is one  $alm$ , or the auto- and cross-spectra  $\langle alm[i] \times alm2[j] \rangle$  if  $alm$  (and  $alm2$ ) contains more than one spectra. If more than one spectrum is returned, they are ordered by diagonal. For example, if  $*alm$  is  $almT$ ,  $almE$ ,  $almB$ , then the returned spectra are: TT, EE, BB, TE, EB, TB.

## healpy.sphtfunc.synalm

`healpy.sphtfunc.synalm(cls, lmax=None, mmax=None, new=False)`

Generate a set of  $alm$  given  $cl$ . The  $cl$  are given as a float array. Corresponding  $alm$  are generated. If  $l_{\max}$  is None, it is assumed  $l_{\max}=cl.size-1$ . If  $m_{\max}$  is None, it is assumed  $m_{\max}=l_{\max}$ .

**Parameters** **cls** : float, array or tuple of arrays

Either one  $cl$  (1D array) or a tuple of either 4  $cl$  or of  $n(n+1)/2$   $cl$ . Some of the  $cl$  may be None, implying no cross-correlation. See *new* parameter.

**lmax** : int, scalar, optional

The  $l_{\max}$  (if None or  $<0$ , the largest size-1 of  $cls$ )

**mmax** : int, scalar, optional

The  $m_{\max}$  (if None or  $<0$ ,  $=l_{\max}$ )

**new** : bool, optional

If True, use the new ordering of  $cl$ 's, ie by diagonal (e.g. TT, EE, BB, TE, EB, TB or TT, EE, BB, TE if 4  $cl$  as input). If False, use the old ordering, ie by row (e.g. TT, TE, TB, EE, EB, BB or TT, TE, EE, BB if 4  $cl$  as input).

**Returns** **alms** : array or list of arrays

the generated  $alm$  if one spectrum is given, or a list of  $n$   $alms$  (with  $n(n+1)/2$  the number of input  $cl$ , or  $n=3$  if there are 4 input  $cl$ ).

## Notes

The order of the spectra will change in a future release. The *new=* parameter help to make the transition smoother. You can start using the new order by setting *new=True*. In the next version of healpy, the default will be *new=True*. This change is done for consistency between the different tools (*alm2cl*, *synfast*, *anafast*). In the new order, the spectra are ordered by diagonal of the correlation matrix. Eg, if fields are T, E, B, the spectra are TT, EE, BB, TE, EB, TB with *new=True*, and TT, TE, TB, EE, EB, BB if *new=False*.

## healpy.sphtfunc.almxfl

`healpy.sphtfunc.almxfl(alm, fl, mmax=None, inplace=False)`

Multiply  $alm$  by a function of  $l$ . The function is assumed to be zero where not defined.

**Parameters** **alm** : array

The  $alm$  to multiply

**fl** : array

The function (at  $l=0..fl.size-1$ ) by which alm must be multiplied.

**mmax** : None or int, optional

The maximum m defining the alm layout. Default: lmax.

**inplace** : bool, optional

If True, modify the given alm, otherwise make a copy before multiplying.

**Returns alm** : array

The modified alm, either a new array or a reference to input alm, if inplace is True.

## healpy.sphtfunc.pixwin

healpy.sphtfunc.**pixwin** (*nside*, *pol=False*)

Return the pixel window function for the given nside.

**Parameters nside** : int

The nside for which to return the pixel window function

**pol** : bool, optional

If True, return also the polar pixel window. Default: False

**Returns pw or pwT,pwP** : array or tuple of 2 arrays

The temperature pixel window function, or a tuple with both temperature and polarisation pixel window functions.

## healpy.sphtfunc.Alm

class healpy.sphtfunc.**Alm**

This class provides some static methods for alm index computation.

### Methods

<code>getlm(lmax[, i])</code>	Get the l and m from index and lmax.
<code>getidx(lmax, l, m)</code>	Returns index corresponding to (l,m) in an array describing alm up to lmax.
<code>getsize(lmax[, mmax])</code>	Returns the size of the array needed to store alm up to <i>lmax</i> and <i>mmax</i>
<code>getlmax(s[, mmax])</code>	Returns the lmax corresponding to a given array size.

### healpy.sphtfunc.Alm.getlm

static Alm.**getlm** (*lmax*, *i=None*)

Get the l and m from index and lmax.

**Parameters lmax** : int

The maximum l defining the alm layout

**i** : int or None

The index for which to compute the l and m. If None, the function return l and m for

`i=0..Alm.getsize(lmax)`

### healpy.sphtfunc.Alm.getidx

**static** `Alm.getidx(lmax, l, m)`

Returns index corresponding to (l,m) in an array describing alm up to lmax.

**Parameters** `lmax` : int

The maximum l, defines the alm layout

`l` : int

The l for which to get the index

`m` : int

The m for which to get the index

**Returns** `idx` : int

The index corresponding to (l,m)

### healpy.sphtfunc.Alm.getsize

**static** `Alm.getsize(lmax, mmax=None)`

Returns the size of the array needed to store alm up to *lmax* and *mmax*

**Parameters** `lmax` : int

The maximum l, defines the alm layout

`mmax` : int, optional

The maximum m, defines the alm layout. Default: lmax.

**Returns** `size` : int

The size of the array needed to store alm up to lmax, mmax.

### healpy.sphtfunc.Alm.getlmax

**static** `Alm.getlmax(s, mmax=None)`

Returns the lmax corresponding to a given array size.

**Parameters** `s` : int

Size of the array

`mmax` : None or int, optional

The maximum m, defines the alm layout. Default: lmax.

**Returns** `lmax` : int

The maximum l of the array, or -1 if it is not a valid size.



## 3.3 visufunc – Visualisation

### 3.3.1 Map projections

<code>mollview([map, fig, rot, coord, unit, ...])</code>	Plot an healpix map (given as an array) in Mollweide projection.
<code>gnomview([map, fig, rot, coord, unit, ...])</code>	Plot an healpix map (given as an array) in Gnomonic projection.
<code>cartview([map, fig, rot, zat, coord, unit, ...])</code>	Plot an healpix map (given as an array) in Cartesian projection.

#### healpy.visufunc.mollview

`healpy.visufunc.mollview` (*map=None, fig=None, rot=None, coord=None, unit='', xsize=800, title='Mollweide view', nest=False, min=None, max=None, flip='astro', remove\_dip=False, remove\_mono=False, gal\_cut=0, format='%g', format2='%g', cbar=True, cmap=None, notext=False, norm=None, hold=False, margins=None, sub=None*)

Plot an healpix map (given as an array) in Mollweide projection.

**Parameters** **map** : float, array-like or None

An array containing the map. If None, will display a blank map, useful for overplotting.

**fig** : int or None, optional

The figure number to use. Default: create a new figure

**rot** : scalar or sequence, optional

Describe the rotation to apply. In the form (lon, lat, psi) (unit: degrees) : the point at longitude *lon* and latitude *lat* will be at the center. An additional rotation of angle *psi* around this direction is applied.

**coord** : sequence of character, optional

Either one of 'G', 'E' or 'C' to describe the coordinate system of the map, or a sequence of 2 of these to rotate the map from the first to the second coordinate system.

**unit** : str, optional

A text describing the unit of the data. Default: ''

**xsize** : int, optional

The size of the image. Default: 800

**title** : str, optional

The title of the plot. Default: 'Mollweide view'

**nest** : bool, optional

If True, ordering scheme is NESTED. Default: False (RING)

**min** : float, optional

The minimum range value

**max** : float, optional

The maximum range value

**flip** : {'astro', 'geo'}, optional

Defines the convention of projection : ‘astro’ (default, east towards left, west towards right) or ‘geo’ (east towards right, west towards left)

**remove\_dip** : bool, optional

If `True`, remove the dipole+monopole

**remove\_mono** : bool, optional

If `True`, remove the monopole

**gal\_cut** : float, scalar, optional

Symmetric galactic cut for the dipole/monopole fit. Removes points in latitude range `[-gal_cut, +gal_cut]`

**format** : str, optional

The format of the scale label. Default: ‘%g’

**format2** : str, optional

Format of the pixel value under mouse. Default: ‘%g’

**cbar** : bool, optional

Display the colorbar. Default: `True`

**notext** : bool, optional

If `True`, no text is printed around the map

**norm** : {‘hist’, ‘log’, `None`}

Color normalization, `hist=` histogram equalized color mapping, `log=` logarithmic color mapping, default: `None` (linear color mapping)

**hold** : bool, optional

If `True`, replace the current Axes by a `MollweideAxes`. use this if you want to have multiple maps on the same figure. Default: `False`

**sub** : int, scalar or sequence, optional

Use only a zone of the current figure (same syntax as `subplot`). Default: `None`

**margins** : `None` or sequence, optional

Either `None`, or a sequence (`left,bottom,right,top`) giving the margins on left,bottom,right and top of the axes. Values are relative to figure (0-1). Default: `None`

#### See Also:

`gnomview`, `cartview`

### healpy.visufunc.gnomview

```
healpy.visufunc.gnomview(map=None, fig=None, rot=None, coord=None, unit='', xsize=200,
                        ysize=None, reso=1.5, degree=False, title='Gnomonic view', nest=False,
                        remove_dip=False, remove_mono=False, gal_cut=0, min=None,
                        max=None, flip='astro', format='%.3g', cbar=True, cmap=None,
                        norm=None, hold=False, sub=None, margins=None, notext=False)
```

Plot an healpix map (given as an array) in Gnomonic projection.

**Parameters** **map** : array-like

The map to project. If None, use a blank map, useful for overplotting.

**fig** : None or int, optional

A figure number. Default: None= create a new figure

**rot** : scalar or sequence, optional

Describe the rotation to apply. In the form (lon, lat, psi) (unit: degrees) : the point at longitude *lon* and latitude *lat* will be at the center. An additional rotation of angle *psi* around this direction is applied.

**coord** : sequence of character, optional

Either one of 'G', 'E' or 'C' to describe the coordinate system of the map, or a sequence of 2 of these to rotate the map from the first to the second coordinate system.

**unit** : str, optional

A text describing the unit of the data. Default: ''

**xsize** : int, optional

The size of the image. Default: 200

**ysize** : None or int, optional

The size of the image. Default: None= xsize

**reso** : float, optional

Resolution (in arcmin if degree is False). Default: 1.5 arcmin

**degree** : bool, optional

if True, reso is in degree. Default: False

**title** : str, optional

The title of the plot. Default: 'Gnomonic view'

**nest** : bool, optional

If True, ordering scheme is NESTED. Default: False (RING)

**min** : float, scalar, optional

The minimum range value

**max** : float, scalar, optional

The maximum range value

**flip** : {'astro', 'geo'}, optional

Defines the convention of projection : 'astro' (default, east towards left, west towards right) or 'geo' (east towards right, west towards left)

**remove\_dip** : bool, optional

If True, remove the dipole+monopole

**remove\_mono** : bool, optional

If True, remove the monopole

**gal\_cut** : float, scalar, optional

Symmetric galactic cut for the dipole/monopole fit. Removes points in latitude range [-gal\_cut, +gal\_cut]

**format** : str, optional

The format of the scale label. Default: ‘%g’

**hold** : bool, optional

If True, replace the current Axes by a MollweideAxes. use this if you want to have multiple maps on the same figure. Default: False

**sub** : int or sequence, optional

Use only a zone of the current figure (same syntax as subplot). Default: None

**margins** : None or sequence, optional

Either None, or a sequence (left,bottom,right,top) giving the margins on left,bottom,right and top of the axes. Values are relative to figure (0-1). Default: None

**notext: bool, optional :**

If True: do not add resolution info text. Default=False

**See Also:**

`mollview`, `cartview`

## healpy.visufunc.cartview

```
healpy.visufunc.cartview(map=None, fig=None, rot=None, zat=None, coord=None, unit='',
                           xsize=800, ysize=None, lonra=None, latra=None, title='Cartesian
                           view', nest=False, remove_dip=False, remove_mono=False, gal_cut=0,
                           min=None, max=None, flip='astro', format='%.3g', cbar=True,
                           cmap=None, norm=None, aspect=None, hold=False, sub=None,
                           margins=None, notext=False)
```

Plot an healpix map (given as an array) in Cartesian projection.

**Parameters** **map** : float, array-like or None

An array containing the map. If None, will display a blank map, useful for overplotting.

**fig** : int or None, optional

The figure number to use. Default: create a new figure

**rot** : scalar or sequence, optional

Describe the rotation to apply. In the form (lon, lat, psi) (unit: degrees) : the point at longitude *lon* and latitude *lat* will be at the center. An additional rotation of angle *psi* around this direction is applied.

**coord** : sequence of character, optional

Either one of ‘G’, ‘E’ or ‘C’ to describe the coordinate system of the map, or a sequence of 2 of these to rotate the map from the first to the second coordinate system.

**unit** : str, optional

A text describing the unit of the data. Default: ‘’

**xsize** : int, optional

The size of the image. Default: 800

**lonra** : sequence, optional

Range in longitude. Default: [-180,180]

**latra** : sequence, optional

Range in latitude. Default: [-90,90]

**title** : str, optional

The title of the plot. Default: 'Mollweide view'

**nest** : bool, optional

If `True`, ordering scheme is NESTED. Default: `False` (RING)

**min** : float, optional

The minimum range value

**max** : float, optional

The maximum range value

**flip** : { 'astro', 'geo' }, optional

Defines the convention of projection : 'astro' (default, east towards left, west towards right) or 'geo' (east towards right, west towards left)

**remove\_dip** : bool, optional

If `True`, remove the dipole+monopole

**remove\_mono** : bool, optional

If `True`, remove the monopole

**gal\_cut** : float, scalar, optional

Symmetric galactic cut for the dipole/monopole fit. Removes points in latitude range [-gal\_cut, +gal\_cut]

**format** : str, optional

The format of the scale label. Default: '%g'

**cbar** : bool, optional

Display the colorbar. Default: `True`

**notext** : bool, optional

If `True`, no text is printed around the map

**norm** : { 'hist', 'log', `None` }, optional

Color normalization, hist= histogram equalized color mapping, log= logarithmic color mapping, default: `None` (linear color mapping)

**hold** : bool, optional

If `True`, replace the current Axes by a CartesianAxes. use this if you want to have multiple maps on the same figure. Default: `False`

**sub** : int, scalar or sequence, optional

Use only a zone of the current figure (same syntax as subplot). Default: `None`

**margins** : `None` or sequence, optional

Either None, or a sequence (left,bottom,right,top) giving the margins on left,bottom,right and top of the axes. Values are relative to figure (0-1). Default: None

**See Also:**

`mollview`, `gnomview`

### 3.3.2 Graticules

<code>graticule([dpar, dmer, coord, local])</code>	Draw a graticule on the current Axes.
<code>delgraticules()</code>	Delete all graticules previously created on the Axes.

#### healpy.visufunc.graticule

`healpy.visufunc.graticule` (*dpar=None, dmer=None, coord=None, local=None, \*\*kws*)

Draw a graticule on the current Axes.

**Parameters** *dpar, dmer* : float, scalars

Interval in degrees between meridians and between parallels

**coord** : { 'E', 'G', 'C' }

The coordinate system of the graticule (make rotation if needed, using coordinate system of the map if it is defined).

**local** : bool

If True, draw a local graticule (no rotation is performed, useful for a gnomonic view, for example)

**See Also:**

`delgraticules`

**Notes**

Other keyword parameters will be transmitted to the `projplot` function.

#### healpy.visufunc.delgraticules

`healpy.visufunc.delgraticules` ()

Delete all graticules previously created on the Axes.

**See Also:**

`graticule`

### 3.3.3 Tracing lines or points

<code>projplot(*args, **kws)</code>	<code>projplot</code> is a wrapper around <code>matplotlib.Axes.plot()</code> to take into account the
<code>projscatter(*args, **kws)</code>	<code>Projscatter</code> is a wrapper around <code>matplotlib.Axes.scatter()</code> to take into account the
<code>projtext(*args, **kws)</code>	<code>Projtext</code> is a wrapper around <code>matplotlib.Axes.text()</code> to take into account the

## healpy.visufunc.projplot

healpy.visufunc.**projplot**(\*args, \*\*kws)

projplot is a wrapper around `matplotlib.Axes.plot()` to take into account the spherical projection.

You can call this function as:

```

projplot(theta, phi)           # plot a line going through points at coord (theta, phi)
projplot(theta, phi, 'bo')     # plot 'o' in blue at coord (theta, phi)
projplot(thetaphi)            # plot a line going through points at coord (thetaphi[0], thetaphi[1])
projplot(thetaphi, 'bx')      # idem but with blue 'x'

```

**Parameters** **theta, phi** : float, array-like

Coordinates of point to plot. Can be put into one 2-d array, first line is then *theta* and second line is *phi*. See *lonlat* parameter for unit.

**fmt** : str

A format string (see `matplotlib.Axes.plot()` for details)

**lonlat** : bool, optional

If True, theta and phi are interpreted as longitude and latitude in degree, otherwise, as colatitude and longitude in radian

**coord** : {'E', 'G', 'C', None}

The coordinate system of the points, only used if the coordinate coordinate system of the Axes has been defined and in this case, a rotation is performed

**rot** : None or sequence

rotation to be applied =(lon, lat, psi) : lon, lat will be position of the new Z axis, and psi is rotation around this axis, all in degree. if None, no rotation is performed

**direct** : bool

if True, the rotation to center the projection is not taken into account

**See Also:**

`projscatter`, `projtext`

### Notes

Other keywords are passed to `matplotlib.Axes.plot()`.

## healpy.visufunc.projscatter

healpy.visufunc.**projscatter**(\*args, \*\*kws)

Projscatter is a wrapper around `matplotlib.Axes.scatter()` to take into account the spherical projection.

You can call this function as:

```

projscatter(theta, phi)       # plot points at coord (theta, phi)
projplot(thetaphi)           # plot points at coord (thetaphi[0], thetaphi[1])

```

**Parameters** **theta, phi** : float, array-like

Coordinates of point to plot. Can be put into one 2-d array, first line is then *theta* and second line is *phi*. See *lonlat* parameter for unit.

**lonlat** : bool, optional

If True, theta and phi are interpreted as longitude and latitude in degree, otherwise, as colatitude and longitude in radian

**coord** : {'E', 'G', 'C', None}, optional

The coordinate system of the points, only used if the coordinate coordinate system of the axes has been defined and in this case, a rotation is performed

**rot** : None or sequence, optional

rotation to be applied =(lon, lat, psi) : lon, lat will be position of the new Z axis, and psi is rotation around this axis, all in degree. if None, no rotation is performed

**direct** : bool, optional

if True, the rotation to center the projection is not taken into account

**See Also:**

`projplot`, `projtext`

### Notes

Other keywords are passed to `matplotlib.Axes.plot()`.

## healpy.visufunc.projtext

`healpy.visufunc.projtext(*args, **kws)`

Projtext is a wrapper around `matplotlib.Axes.text()` to take into account the spherical projection.

**Parameters** **theta, phi** : float, array-like

Coordinates of point to plot. Can be put into one 2-d array, first line is then *theta* and second line is *phi*. See *lonlat* parameter for unit.

**text** : str

The text to be displayed.

**lonlat** : bool, optional

If True, theta and phi are interpreted as longitude and latitude in degree, otherwise, as colatitude and longitude in radian

**coord** : {'E', 'G', 'C', None}, optional

The coordinate system of the points, only used if the coordinate coordinate system of the axes has been defined and in this case, a rotation is performed

**rot** : None or sequence, optional

rotation to be applied =(lon, lat, psi) : lon, lat will be position of the new Z axis, and psi is rotation around this axis, all in degree. if None, no rotation is performed

**direct** : bool, optional

if True, the rotation to center the projection is not taken into account



**See Also:**`projplot, projscatter`**Notes**

Other keywords are passed to `matplotlib.Axes.text()`.

## 3.4 fitsfunc – Pixelisation related functions

### 3.4.1 Reading/writing maps

---

<code>read_map(filename[, field, dtype, nest, ...])</code>	Read an healpix map from a fits file.
<code>write_map(filename, m[, nest, dtype, ...])</code>	Writes an healpix map into an healpix file.

---

#### `healpy.fitsfunc.read_map`

`healpy.fitsfunc.read_map(filename, field=0, dtype=<class 'Mock'>, nest=False, hdu=1, h=False, verbose=False)`

Read an healpix map from a fits file.

**Parameters** `filename` : str

the fits file name

**field** : int or tuple of int, optional

The column to read. Default: 0. By convention 0 is temperature, 1 is Q, 2 is U. Field can be a tuple to read multiple columns (0,1,2)

**dtype** : data type, optional

Force the conversion to some type. Default: `np.float64`

**nest** : bool, optional

If True return the map in NEST ordering, otherwise in RING ordering; use fits keyword ORDERING to decide whether conversion is needed or not If None, no conversion is performed.

**hdu** : int, optional

the header number to look at (start at 0)

**h** : bool, optional

If True, return also the header. Default: False.

**verbose** : bool, optional

If True, print a number of diagnostic messages

**Returns** `m | (m0, m1, ...) [, header]` : array or a tuple of arrays, optionally with header appended

The map(s) read from the file, and the header if `h` is True.

## healpy.fitsfunc.write\_map

`healpy.fitsfunc.write_map(filename, m, nest=False, dtype=<class 'Mock'>, fits_IDL=True, coord=None)`

Writes an healpix map into an healpix file.

**Parameters** `filename` : str

the fits file name

`m` : array or sequence of 3 arrays

the map to write. Possibly a sequence of 3 maps of same size. They will be considered as I, Q, U maps

`nest` : bool, optional

If False, ordering scheme is NESTED, otherwise, it is RING. Default: RING.

`fits_IDL` : bool, optional

If True, reshapes columns in rows of 1024, otherwise all the data will go in one column. Default: True

`coord` : str

The coordinate system, typically 'E' for Ecliptic, 'G' for Galactic or 'Q' for Equatorial

## 3.4.2 Reading/writing alm

---

`read_alm(filename[, hdu, return_mmax])`      Read alm from a fits file.

`write_alm(filename, alms[, out_dtype, lmax, ...])`      Write alms to a fits file.

---

## healpy.fitsfunc.read\_alm

`healpy.fitsfunc.read_alm(filename, hdu=1, return_mmax=False)`

Read alm from a fits file.

In the fits file, the alm are written with explicit index scheme,  $\text{index} = l**2 + l + m + 1$ , while healpix cxx uses  $\text{index} = m*(2*l_{\text{max}} + 1 - m)/2 + 1$ . The conversion is done in this function.

**Parameters** `filename` : str

The name of the fits file to read

`hdu` : int, optional

The header to read. Start at 0. Default: hdu=1

`return_mmax` : bool, optional

If true, both the alms and mmax is returned in a tuple. Default: return\_mmax=False

**Returns** `alms[, mmax]` : complex array or tuple of a complex array and an int

The alms read from the file and optionally mmax read from the file

## healpy.fitsfunc.write\_alm

`healpy.fitsfunc.write_alm(filename, alms, out_dtype=None, lmax=-1, mmax=-1, mmax_in=-1)`

Write alms to a fits file.

In the fits file the alms are written with explicit index scheme,  $\text{index} = l * l + l + m + 1$ , possibly out of order. By default `write_alm` makes a table with the same precision as the alms. If specified, the `lmax` and `mmax` parameters truncate the input data to include only alms for which  $l \leq l_{\text{max}}$  and  $m \leq m_{\text{max}}$ .

**Parameters** `filename` : str

The filename of the output fits file

**alms** : array, complex

A complex ndarray holding the alms.

**lmax** : int, optional

The maximum  $l$  in the output file

**mmax** : int, optional

The maximum  $m$  in the output file

**out\_dtype** : data type, optional

data type in the output file (must be a numpy dtype). Default: `alms.real.dtype`

**mmax\_in** : int, optional

maximum  $m$  in the input array

### 3.4.3 Reading/writing cl

---

`read_cl(filename[, dtype, h])` Reads Cl from an healpix file, as IDL fits2cl.

`write_cl(filename, cl[, dtype])` Writes Cl into an healpix file, as IDL cl2fits.

---

#### healpy.fitsfunc.read\_cl

`healpy.fitsfunc.read_cl(filename, dtype=<class 'Mock'>, h=False)`

Reads Cl from an healpix file, as IDL fits2cl.

**Parameters** `filename` : str

the fits file name

**dtype** : data type, optional

the data type of the returned array

**Returns** `cl` : array

the cl array, currently TT only

#### healpy.fitsfunc.write\_cl

`healpy.fitsfunc.write_cl(filename, cl, dtype=<class 'Mock'>)`

Writes Cl into an healpix file, as IDL cl2fits.

**Parameters** `filename` : str

the fits file name

**cl** : array

the cl array to write to file, currently TT only

### 3.4.4 Reading/writing column in fits file

<code>mrdfits(filename[, hdu])</code>	Read a table in a fits file.
<code>mwrfits(filename, data[, hdu, colnames, keys])</code>	Write columns to a fits file in a table extension.

#### healpy.fitsfunc.mrdfits

`healpy.fitsfunc.mrdfits(filename, hdu=1)`

Read a table in a fits file.

**Parameters** `filename` : str

The name of the fits file to read

`hdu` : int, optional

The header to read. Start at 0. Default: `hdu=1`

**Returns** `cols` : a list of arrays

A list of column data in the given header

#### healpy.fitsfunc.mwrfits

`healpy.fitsfunc.mwrfits(filename, data, hdu=1, colnames=None, keys=None)`

Write columns to a fits file in a table extension.

**Parameters** `filename` : str

The fits file name

`data` : list of 1D arrays

A list of 1D arrays to write in the table

`hdu` : int, optional

The header where to write the data. Default: 1

`colnames` : list of str

The column names

`keys` : dict-like

A dictionary with keywords to write in the header

### 3.4.5 Helper

---

`getformat(t)` Get the FITS convention format string of data type t.

#### healpy.fitsfunc.getformat

`healpy.fitsfunc.getformat(t)`

Get the FITS convention format string of data type t.

**Parameters** `t` : data type

The data type for which the FITS type is requested

**Returns** `fits_type` : str or None

The FITS string code describing the data type, or None if unknown type.

## 3.5 rotator – Rotation and geometry functions

### 3.5.1 Rotation

<code>Rotator([rot, coord, inv, deg, eulertype])</code>	Rotation operator, including astronomical coordinate systems.
<code>rotateVector(rotmat, vec[, vy, vz, do_rot])</code>	Rotate a vector (or a list of vectors) using the rotation matrix given as first argument
<code>rotateDirection(rotmat, theta[, phi, ...])</code>	Rotate the vector described by angles theta,phi using the rotation matrix given as first argument

#### healpy.rotator.Rotator

**class** `healpy.rotator.Rotator` (*rot=None, coord=None, inv=None, deg=True, eulertype='ZYX'*)

Rotation operator, including astronomical coordinate systems.

This class provides tools for spherical rotations. It is meant to be used in the healpy library for plotting, and for this reason reflects the convention used in the Healpix IDL library.

**Parameters** `rot` : None or sequence

Describe the rotation by its euler angle. See `euler_matrix_new()`.

`coord` : None or sequence of str

Describe the coordinate system transform. If `rot` is also given, the coordinate transform is applied first, and then the rotation.

`inv` : bool

If True, the inverse rotation is defined. (Default: False)

`deg` : bool

If True, angles are assumed to be in degree. (Default: True)

`eulertype` : str

The Euler angle convention used. See `euler_matrix_new()`.

#### Examples

```
>>> r = Rotator(coord=['G','E']) # Transforms galactic to ecliptic coordinates
>>> theta_gal, phi_gal = np.pi/2., 0.
>>> theta_ecl, phi_ecl = r(theta_gal, phi_gal) # Apply the conversion
>>> print theta_ecl, phi_ecl
1.66742286715 -1.62596400306
>>> theta_ecl, phi_ecl = Rotator(coord='ge')(theta_gal, phi_gal) # In one line
>>> print theta_ecl, phi_ecl
1.66742286715 -1.62596400306
>>> vec_gal = np.array([1, 0, 0]) #Using vectors
>>> vec_ecl = r(vec_gal)
>>> print vec_ecl
[-0.05488249 -0.99382103 -0.09647625]
```

## Attributes

<code>mat</code>	The matrix representing the rotation.
<code>coordin</code>	The input coordinate system.
<code>coordout</code>	The output coordinate system.
<code>coordinstr</code>	The input coordinate system in str.
<code>coordoutstr</code>	The output coordinate system in str.
<code>rots</code>	The sequence of rots defining the rotation.
<code>coords</code>	The sequence of coords defining the rotation.

### `healpy.rotator.Rotator.mat`

`Rotator.mat`

The matrix representing the rotation.

### `healpy.rotator.Rotator.coordin`

`Rotator.coordin`

The input coordinate system.

### `healpy.rotator.Rotator.coordout`

`Rotator.coordout`

The output coordinate system.

### `healpy.rotator.Rotator.coordinstr`

`Rotator.coordinstr`

The input coordinate system in str.

### `healpy.rotator.Rotator.coordoutstr`

`Rotator.coordoutstr`

The output coordinate system in str.

### `healpy.rotator.Rotator.rots`

`Rotator.rots`

The sequence of rots defining the rotation.

### `healpy.rotator.Rotator.coords`

`Rotator.coords`

The sequence of coords defining the rotation.

## Methods

<code>I(*args, **kws)</code>	Rotate the given vector or direction using the inverse matrix.
<code>__call__(*args, **kws)</code>	Use the rotator to rotate either spherical coordinates (theta, phi) or a vector (x,y,z).
<code>angle_ref(*args, **kws)</code>	Compute the angle between transverse reference direction of initial and final frames
<code>do_rot(i)</code>	Returns True if rotation is not (close to) identity.
<code>get_inverse()</code>	

### healpy.rotator.Rotator.I

`Rotator.I(*args, **kws)`

Rotate the given vector or direction using the inverse matrix. `rot.I(vec) <==> rot(vec, inv=True)`

### healpy.rotator.Rotator.\_\_call\_\_

`Rotator.__call__(*args, **kws)`

Use the rotator to rotate either spherical coordinates (theta, phi) or a vector (x,y,z). You can use lonlat keyword to use longitude, latitude (in degree) instead of theta, phi (in radian). In this case, returns longitude, latitude in degree.

Accepted forms:

`r(x,y,z)` # x,y,z either scalars or arrays `r(theta,phi)` # theta, phi scalars or arrays `r(lon,lat,lonlat=True)` # lon, lat scalars or arrays `r(vec)` # vec 1-D array with 3 elements, or 2-D array 3xN `r(direction)` # direction 1-D array with 2 elements, or 2xN array

**Parameters** `vec_or_dir` : array or multiple arrays

The direction to rotate. See above for accepted formats.

**lonlat** : bool, optional

If True, assumes the input direction is longitude/latitude in degrees. Otherwise, assumes co-latitude/longitude in radians. Default: False

**inv** : bool, optional

If True, applies the inverse rotation. Default: False.

### healpy.rotator.Rotator.angle\_ref

`Rotator.angle_ref(*args, **kws)`

Compute the angle between transverse reference direction of initial and final frames

For example, if angle of polarisation is psi in initial frame, it will be psi+angle\_ref in final frame.

**Parameters** `dir_or_vec` : array

Direction or vector (see `Rotator.__call__`)

**lonlat**: bool, optional :

If True, assume input is longitude,latitude in degrees. Otherwise, theta,phi in radian. Default: False

**inv** : bool, optional

If True, use the inverse transforms. Default: False

**Returns** **angle** : float, scalar or array

Angle in radian (a scalar or an array if input is a sequence of direction/vector)

### healpy.rotator.Rotator.do\_rot

`Rotator.do_rot(i)`

Returns True if rotation is not (close to) identity.

### healpy.rotator.Rotator.get\_inverse

`Rotator.get_inverse()`

## healpy.rotator.rotateVector

`healpy.rotator.rotateVector(rotmat, vec, vy=None, vz=None, do_rot=True)`

Rotate a vector (or a list of vectors) using the rotation matrix given as first argument.

**Parameters** **rotmat** : float, array-like shape (3,3)

The rotation matrix

**vec** : float, scalar or array-like

The vector to transform (shape (3,) or (3,N)), or x component (scalar or shape (N,)) if vy and vz are given

**vy** : float, scalar or array-like, optional

The y component of the vector (scalar or shape (N,))

**vz** : float, scalar or array-like, optional

The z component of the vector (scalar or shape (N,))

**do\_rot** : bool, optional

if True, really perform the operation, if False do nothing.

**Returns** **vec** : float, array

The component of the rotated vector(s).

**See Also:**

[Rotator](#)

## healpy.rotator.rotateDirection

`healpy.rotator.rotateDirection(rotmat, theta, phi=None, do_rot=True, lonlat=False)`

Rotate the vector described by angles theta,phi using the rotation matrix given as first argument.

**Parameters** **rotmat** : float, array-like shape (3,3)

The rotation matrix

**theta** : float, scalar or array-like



The angle theta (scalar or shape (N,)) or both angles (scalar or shape (2, N)) if phi is not given.

**phi** : float, scalar or array-like, optional

The angle phi (scalar or shape (N,)).

**do\_rot** : bool, optional

if True, really perform the operation, if False do nothing.

**lonlat** : bool

If True, input angles are assumed to be longitude and latitude in degree, otherwise, they are co-latitude and longitude in radians.

**Returns** **angles** : float, array

The angles of describing the rotated vector(s).

**See Also:**

`Rotator`

### 3.5.2 Geometrical helpers

<code>vec2dir(vec[, vy, vz, lonlat])</code>	Transform a vector to angle given by theta,phi.
<code>dir2vec(theta[, phi, lonlat])</code>	Transform a direction theta,phi to a unit vector.
<code>angdist(dir1, dir2[, lonlat])</code>	Returns the angular distance between dir1 and dir2.

#### healpy.rotator.vec2dir

`healpy.rotator.vec2dir (vec, vy=None, vz=None, lonlat=False)`

Transform a vector to angle given by theta,phi.

**Parameters** **vec** : float, scalar or array-like

The vector to transform (shape (3,) or (3,N)), or x component (scalar or shape (N,)) if vy and vz are given

**vy** : float, scalar or array-like, optional

The y component of the vector (scalar or shape (N,))

**vz** : float, scalar or array-like, optional

The z component of the vector (scalar or shape (N,))

**lonlat** : bool, optional

If True, return angles will be longitude and latitude in degree, otherwise, angles will be longitude and co-latitude in radians (default)

**Returns** **angles** : float, array

The angles (unit depending on *lonlat*) in an array of shape (2,) (if scalar input) or (2, N)

**See Also:**

`dir2vec()`, `pixelfunc.ang2vec()`, `pixelfunc.vec2ang()`

## healpy.rotator.dir2vec

`healpy.rotator.dir2vec(theta, phi=None, lonlat=False)`

Transform a direction theta,phi to a unit vector.

**Parameters** **theta** : float, scalar or array-like

The angle theta (scalar or shape (N,)) or both angles (scalar or shape (2, N)) if phi is not given.

**phi** : float, scalar or array-like, optional

The angle phi (scalar or shape (N,)).

**lonlat** : bool

If True, input angles are assumed to be longitude and latitude in degree, otherwise, they are co-latitude and longitude in radians.

**Returns** **vec** : array

The vector(s) corresponding to given angles, shape is (3,) or (3, N).

**See Also:**

`vec2dir()`, `pixelfunc.ang2vec()`, `pixelfunc.vec2ang()`

## healpy.rotator.angdist

`healpy.rotator.angdist(dir1, dir2, lonlat=False)`

Returns the angular distance between dir1 and dir2.

**Parameters** **dir1, dir2** : float, array-like

The directions between which computing the angular distance. Angular if `len(dir) == 2` or vector if `len(dir) == 3`. See *lonlat* for unit

**lonlat** : bool, scalar or sequence

If True, angles are assumed to be longitude and latitude in degree, otherwise they are interpreted as colatitude and longitude in radian. If a sequence, `lonlat[0]` applies to `dir1` and `lonlat[1]` applies to `dir2`.

**Returns** **angles** : float, scalar or array-like

The angle(s) between `dir1` and `dir2` in radian.

## 3.6 projector – Spherical projections

### 3.6.1 Basic classes

<code>SphericalProj([rot, coord, flipconv])</code>	This class defines functions for spherical projection.
<code>GnomonicProj([rot, coord, xsize, ysize, reso])</code>	This class provides class methods for Gnomonic projection.
<code>MollweideProj([rot, coord, xsize])</code>	This class provides class methods for Mollweide projection.
<code>CartesianProj([rot, coord, xsize, ysize, ...])</code>	This class provides class methods for Cartesian projection.

## healpy.projector.SphericalProj

**class** healpy.projector.**SphericalProj** (*rot=None, coord=None, flipconv=None, \*\*kws*)

This class defines functions for spherical projection.

This class contains class method for spherical projection computation. It should not be instantiated. It should be inherited from and methods should be overloaded for desired projection.

### Methods

<code>ang2xy(theta[, phi, lonlat, direct])</code>	From angular direction to position in the projection plane (%s).
<code>get_center([lonlat])</code>	Get the center of the projection.
<code>get_extent()</code>	Get the extension of the projection plane.
<code>get_fov()</code>	Get the field of view in degree of the plane of projection
<code>get_proj_plane_info()</code>	
<code>ij2xy([i, j])</code>	From image array indices to position in projection plane (%s).
<code>mkcoord(coord)</code>	
<code>projmap(map, vec2pix_func[, rot, coord])</code>	Create an array containing the projection of the map.
<code>set_flip(flipconv)</code>	flipconv is either 'astro' or 'geo'. None will be default.
<code>set_proj_plane_info(**kws)</code>	
<code>vec2xy(vx[, vy, vz, direct])</code>	From unit vector direction to position in the projection plane (%s).
<code>xy2ang(x[, y, lonlat, direct])</code>	From position in the projection plane to angular direction (%s).
<code>xy2ij(x[, y])</code>	From position in the projection plane to image array index (%s).
<code>xy2vec(x[, y, direct])</code>	From position in the projection plane to unit vector direction (%s).

### healpy.projector.SphericalProj.ang2xy

**SphericalProj.ang2xy** (*theta, phi=None, lonlat=False, direct=False*)

From angular direction to position in the projection plane (%s).

#### Input:

- theta: if phi is None, theta[0] contains theta, theta[1] contains phi
- phi : if phi is not None, theta,phi are direction
- lonlat: if True, angle are assumed in degree, and longitude, latitude
- flipconv is either 'astro' or 'geo'. None will be default.

#### Return:

- x, y: position in %s plane.

### healpy.projector.SphericalProj.get\_center

**SphericalProj.get\_center** (*lonlat=False*)

Get the center of the projection.

#### Input:

- **lonlat** [if True, will return longitude and latitude in degree,] otherwise, theta and phi in radian

#### Return:

- theta,phi or lonlat depending on lonlat keyword

### healpy.projector.SphericalProj.get\_extent

`SphericalProj.get_extent()`  
 Get the extension of the projection plane.  
**Return:** extent = (left,right,bottom,top)

### healpy.projector.SphericalProj.get\_fov

`SphericalProj.get_fov()`  
 Get the field of view in degree of the plane of projection  
**Return:** fov: the diameter in radian of the field of view

### healpy.projector.SphericalProj.get\_proj\_plane\_info

`SphericalProj.get_proj_plane_info()`

### healpy.projector.SphericalProj.ij2xy

`SphericalProj.ij2xy(i=None, j=None)`  
 From image array indices to position in projection plane (%s).

**Input:**

- if i and j are None, generate arrays of i and j as input
- i : if j is None, i[0], j[1] define array indices in %s image.
- j : if defined, i,j define array indices in image.
- projinfo : additional projection information.

**Return:**

- x,y : position in projection plane.

### healpy.projector.SphericalProj.mkcoord

`SphericalProj.mkcoord(coord)`

### healpy.projector.SphericalProj.projmap

`SphericalProj.projmap(map, vec2pix_func, rot=None, coord=None)`  
 Create an array containing the projection of the map.

**Input:**

- vec2pix\_func: a function taking theta,phi and returning pixel number
- **map:** an array containing the spherical map to project, the pixelisation is described by vec2pix\_func

**Return:**

- a 2D array with the projection of the map.

Note: the Projector must contain information on the array.

### healpy.projector.SphericalProj.set\_flip

SphericalProj.**set\_flip** (*flipconv*)

flipconv is either 'astro' or 'geo'. None will be default.

With 'astro', east is toward left and west toward right. It is the opposite for 'geo'

### healpy.projector.SphericalProj.set\_proj\_plane\_info

SphericalProj.**set\_proj\_plane\_info** (\*\*kws)

### healpy.projector.SphericalProj.vec2xy

SphericalProj.**vec2xy** (vx, vy=None, vz=None, direct=False)

From unit vector direction to position in the projection plane (%s).

#### Input:

- vx: if vy and vz are None, vx[0],vx[1],vx[2] defines the unit vector.
- vy,vz: if defined, vx,vy,vz define the unit vector
- lonlat: if True, angle are assumed in degree, and longitude, latitude
- flipconv is either 'astro' or 'geo'. None will be default.

#### Return:

- x, y: position in %s plane.

### healpy.projector.SphericalProj.xy2ang

SphericalProj.**xy2ang** (x, y=None, lonlat=False, direct=False)

From position in the projection plane to angular direction (%s).

#### Input:

- x : if y is None, x[0], x[1] define the position in %s plane.
- y : if defined, x,y define the position in projection plane.
- lonlat: if True, angle are assumed in degree, and longitude, latitude
- flipconv is either 'astro' or 'geo'. None will be default.

#### Return:

- theta, phi : angular direction.

### healpy.projector.SphericalProj.xy2ij

SphericalProj.**xy2ij** (x, y=None)

From position in the projection plane to image array index (%s).

#### Input:

- `x` : if `y` is `None`, `x[0]`, `x[1]` define the position in %s plane.
- `y` : if defined, `x,y` define the position in projection plane.
- `projinfo` : additional projection information.

**Return:**

- `i,j` : image array indices.

### healpy.projector.SphericalProj.xy2vec

SphericalProj.**xy2vec** (`x`, `y=None`, `direct=False`)

From position in the projection plane to unit vector direction (%s).

**Input:**

- `x` : if `y` is `None`, `x[0]`, `x[1]` define the position in %s plane.
- `y` : if defined, `x,y` define the position in projection plane.
- `lonlat`: if `True`, angle are assumed in degree, and longitude, latitude
- `flipconv` is either 'astro' or 'geo'. `None` will be default.

**Return:**

- `theta, phi` : angular direction.

### healpy.projector.GnomonicProj

**class** healpy.projector.**GnomonicProj** (`rot=None`, `coord=None`, `xsize=None`, `ysize=None`,  
`reso=None`, `**kws`)

This class provides class methods for Gnomonic projection.

#### Methods

<code>ang2xy(theta[, phi, lonlat, direct])</code>	From angular direction to position in the projection plane (Gnomonic).
<code>get_center([lonlat])</code>	Get the center of the projection.
<code>get_extent()</code>	
<code>get_fov()</code>	
<code>get_proj_plane_info()</code>	
<code>ij2xy([i, j])</code>	From image array indices to position in projection plane (Gnomonic).
<code>mkcoord(coord)</code>	
<code>projmap(map, vec2pix_func[, rot, coord])</code>	Create an array containing the projection of the map.
<code>set_flip(flipconv)</code>	<code>flipconv</code> is either 'astro' or 'geo'. <code>None</code> will be default.
<code>set_proj_plane_info([xsize, ysize, reso])</code>	
<code>vec2xy(vx[, vy, vz, direct])</code>	From angular direction to position in the projection plane (Gnomonic).
<code>xy2ang(x[, y, lonlat, direct])</code>	From position in the projection plane to angular direction (Gnomonic).
<code>xy2ij(x[, y])</code>	From position in the projection plane to image array index (Gnomonic).
<code>xy2vec(x[, y, direct])</code>	From position in the projection plane to unit vector direction (Gnomonic).

**healpy.projector.GnomonicProj.ang2xy**

`GnomonicProj.ang2xy(theta, phi=None, lonlat=False, direct=False)`

From angular direction to position in the projection plane (Gnomonic).

**Input:**

- theta: if phi is None, theta[0] contains theta, theta[1] contains phi
- phi : if phi is not None, theta, phi are direction
- lonlat: if True, angle are assumed in degree, and longitude, latitude
- flipconv is either 'astro' or 'geo'. None will be default.

**Return:**

- x, y: position in Gnomonic plane.

**healpy.projector.GnomonicProj.get\_center**

`GnomonicProj.get_center(lonlat=False)`

Get the center of the projection.

**Input:**

- lonlat [if True, will return longitude and latitude in degree,] otherwise, theta and phi in radian

**Return:**

- theta, phi or lonlat depending on lonlat keyword

**healpy.projector.GnomonicProj.get\_extent**

`GnomonicProj.get_extent()`

**healpy.projector.GnomonicProj.get\_fov**

`GnomonicProj.get_fov()`

**healpy.projector.GnomonicProj.get\_proj\_plane\_info**

`GnomonicProj.get_proj_plane_info()`

**healpy.projector.GnomonicProj.ij2xy**

`GnomonicProj.ij2xy(i=None, j=None)`

From image array indices to position in projection plane (Gnomonic).

**Input:**

- if i and j are None, generate arrays of i and j as input
- i : if j is None, i[0], j[1] define array indices in Gnomonic image.
- j : if defined, i, j define array indices in image.

- projinfo : additional projection information.

**Return:**

- x,y : position in projection plane.

**healpy.projector.GnomonicProj.mkcoord**

GnomonicProj.**mkcoord**(coord)

**healpy.projector.GnomonicProj.projmap**

GnomonicProj.**projmap**(map, vec2pix\_func, rot=None, coord=None)

Create an array containing the projection of the map.

**Input:**

- vec2pix\_func: a function taking theta,phi and returning pixel number
- **map: an array containing the spherical map to project**, the pixelisation is described by vec2pix\_func

**Return:**

- a 2D array with the projection of the map.

Note: the Projector must contain information on the array.

**healpy.projector.GnomonicProj.set\_flip**

GnomonicProj.**set\_flip**(flipconv)

flipconv is either 'astro' or 'geo'. None will be default.

With 'astro', east is toward left and west toward right. It is the opposite for 'geo'

**healpy.projector.GnomonicProj.set\_proj\_plane\_info**

GnomonicProj.**set\_proj\_plane\_info**(xsize=200, ysize=None, reso=1.5)

**healpy.projector.GnomonicProj.vec2xy**

GnomonicProj.**vec2xy**(vx, vy=None, vz=None, direct=False)

From angular direction to position in the projection plane (Gnomonic).

**Input:**

- theta: if phi is None, theta[0] contains theta, theta[1] contains phi
- phi : if phi is not None, theta,phi are direction
- lonlat: if True, angle are assumed in degree, and longitude, latitude
- flipconv is either 'astro' or 'geo'. None will be default.

**Return:**

- x, y: position in Gnomonic plane.



### healpy.projector.GnomonicProj.xy2ang

`GnomonicProj.xy2ang` (*x*, *y=None*, *lonlat=False*, *direct=False*)

From position in the projection plane to angular direction (Gnomonic).

**Input:**

- *x* : if *y* is None, *x*[0], *x*[1] define the position in Gnomonic plane.
- *y* : if defined, *x*,*y* define the position in projection plane.
- *lonlat*: if True, angle are assumed in degree, and longitude, latitude
- *flipconv* is either 'astro' or 'geo'. None will be default.

**Return:**

- *theta*, *phi* : angular direction.

### healpy.projector.GnomonicProj.xy2ij

`GnomonicProj.xy2ij` (*x*, *y=None*)

From position in the projection plane to image array index (Gnomonic).

**Input:**

- *x* : if *y* is None, *x*[0], *x*[1] define the position in Gnomonic plane.
- *y* : if defined, *x*,*y* define the position in projection plane.
- *projinfo* : additional projection information.

**Return:**

- *i*,*j* : image array indices.

### healpy.projector.GnomonicProj.xy2vec

`GnomonicProj.xy2vec` (*x*, *y=None*, *direct=False*)

From position in the projection plane to unit vector direction (Gnomonic).

**Input:**

- *x* : if *y* is None, *x*[0], *x*[1] define the position in Gnomonic plane.
- *y* : if defined, *x*,*y* define the position in projection plane.
- *lonlat*: if True, angle are assumed in degree, and longitude, latitude
- *flipconv* is either 'astro' or 'geo'. None will be default.

**Return:**

- *theta*, *phi* : angular direction.

### healpy.projector.MollweideProj

`class healpy.projector.MollweideProj` (*rot=None*, *coord=None*, *xsize=800*, *\*\*kws*)

This class provides class methods for Mollweide projection.

## Methods

<code>ang2xy(theta[, phi, lonlat, direct])</code>	From angular direction to position in the projection plane (Mollweide).
<code>get_center([lonlat])</code>	Get the center of the projection.
<code>get_extent()</code>	
<code>get_fov()</code>	Get the field of view in degree of the plane of projection
<code>get_proj_plane_info()</code>	
<code>ij2xy([i, j])</code>	From image array indices to position in projection plane (Mollweide).
<code>mkcoord(coord)</code>	
<code>projmap(map, vec2pix_func[, rot, coord])</code>	Create an array containing the projection of the map.
<code>set_flip(flipconv)</code>	flipconv is either 'astro' or 'geo'. None will be default.
<code>set_proj_plane_info(xsize)</code>	
<code>vec2xy(vx[, vy, vz, direct])</code>	From unit vector direction to position in the projection plane (Mollweide).
<code>xy2ang(x[, y, lonlat, direct])</code>	From position in the projection plane to angular direction (Mollweide).
<code>xy2ij(x[, y])</code>	From position in the projection plane to image array index (Mollweide).
<code>xy2vec(x[, y, direct])</code>	From position in the projection plane to unit vector direction (Mollweide).

### healpy.projector.MollweideProj.ang2xy

MollweideProj.**ang2xy** (*theta, phi=None, lonlat=False, direct=False*)

From angular direction to position in the projection plane (Mollweide).

#### Input:

- theta: if phi is None, theta[0] contains theta, theta[1] contains phi
- phi : if phi is not None, theta,phi are direction
- lonlat: if True, angle are assumed in degree, and longitude, latitude
- flipconv is either 'astro' or 'geo'. None will be default.

#### Return:

- x, y: position in Mollweide plane.

### healpy.projector.MollweideProj.get\_center

MollweideProj.**get\_center** (*lonlat=False*)

Get the center of the projection.

#### Input:

- **lonlat** [if True, will return longitude and latitude in degree,] otherwise, theta and phi in radian

#### Return:

- theta,phi or lonlat depending on lonlat keyword

### healpy.projector.MollweideProj.get\_extent

MollweideProj.**get\_extent** ()

### healpy.projector.MollweideProj.get\_fov

`MollweideProj.get_fov()`

Get the field of view in degree of the plane of projection

**Return:** fov: the diameter in radian of the field of view

### healpy.projector.MollweideProj.get\_proj\_plane\_info

`MollweideProj.get_proj_plane_info()`

### healpy.projector.MollweideProj.ij2xy

`MollweideProj.ij2xy(i=None, j=None)`

From image array indices to position in projection plane (Mollweide).

**Input:**

- if i and j are None, generate arrays of i and j as input
- i : if j is None, i[0], j[1] define array indices in Mollweide image.
- j : if defined, i,j define array indices in image.
- projinfo : additional projection information.

**Return:**

- x,y : position in projection plane.

### healpy.projector.MollweideProj.mkcoord

`MollweideProj.mkcoord(coord)`

### healpy.projector.MollweideProj.projmap

`MollweideProj.projmap(map, vec2pix_func, rot=None, coord=None)`

Create an array containing the projection of the map.

**Input:**

- vec2pix\_func: a function taking theta,phi and returning pixel number
- **map: an array containing the spherical map to project**, the pixelisation is described by vec2pix\_func

**Return:**

- a 2D array with the projection of the map.

Note: the Projector must contain information on the array.

### healpy.projector.MollweideProj.set\_flip

MollweideProj.**set\_flip** (*flipconv*)

flipconv is either 'astro' or 'geo'. None will be default.

With 'astro', east is toward left and west toward right. It is the opposite for 'geo'

### healpy.projector.MollweideProj.set\_proj\_plane\_info

MollweideProj.**set\_proj\_plane\_info** (*xsize*)

### healpy.projector.MollweideProj.vec2xy

MollweideProj.**vec2xy** (*vx*, *vy=None*, *vz=None*, *direct=False*)

From unit vector direction to position in the projection plane (Mollweide).

#### Input:

- vx: if vy and vz are None, vx[0],vx[1],vx[2] defines the unit vector.
- vy,vz: if defined, vx,vy,vz define the unit vector
- lonlat: if True, angle are assumed in degree, and longitude, latitude
- flipconv is either 'astro' or 'geo'. None will be default.

#### Return:

- x, y: position in Mollweide plane.

### healpy.projector.MollweideProj.xy2ang

MollweideProj.**xy2ang** (*x*, *y=None*, *lonlat=False*, *direct=False*)

From position in the projection plane to angular direction (Mollweide).

#### Input:

- x : if y is None, x[0], x[1] define the position in Mollweide plane.
- y : if defined, x,y define the position in projection plane.
- lonlat: if True, angle are assumed in degree, and longitude, latitude
- flipconv is either 'astro' or 'geo'. None will be default.

#### Return:

- theta, phi : angular direction.

### healpy.projector.MollweideProj.xy2ij

MollweideProj.**xy2ij** (*x*, *y=None*)

From position in the projection plane to image array index (Mollweide).

#### Input:

- x : if y is None, x[0], x[1] define the position in Mollweide plane.
- y : if defined, x,y define the position in projection plane.

- projinfo : additional projection information.

#### Return:

- i,j : image array indices.

### healpy.projector.MollweideProj.xy2vec

MollweideProj.**xy2vec**(x, y=None, direct=False)

From position in the projection plane to unit vector direction (Mollweide).

#### Input:

- x : if y is None, x[0], x[1] define the position in Mollweide plane.
- y : if defined, x,y define the position in projection plane.
- lonlat: if True, angle are assumed in degree, and longitude, latitude
- flipconv is either 'astro' or 'geo'. None will be default.

#### Return:

- theta, phi : angular direction.

### healpy.projector.CartesianProj

**class** healpy.projector.**CartesianProj**(rot=None, coord=None, xsize=800, ysize=None, lonra=None, latra=None, \*\*kws)

This class provides class methods for Cartesian projection.

#### Methods

<code>ang2xy(theta[, phi, lonlat, direct])</code>	From angular direction to position in the projection plane (Cartesian).
<code>get_center([lonlat])</code>	Get the center of the projection.
<code>get_extent()</code>	Get the extension of the projection plane.
<code>get_fov()</code>	
<code>get_proj_plane_info()</code>	
<code>ij2xy([i, j])</code>	From image array indices to position in projection plane (Cartesian).
<code>mkcoord(coord)</code>	
<code>projmap(map, vec2pix_func[, rot, coord])</code>	Create an array containing the projection of the map.
<code>set_flip(flipconv)</code>	flipconv is either 'astro' or 'geo'. None will be default.
<code>set_proj_plane_info(xsize, ysize, lonra, latra)</code>	
<code>vec2xy(vx[, vy, vz, direct])</code>	From unit vector direction to position in the projection plane (Cartesian).
<code>xy2ang(x[, y, lonlat, direct])</code>	From position in the projection plane to angular direction (Cartesian).
<code>xy2ij(x[, y])</code>	From position in the projection plane to image array index (Cartesian).
<code>xy2vec(x[, y, direct])</code>	From position in the projection plane to unit vector direction (Cartesian).

### healpy.projector.CartesianProj.ang2xy

CartesianProj.**ang2xy**(theta, phi=None, lonlat=False, direct=False)

From angular direction to position in the projection plane (Cartesian).

#### Input:

- theta: if phi is None, theta[0] contains theta, theta[1] contains phi
- phi : if phi is not None, theta,phi are direction
- lonlat: if True, angle are assumed in degree, and longitude, latitude
- flipconv is either 'astro' or 'geo'. None will be default.

**Return:**

- x, y: position in Cartesian plane.

**healpy.projector.CartesianProj.get\_center**

`CartesianProj.get_center(lonlat=False)`

Get the center of the projection.

**Input:**

- **lonlat** [if True, will return longitude and latitude in degree,] otherwise, theta and phi in radian

**Return:**

- theta,phi or lonlat depending on lonlat keyword

**healpy.projector.CartesianProj.get\_extent**

`CartesianProj.get_extent()`

Get the extension of the projection plane.

**Return:** extent = (left,right,bottom,top)

**healpy.projector.CartesianProj.get\_fov**

`CartesianProj.get_fov()`

**healpy.projector.CartesianProj.get\_proj\_plane\_info**

`CartesianProj.get_proj_plane_info()`

**healpy.projector.CartesianProj.ij2xy**

`CartesianProj.ij2xy(i=None,j=None)`

From image array indices to position in projection plane (Cartesian).

**Input:**

- if i and j are None, generate arrays of i and j as input
- i : if j is None, i[0], j[1] define array indices in Cartesian image.
- j : if defined, i,j define array indices in image.
- projinfo : additional projection information.

**Return:**

- x,y : position in projection plane.

**healpy.projector.CartesianProj.mkcoord**

`CartesianProj.mkcoord(coord)`

**healpy.projector.CartesianProj.projmap**

`CartesianProj.projmap(map, vec2pix_func, rot=None, coord=None)`

Create an array containing the projection of the map.

**Input:**

- `vec2pix_func`: a function taking `theta, phi` and returning pixel number
- **map**: an array containing the spherical map to project, the pixelisation is described by `vec2pix_func`

**Return:**

- a 2D array with the projection of the map.

Note: the Projector must contain information on the array.

**healpy.projector.CartesianProj.set\_flip**

`CartesianProj.set_flip(flipconv)`

`flipconv` is either 'astro' or 'geo'. None will be default.

With 'astro', east is toward left and west toward right. It is the opposite for 'geo'

**healpy.projector.CartesianProj.set\_proj\_plane\_info**

`CartesianProj.set_proj_plane_info(xsize, ysize, lonra, latra)`

**healpy.projector.CartesianProj.vec2xy**

`CartesianProj.vec2xy(vx, vy=None, vz=None, direct=False)`

From unit vector direction to position in the projection plane (Cartesian).

**Input:**

- `vx`: if `vy` and `vz` are None, `vx[0], vx[1], vx[2]` defines the unit vector.
- `vy, vz`: if defined, `vx, vy, vz` define the unit vector
- `lonlat`: if True, angle are assumed in degree, and longitude, latitude
- `flipconv` is either 'astro' or 'geo'. None will be default.

**Return:**

- `x, y`: position in Cartesian plane.

### healpy.projector.CartesianProj.xy2ang

`CartesianProj.xy2ang` (*x*, *y=None*, *lonlat=False*, *direct=False*)

From position in the projection plane to angular direction (Cartesian).

**Input:**

- *x* : if *y* is None, *x*[0], *x*[1] define the position in Cartesian plane.
- *y* : if defined, *x*,*y* define the position in projection plane.
- *lonlat*: if True, angle are assumed in degree, and longitude, latitude
- *flipconv* is either 'astro' or 'geo'. None will be default.

**Return:**

- *theta*, *phi* : angular direction.

### healpy.projector.CartesianProj.xy2ij

`CartesianProj.xy2ij` (*x*, *y=None*)

From position in the projection plane to image array index (Cartesian).

**Input:**

- *x* : if *y* is None, *x*[0], *x*[1] define the position in Cartesian plane.
- *y* : if defined, *x*,*y* define the position in projection plane.
- *projinfo* : additional projection information.

**Return:**

- *i*,*j* : image array indices.

### healpy.projector.CartesianProj.xy2vec

`CartesianProj.xy2vec` (*x*, *y=None*, *direct=False*)

From position in the projection plane to unit vector direction (Cartesian).

**Input:**

- *x* : if *y* is None, *x*[0], *x*[1] define the position in Cartesian plane.
- *y* : if defined, *x*,*y* define the position in projection plane.
- *lonlat*: if True, angle are assumed in degree, and longitude, latitude
- *flipconv* is either 'astro' or 'geo'. None will be default.

**Return:**

- *theta*, *phi* : angular direction.

## 3.7 projaxes – Axes for projection

### 3.7.1 Basic classes



<code>SphericalProjAxes(ProjClass, *args, **kws)</code>	Define a special Axes to take care of spherical projection.
<code>GnomonicAxes(*args, **kws)</code>	Define a gnomonic Axes to handle gnomonic projection.
<code>HpxGnomonicAxes(*args, **kws)</code>	
<code>MollweideAxes(*args, **kws)</code>	Define a mollweide Axes to handle mollweide projection.
<code>HpxMollweideAxes(*args, **kws)</code>	
<code>CartesianAxes(*args, **kws)</code>	Define a cylindrical Axes to handle cylindrical projection.
<code>HpxCartesianAxes(*args, **kws)</code>	

## healpy.projaxes.SphericalProjAxes

**class** `healpy.projaxes.SphericalProjAxes` (*ProjClass, \*args, \*\*kws*)

Define a special Axes to take care of spherical projection.

**Parameters** **projection** : a SphericalProj class or a class derived from it.

type of projection

**rot** : list or string

define rotation. See rotator.

**coord** : list or string

define coordinate system. See rotator.

**coordprec** : number of digit after floating point for coordinates display.

**format** : format string for value display.

### Notes

Other keywords from Axes (see Axes).

### Methods

<code>delgraticules()</code>	Delete all graticules previously created on the Axes.
<code>format_coord(x, y)</code>	Format the coordinate for display in status bar.
<code>get_lonlat(x, y)</code>	Get the coordinate in the coord system of the image, in lon/lat in deg.
<code>get_meridian_interval(vx[, vy, vz])</code>	Get the min and max value of phi of the meridians to cover the field of view.
<code>get_parallel_interval(vx[, vy, vz])</code>	Get the min and max value of theta of the parallel to cover the field of view.
<code>get_value(x, y)</code>	Get the value of the map at position x,y
<code>graticule([dpar, dmer, coord, local, verbose])</code>	Draw a graticule.
<code>projmap(map, vec2pix_func[, vmin, vmax, ...])</code>	Project a map on the SphericalProjAxes.
<code>projplot(*args, **kws)</code>	projplot is a wrapper around <code>matplotlib.Axes.plot()</code> to take into account the projection.
<code>projscatter(theta[, phi])</code>	Projscatter is a wrapper around <code>matplotlib.Axes.scatter()</code> to take into account the projection.
<code>projtext(theta, phi, s, **kws)</code>	Projtext is a wrapper around <code>matplotlib.Axes.text()</code> to take into account the projection.
<code>set_coordprec(n)</code>	Set the number of digits after floating point for coord display.
<code>set_format(f)</code>	Set the format string for value display

### healpy.projaxes.SphericalProjAxes.delgraticules

`SphericalProjAxes.delgraticules()`  
Delete all graticules previously created on the Axes.

### healpy.projaxes.SphericalProjAxes.format\_coord

`SphericalProjAxes.format_coord(x, y)`  
Format the coordinate for display in status bar. Take projection into account.

### healpy.projaxes.SphericalProjAxes.get\_lonlat

`SphericalProjAxes.get_lonlat(x, y)`  
Get the coordinate in the coord system of the image, in lon/lat in deg.

### healpy.projaxes.SphericalProjAxes.get\_meridian\_interval

`SphericalProjAxes.get_meridian_interval(vx, vy=None, vz=None)`  
Get the min and max value of phi of the meridians to cover the field of view.

**Input:**

- the normalized vector of the direction of the center of the projection, in the reference frame of the graticule.

**Return:**

- **vmin,vmax** [the interval of phi for the] meridians crossing the field of view.

### healpy.projaxes.SphericalProjAxes.get\_parallel\_interval

`SphericalProjAxes.get_parallel_interval(vx, vy=None, vz=None)`  
Get the min and max value of theta of the parallel to cover the field of view.

**Input:**

- the normalized vector of the direction of the center of the projection, in the reference frame of the graticule.

**Return:**

- **vmin,vmax** [between 0 and pi, vmin<vmax, the interval of theta] for the parallels crossing the field of view

### healpy.projaxes.SphericalProjAxes.get\_value

`SphericalProjAxes.get_value(x, y)`  
Get the value of the map at position x,y

**healpy.projaxes.SphericalProjAxes.graticule**

`SphericalProjAxes.graticule` (*dpar=None, dmer=None, coord=None, local=None, verbose=True, \*\*kwds*)

Draw a graticule.

**Input:**

- **dpar**: angular separation between parallels in degree
- **dmer**: angular separation between meridians in degree
- **coord**: coordinate system of the graticule ('G', 'E' or 'C')
- **local**: if True, no rotation performed at all

**healpy.projaxes.SphericalProjAxes.projmap**

`SphericalProjAxes.projmap` (*map, vec2pix\_func, vmin=None, vmax=None, badval=<class 'Mock'>, cmap=None, norm=None, rot=None, coord=None, \*\*kwds*)

Project a map on the SphericalProjAxes.

**Parameters** **map** : array-like

The map to project.

**vec2pix\_func** : function

The function describing the pixelisation.

**vmin, vmax** : float, scalars

min and max value to use instead of min max of the map

**badval** : float

The value of the bad pixels

**cmap** : a color map

The colormap to use (see matplotlib.cm)

**rot** : sequence

In the form (lon, lat, psi) (unit: degree): the center of the map is at (lon, lat) and rotated by angle psi around that direction.

**coord** : {'G', 'E', 'C', None}

The coordinate system of the map ('G', 'E' or 'C'), rotate the map if different from the axes coord syst.

**Notes**

Other keywords are transmitted to `matplotlib.Axes.imshow()`

### healpy.projaxes.SphericalProjAxes.projplot

SphericalProjAxes.**projplot** (\*args, \*\*kws)

projplot is a wrapper around `matplotlib.Axes.plot()` to take into account the spherical projection.

You can call this function as:

```
projplot(theta, phi)           # plot a line going through points at coord (theta, phi)
projplot(theta, phi, 'bo')     # plot 'o' in blue at coord (theta, phi)
projplot(thetaphi)            # plot a line going through points at coord (thetaphi[0], thetaphi[1])
projplot(thetaphi, 'bx')       # idem but with blue 'x'
```

**Parameters** **theta, phi** : float, array-like

Coordinates of point to plot. Can be put into one 2-d array, first line is then *theta* and second line is *phi*. See *lonlat* parameter for unit.

**fmt** : str

A format string (see `matplotlib.Axes.plot()` for details)

**lonlat** : bool, optional

If True, theta and phi are interpreted as longitude and latitude in degree, otherwise, as colatitude and longitude in radian

**coord** : {'E', 'G', 'C', None}

The coordinate system of the points, only used if the coordinate coordinate system of the Axes has been defined and in this case, a rotation is performed

**rot** : None or sequence

rotation to be applied =(lon, lat, psi) : lon, lat will be position of the new Z axis, and psi is rotation around this axis, all in degree. if None, no rotation is performed

**direct** : bool

if True, the rotation to center the projection is not taken into account

**See Also:**

`projscatter`, `projtext`

#### Notes

Other keywords are passed to `matplotlib.Axes.plot()`.

### healpy.projaxes.SphericalProjAxes.projscatter

SphericalProjAxes.**projscatter** (theta, phi=None, \*args, \*\*kws)

Projscatter is a wrapper around `matplotlib.Axes.scatter()` to take into account the spherical projection.

You can call this function as:

```
projscatter(theta, phi)       # plot points at coord (theta, phi)
projplot(thetaphi)           # plot points at coord (thetaphi[0], thetaphi[1])
```

**Parameters** *theta, phi* : float, array-like

Coordinates of point to plot. Can be put into one 2-d array, first line is then *theta* and second line is *phi*. See *lonlat* parameter for unit.

**lonlat** : bool, optional

If True, *theta* and *phi* are interpreted as longitude and latitude in degree, otherwise, as colatitude and longitude in radian

**coord** : {'E', 'G', 'C', None}, optional

The coordinate system of the points, only used if the coordinate coordinate system of the axes has been defined and in this case, a rotation is performed

**rot** : None or sequence, optional

rotation to be applied =(lon, lat, psi) : lon, lat will be position of the new Z axis, and psi is rotation around this axis, all in degree. if None, no rotation is performed

**direct** : bool, optional

if True, the rotation to center the projection is not taken into account

**See Also:**

`projplot`, `projtext`

## Notes

Other keywords are passed to `matplotlib.Axes.plot()`.

## healpy.projaxes.SphericalProjAxes.projtext

`SphericalProjAxes.projtext(theta, phi, s, **kws)`

`Projtext` is a wrapper around `matplotlib.Axes.text()` to take into account the spherical projection.

**Parameters** *theta, phi* : float, array-like

Coordinates of point to plot. Can be put into one 2-d array, first line is then *theta* and second line is *phi*. See *lonlat* parameter for unit.

**text** : str

The text to be displayed.

**lonlat** : bool, optional

If True, *theta* and *phi* are interpreted as longitude and latitude in degree, otherwise, as colatitude and longitude in radian

**coord** : {'E', 'G', 'C', None}, optional

The coordinate system of the points, only used if the coordinate coordinate system of the axes has been defined and in this case, a rotation is performed

**rot** : None or sequence, optional

rotation to be applied =(lon, lat, psi) : lon, lat will be position of the new Z axis, and psi is rotation around this axis, all in degree. if None, no rotation is performed

**direct** : bool, optional

if True, the rotation to center the projection is not taken into account

#### See Also:

`projplot`, `projscatter`

#### Notes

Other keywords are passed to `matplotlib.Axes.text()`.

### healpy.projaxes.SphericalProjAxes.set\_coordprec

`SphericalProjAxes.set_coordprec(n)`

Set the number of digits after floating point for coord display.

### healpy.projaxes.SphericalProjAxes.set\_format

`SphericalProjAxes.set_format(f)`

Set the format string for value display

## healpy.projaxes.GnomonicAxes

**class** `healpy.projaxes.GnomonicAxes(*args, **kws)`

Define a gnomonic Axes to handle gnomonic projection.

#### Input:

- `rot=`, `coord=` : define rotation and coordinate system. See `rotator`.
- `coordprec=` : number of digit after floating point for coordinates display.
- `format=` : format string for value display.

Other keywords from Axes (see Axes).

#### Methods

<code>delgraticules()</code>	Delete all graticules previously created on the Axes.
<code>format_coord(x, y)</code>	Format the coordinate for display in status bar.
<code>get_lonlat(x, y)</code>	Get the coordinate in the coord system of the image, in lon/lat in deg.
<code>get_meridian_interval(vx[, vy, vz])</code>	Get the min and max value of phi of the meridians to cover the field of view.
<code>get_parallel_interval(vx[, vy, vz])</code>	Get the min and max value of theta of the parallel to cover the field of view.
<code>get_value(x, y)</code>	Get the value of the map at position x,y
<code>graticule([dpar, dmer, coord, local, verbose])</code>	Draw a graticule.
<code>projmap(map, vec2pix_func[, xsize, ysize, reso])</code>	
<code>projplot(*args, **kws)</code>	<code>projplot</code> is a wrapper around <code>matplotlib.Axes.plot()</code> to take into account the projection.
<code>projscatter(theta[, phi])</code>	<code>Projscatter</code> is a wrapper around <code>matplotlib.Axes.scatter()</code> to take into account the projection.
<code>projtext(theta, phi, s, **kws)</code>	<code>Projtext</code> is a wrapper around <code>matplotlib.Axes.text()</code> to take into account the projection.
<code>set_coordprec(n)</code>	Set the number of digits after floating point for coord display.
<code>set_format(f)</code>	Set the format string for value display

### healpy.projaxes.GnomonicAxes.delgraticules

`GnomonicAxes.delgraticules()`  
Delete all graticules previously created on the Axes.

### healpy.projaxes.GnomonicAxes.format\_coord

`GnomonicAxes.format_coord(x, y)`  
Format the coordinate for display in status bar. Take projection into account.

### healpy.projaxes.GnomonicAxes.get\_lonlat

`GnomonicAxes.get_lonlat(x, y)`  
Get the coordinate in the coord system of the image, in lon/lat in deg.

### healpy.projaxes.GnomonicAxes.get\_meridian\_interval

`GnomonicAxes.get_meridian_interval(vx, vy=None, vz=None)`  
Get the min and max value of phi of the meridians to cover the field of view.

**Input:**

- the normalized vector of the direction of the center of the projection, in the reference frame of the graticule.

**Return:**

- **vmin,vmax** [the interval of phi for the] meridians crossing the field of view.

### healpy.projaxes.GnomonicAxes.get\_parallel\_interval

`GnomonicAxes.get_parallel_interval(vx, vy=None, vz=None)`  
Get the min and max value of theta of the parallel to cover the field of view.

**Input:**

- the normalized vector of the direction of the center of the projection, in the reference frame of the graticule.

**Return:**

- **vmin,vmax** [between 0 and pi, vmin<vmax, the interval of theta] for the parallels crossing the field of view

### healpy.projaxes.GnomonicAxes.get\_value

`GnomonicAxes.get_value(x, y)`  
Get the value of the map at position x,y

### healpy.projaxes.GnomonicAxes.graticule

GnomonicAxes.**graticule** (*dpar=None, dmer=None, coord=None, local=None, verbose=True, \*\*kws*)

Draw a graticule.

#### Input:

- *dpar*: angular separation between parallels in degree
- *dmer*: angular separation between meridians in degree
- *coord*: coordinate system of the graticule ('G', 'E' or 'C')
- *local*: if True, no rotation performed at all

### healpy.projaxes.GnomonicAxes.projmap

GnomonicAxes.**projmap** (*map, vec2pix\_func, xsize=200, ysize=None, reso=1.5, \*\*kws*)

### healpy.projaxes.GnomonicAxes.projplot

GnomonicAxes.**projplot** (*\*args, \*\*kws*)

`projplot` is a wrapper around `matplotlib.Axes.plot()` to take into account the spherical projection.

You can call this function as:

```
projplot(theta, phi)           # plot a line going through points at coord (theta, phi)
projplot(theta, phi, 'bo')    # plot 'o' in blue at coord (theta, phi)
projplot(thetaphi)           # plot a line going through points at coord (thetaphi[0], thetaphi[1])
projplot(thetaphi, 'bx')      # idem but with blue 'x'
```

**Parameters** *theta, phi* : float, array-like

Coordinates of point to plot. Can be put into one 2-d array, first line is then *theta* and second line is *phi*. See *lonlat* parameter for unit.

**fmt** : str

A format string (see `matplotlib.Axes.plot()` for details)

**lonlat** : bool, optional

If True, *theta* and *phi* are interpreted as longitude and latitude in degree, otherwise, as colatitude and longitude in radian

**coord** : {'E', 'G', 'C', None}

The coordinate system of the points, only used if the coordinate coordinate system of the Axes has been defined and in this case, a rotation is performed

**rot** : None or sequence

rotation to be applied =(lon, lat, psi) : lon, lat will be position of the new Z axis, and psi is rotation around this axis, all in degree. if None, no rotation is performed

**direct** : bool

if True, the rotation to center the projection is not taken into account



**See Also:**

`projscatter`, `projtext`

**Notes**

Other keywords are passed to `matplotlib.Axes.plot()`.

**healpy.projaxes.GnomonicAxes.projscatter**

`GnomonicAxes.projscatter` (*theta*, *phi*=None, \*args, \*\*kws)

`Projscatter` is a wrapper around `matplotlib.Axes.scatter()` to take into account the spherical projection.

You can call this function as:

```
projscatter(theta, phi)      # plot points at coord (theta, phi)
projplot(thetaphi)          # plot points at coord (thetaphi[0], thetaphi[1])
```

**Parameters** *theta*, *phi* : float, array-like

Coordinates of point to plot. Can be put into one 2-d array, first line is then *theta* and second line is *phi*. See *lonlat* parameter for unit.

**lonlat** : bool, optional

If True, *theta* and *phi* are interpreted as longitude and latitude in degree, otherwise, as colatitude and longitude in radian

**coord** : {'E', 'G', 'C', None}, optional

The coordinate system of the points, only used if the coordinate coordinate system of the axes has been defined and in this case, a rotation is performed

**rot** : None or sequence, optional

rotation to be applied =(lon, lat, psi) : lon, lat will be position of the new Z axis, and psi is rotation around this axis, all in degree. if None, no rotation is performed

**direct** : bool, optional

if True, the rotation to center the projection is not taken into account

**See Also:**

`projplot`, `projtext`

**Notes**

Other keywords are passed to `matplotlib.Axes.plot()`.

**healpy.projaxes.GnomonicAxes.projtext**

`GnomonicAxes.projtext` (*theta*, *phi*, *s*, \*\*kws)

`Projtext` is a wrapper around `matplotlib.Axes.text()` to take into account the spherical projection.

**Parameters** *theta*, *phi* : float, array-like

Coordinates of point to plot. Can be put into one 2-d array, first line is then *theta* and second line is *phi*. See *lonlat* parameter for unit.

**text** : str

The text to be displayed.

**lonlat** : bool, optional

If True, theta and phi are interpreted as longitude and latitude in degree, otherwise, as colatitude and longitude in radian

**coord** : {'E', 'G', 'C', None}, optional

The coordinate system of the points, only used if the coordinate coordinate system of the axes has been defined and in this case, a rotation is performed

**rot** : None or sequence, optional

rotation to be applied =(lon, lat, psi) : lon, lat will be position of the new Z axis, and psi is rotation around this axis, all in degree. if None, no rotation is performed

**direct** : bool, optional

if True, the rotation to center the projection is not taken into account

#### See Also:

`projplot`, `projscatter`

#### Notes

Other keywords are passed to `matplotlib.Axes.text()`.

#### `healpy.projaxes.GnomonicAxes.set_coordprec`

`GnomonicAxes.set_coordprec(n)`

Set the number of digits after floating point for coord display.

#### `healpy.projaxes.GnomonicAxes.set_format`

`GnomonicAxes.set_format(f)`

Set the format string for value display

#### `healpy.projaxes.HpxGnomonicAxes`

`class healpy.projaxes.HpxGnomonicAxes(*args, **kws)`

#### Methods

<code>delgraticules()</code>	Delete all graticules previously created on the Axes.
<code>format_coord(x, y)</code>	Format the coordinate for display in status bar.
<code>get_lonlat(x, y)</code>	Get the coordinate in the coord system of the image, in lon/lat in deg.
<code>get_meridian_interval(vx[, vy, vz])</code>	Get the min and max value of phi of the meridians to cover the field of view.
Continued on next page	

Table 3.30 – continued from previous page

<code>get_parallel_interval(vx[, vy, vz])</code>	Get the min and max value of theta of the parallel to cover the field of view.
<code>get_value(x, y)</code>	Get the value of the map at position x,y
<code>graticule([dpar, dmer, coord, local, verbose])</code>	Draw a graticule.
<code>projmap(map[, nest])</code>	
<code>projplot(*args, **kwds)</code>	<code>projplot</code> is a wrapper around <code>matplotlib.Axes.plot()</code> to take into account
<code>projscatter(theta[, phi])</code>	<code>Projscatter</code> is a wrapper around <code>matplotlib.Axes.scatter()</code> to take into
<code>projtext(theta, phi, s, **kwds)</code>	<code>Projtext</code> is a wrapper around <code>matplotlib.Axes.text()</code> to take into account
<code>set_coordprec(n)</code>	Set the number of digits after floating point for coord display.
<code>set_format(f)</code>	Set the format string for value display

**healpy.projaxes.HpxGnomonicAxes.delgraticules**

`HpxGnomonicAxes.delgraticules()`  
Delete all graticules previously created on the Axes.

**healpy.projaxes.HpxGnomonicAxes.format\_coord**

`HpxGnomonicAxes.format_coord(x, y)`  
Format the coordinate for display in status bar. Take projection into account.

**healpy.projaxes.HpxGnomonicAxes.get\_lonlat**

`HpxGnomonicAxes.get_lonlat(x, y)`  
Get the coordinate in the coord system of the image, in lon/lat in deg.

**healpy.projaxes.HpxGnomonicAxes.get\_meridian\_interval**

`HpxGnomonicAxes.get_meridian_interval(vx, vy=None, vz=None)`  
Get the min and max value of phi of the meridians to cover the field of view.

**Input:**

- the normalized vector of the direction of the center of the projection, in the reference frame of the graticule.

**Return:**

- **vmin,vmax** [the interval of phi for the] meridians crossing the field of view.

**healpy.projaxes.HpxGnomonicAxes.get\_parallel\_interval**

`HpxGnomonicAxes.get_parallel_interval(vx, vy=None, vz=None)`  
Get the min and max value of theta of the parallel to cover the field of view.

**Input:**

- the normalized vector of the direction of the center of the projection, in the reference frame of the graticule.

**Return:**

- **vmin,vmax** [between 0 and pi, vmin<vmax, the interval of theta] for the parallels crossing the field of view

### healpy.projaxes.HpxGnomonicAxes.get\_value

HpxGnomonicAxes.**get\_value**(*x*, *y*)  
 Get the value of the map at position *x*,*y*

### healpy.projaxes.HpxGnomonicAxes.graticule

HpxGnomonicAxes.**graticule**(*dpar=None*, *dmer=None*, *coord=None*, *local=None*, *verbose=True*, *\*\*kws*)  
 Draw a graticule.

#### Input:

- *dpar*: angular separation between parallels in degree
- *dmer*: angular separation between meridians in degree
- *coord*: coordinate system of the graticule ('G', 'E' or 'C')
- *local*: if True, no rotation performed at all

### healpy.projaxes.HpxGnomonicAxes.projmap

HpxGnomonicAxes.**projmap**(*map*, *nest=False*, *\*\*kws*)

### healpy.projaxes.HpxGnomonicAxes.projplot

HpxGnomonicAxes.**projplot**(*\*args*, *\*\*kws*)  
 projplot is a wrapper around matplotlib.Axes.plot() to take into account the spherical projection.

You can call this function as:

```
projplot(theta, phi)           # plot a line going through points at coord (theta, phi)
projplot(theta, phi, 'bo')     # plot 'o' in blue at coord (theta, phi)
projplot(thetaphi)             # plot a line going through points at coord (thetaphi[0], thetaphi[1])
projplot(thetaphi, 'bx')       # idem but with blue 'x'
```

**Parameters** *theta, phi* : float, array-like

Coordinates of point to plot. Can be put into one 2-d array, first line is then *theta* and second line is *phi*. See *lonlat* parameter for unit.

**fmt** : str

A format string (see matplotlib.Axes.plot() for details)

**lonlat** : bool, optional

If True, *theta* and *phi* are interpreted as longitude and latitude in degree, otherwise, as colatitude and longitude in radian

**coord** : {'E', 'G', 'C', None}

The coordinate system of the points, only used if the coordinate coordinate system of the Axes has been defined and in this case, a rotation is performed

**rot** : None or sequence

rotation to be applied =(lon, lat, psi) : lon, lat will be position of the new Z axis, and psi is rotation around this axis, all in degree. if None, no rotation is performed

**direct** : bool

if True, the rotation to center the projection is not taken into account

**See Also:**

`projscatter`, `projtext`

**Notes**

Other keywords are passed to `matplotlib.Axes.plot()`.

### healpy.projaxes.HpxGnomonicAxes.projscatter

HpxGnomonicAxes.**projscatter** (*theta*, *phi*=None, \*args, \*\*kws)

Projscatter is a wrapper around `matplotlib.Axes.scatter()` to take into account the spherical projection.

You can call this function as:

```
projscatter(theta, phi)      # plot points at coord (theta, phi)
projplot(thetaphi)          # plot points at coord (thetaphi[0], thetaphi[1])
```

**Parameters** **theta, phi** : float, array-like

Coordinates of point to plot. Can be put into one 2-d array, first line is then *theta* and second line is *phi*. See *lonlat* parameter for unit.

**lonlat** : bool, optional

If True, theta and phi are interpreted as longitude and latitude in degree, otherwise, as colatitude and longitude in radian

**coord** : {'E', 'G', 'C', None}, optional

The coordinate system of the points, only used if the coordinate coordinate system of the axes has been defined and in this case, a rotation is performed

**rot** : None or sequence, optional

rotation to be applied =(lon, lat, psi) : lon, lat will be position of the new Z axis, and psi is rotation around this axis, all in degree. if None, no rotation is performed

**direct** : bool, optional

if True, the rotation to center the projection is not taken into account

**See Also:**

`projplot`, `projtext`

**Notes**

Other keywords are passed to `matplotlib.Axes.plot()`.

### healpy.projaxes.HpxGnomonicAxes.projtext

HpxGnomonicAxes.**projtext** (*theta*, *phi*, *s*, **\*\*kwds**)

Projtext is a wrapper around `matplotlib.Axes.text()` to take into account the spherical projection.

**Parameters** *theta*, *phi* : float, array-like

Coordinates of point to plot. Can be put into one 2-d array, first line is then *theta* and second line is *phi*. See *lonlat* parameter for unit.

**text** : str

The text to be displayed.

**lonlat** : bool, optional

If True, theta and phi are interpreted as longitude and latitude in degree, otherwise, as colatitude and longitude in radian

**coord** : { 'E', 'G', 'C', None }, optional

The coordinate system of the points, only used if the coordinate coordinate system of the axes has been defined and in this case, a rotation is performed

**rot** : None or sequence, optional

rotation to be applied =(lon, lat, psi) : lon, lat will be position of the new Z axis, and psi is rotation around this axis, all in degree. if None, no rotation is performed

**direct** : bool, optional

if True, the rotation to center the projection is not taken into account

**See Also:**

`projplot`, `projscatter`

### Notes

Other keywords are passed to `matplotlib.Axes.text()`.

### healpy.projaxes.HpxGnomonicAxes.set\_coordprec

HpxGnomonicAxes.**set\_coordprec** (*n*)

Set the number of digits after floating point for coord display.

### healpy.projaxes.HpxGnomonicAxes.set\_format

HpxGnomonicAxes.**set\_format** (*f*)

Set the format string for value display

### healpy.projaxes.MollweideAxes

**class** healpy.projaxes.**MollweideAxes** (*\*args*, **\*\*kwds**)

Define a mollweide Axes to handle mollweide projection.

**Input:**

- `rot=`, `coord=` : define rotation and coordinate system. See `rotator`.
- `coordprec=` : number of digit after floating point for coordinates display.
- `format=` : format string for value display.

Other keywords from `Axes` (see `Axes`).

## Methods

<code>delgraticules()</code>	Delete all graticules previously created on the <code>Axes</code> .
<code>format_coord(x, y)</code>	Format the coordinate for display in status bar.
<code>get_lonlat(x, y)</code>	Get the coordinate in the <code>coord</code> system of the image, in lon/lat in deg.
<code>get_meridian_interval(vx[, vy, vz])</code>	Get the min and max value of <code>phi</code> of the meridians to cover the field of view.
<code>get_parallel_interval(vx[, vy, vz])</code>	Get the min and max value of <code>theta</code> of the parallel to cover the field of view.
<code>get_value(x, y)</code>	Get the value of the map at position <code>x,y</code>
<code>graticule([dpar, dmer, coord, local, verbose])</code>	Draw a graticule.
<code>projmap(map, vec2pix_func[, xsize])</code>	
<code>projplot(*args, **kwds)</code>	<code>projplot</code> is a wrapper around <code>matplotlib.Axes.plot()</code> to take into account projection.
<code>projscatter(theta[, phi])</code>	<code>Projscatter</code> is a wrapper around <code>matplotlib.Axes.scatter()</code> to take into account projection.
<code>projtext(theta, phi, s, **kwds)</code>	<code>Projtext</code> is a wrapper around <code>matplotlib.Axes.text()</code> to take into account projection.
<code>set_coordprec(n)</code>	Set the number of digits after floating point for coord display.
<code>set_format(f)</code>	Set the format string for value display

## healpy.projaxes.MollweideAxes.delgraticules

`MollweideAxes.delgraticules()`  
Delete all graticules previously created on the `Axes`.

## healpy.projaxes.MollweideAxes.format\_coord

`MollweideAxes.format_coord(x, y)`  
Format the coordinate for display in status bar. Take projection into account.

## healpy.projaxes.MollweideAxes.get\_lonlat

`MollweideAxes.get_lonlat(x, y)`  
Get the coordinate in the `coord` system of the image, in lon/lat in deg.

## healpy.projaxes.MollweideAxes.get\_meridian\_interval

`MollweideAxes.get_meridian_interval(vx, vy=None, vz=None)`  
Get the min and max value of `phi` of the meridians to cover the field of view.

### Input:

- the normalized vector of the direction of the center of the projection, in the reference frame of the graticule.

### Return:

- `vmin,vmax` [the interval of `phi` for the] meridians crossing the field of view.

### healpy.projaxes.MollweideAxes.get\_parallel\_interval

MollweideAxes.get\_parallel\_interval (vx, vy=None, vz=None)

Get the min and max value of theta of the parallel to cover the field of view.

#### Input:

- the normalized vector of the direction of the center of the projection, in the reference frame of the graticule.

#### Return:

- **vmin,vmax** [between 0 and pi, vmin<vmax, the interval of theta] for the parallels crossing the field of view

### healpy.projaxes.MollweideAxes.get\_value

MollweideAxes.get\_value (x, y)

Get the value of the map at position x,y

### healpy.projaxes.MollweideAxes.graticule

MollweideAxes.graticule (dpar=None, dmer=None, coord=None, local=None, verbose=True, \*\*kws)

Draw a graticule.

#### Input:

- dpar: angular separation between parallels in degree
- dmer: angular separation between meridians in degree
- coord: coordinate system of the graticule ('G', 'E' or 'C')
- local: if True, no rotation performed at all

### healpy.projaxes.MollweideAxes.projmap

MollweideAxes.projmap (map, vec2pix\_func, xsize=800, \*\*kws)

### healpy.projaxes.MollweideAxes.projplot

MollweideAxes.projplot (\*args, \*\*kws)

projplot is a wrapper around matplotlib.Axes.plot () to take into account the spherical projection.

You can call this function as:

```
projplot(theta, phi)           # plot a line going through points at coord (theta, phi)
projplot(theta, phi, 'bo')     # plot 'o' in blue at coord (theta, phi)
projplot(thetaphi)             # plot a line going through points at coord (thetaphi[0], thetaphi[1])
projplot(thetaphi, 'bx')       # idem but with blue 'x'
```

**Parameters** **theta, phi** : float, array-like

Coordinates of point to plot. Can be put into one 2-d array, first line is then *theta* and second line is *phi*. See *lonlat* parameter for unit.



**fmt** : strA format string (see `matplotlib.Axes.plot()` for details)**lonlat** : bool, optional

If True, theta and phi are interpreted as longitude and latitude in degree, otherwise, as colatitude and longitude in radian

**coord** : {'E', 'G', 'C', None}

The coordinate system of the points, only used if the coordinate coordinate system of the Axes has been defined and in this case, a rotation is performed

**rot** : None or sequencerotation to be applied  $=(\text{lon}, \text{lat}, \text{psi})$  : lon, lat will be position of the new Z axis, and psi is rotation around this axis, all in degree. if None, no rotation is performed**direct** : bool

if True, the rotation to center the projection is not taken into account

**See Also:**`projscatter`, `projtext`**Notes**Other keywords are passed to `matplotlib.Axes.plot()`.**healpy.projaxes.MollweideAxes.projscatter**`MollweideAxes.projscatter` (*theta*, *phi*=None, \*args, \*\*kwargs)Projscatter is a wrapper around `matplotlib.Axes.scatter()` to take into account the spherical projection.

You can call this function as:

```
projscatter(theta, phi)      # plot points at coord (theta, phi)
projplot(thetaphi)          # plot points at coord (thetaphi[0], thetaphi[1])
```

**Parameters** **theta, phi** : float, array-likeCoordinates of point to plot. Can be put into one 2-d array, first line is then *theta* and second line is *phi*. See *lonlat* parameter for unit.**lonlat** : bool, optional

If True, theta and phi are interpreted as longitude and latitude in degree, otherwise, as colatitude and longitude in radian

**coord** : {'E', 'G', 'C', None}, optional

The coordinate system of the points, only used if the coordinate coordinate system of the axes has been defined and in this case, a rotation is performed

**rot** : None or sequence, optionalrotation to be applied  $=(\text{lon}, \text{lat}, \text{psi})$  : lon, lat will be position of the new Z axis, and psi is rotation around this axis, all in degree. if None, no rotation is performed

**direct** : bool, optional

if True, the rotation to center the projection is not taken into account

**See Also:**

`projplot`, `projtext`

**Notes**

Other keywords are passed to `matplotlib.Axes.plot()`.

### `healpy.projaxes.MollweideAxes.projtext`

`MollweideAxes.projtext` (*theta*, *phi*, *s*, **\*\*kws**)

`Projtext` is a wrapper around `matplotlib.Axes.text()` to take into account the spherical projection.

**Parameters** **theta**, **phi** : float, array-like

Coordinates of point to plot. Can be put into one 2-d array, first line is then *theta* and second line is *phi*. See *lonlat* parameter for unit.

**text** : str

The text to be displayed.

**lonlat** : bool, optional

If True, *theta* and *phi* are interpreted as longitude and latitude in degree, otherwise, as colatitude and longitude in radian

**coord** : { 'E', 'G', 'C', None }, optional

The coordinate system of the points, only used if the coordinate coordinate system of the axes has been defined and in this case, a rotation is performed

**rot** : None or sequence, optional

rotation to be applied  $=(\text{lon}, \text{lat}, \text{psi})$  : *lon*, *lat* will be position of the new Z axis, and *psi* is rotation around this axis, all in degree. if None, no rotation is performed

**direct** : bool, optional

if True, the rotation to center the projection is not taken into account

**See Also:**

`projplot`, `projscatter`

**Notes**

Other keywords are passed to `matplotlib.Axes.text()`.

### `healpy.projaxes.MollweideAxes.set_coordprec`

`MollweideAxes.set_coordprec` (*n*)

Set the number of digits after floating point for coord display.

### healpy.projaxes.MollweideAxes.set\_format

MollweideAxes.**set\_format** (*f*)  
Set the format string for value display

### healpy.projaxes.HpxMollweideAxes

class healpy.projaxes.**HpxMollweideAxes** (*\*args, \*\*kws*)

#### Methods

<code>delgraticules()</code>	Delete all graticules previously created on the Axes.
<code>format_coord(x, y)</code>	Format the coordinate for display in status bar.
<code>get_lonlat(x, y)</code>	Get the coordinate in the coord system of the image, in lon/lat in deg.
<code>get_meridian_interval(vx[, vy, vz])</code>	Get the min and max value of phi of the meridians to cover the field of view.
<code>get_parallel_interval(vx[, vy, vz])</code>	Get the min and max value of theta of the parallel to cover the field of view.
<code>get_value(x, y)</code>	Get the value of the map at position x,y
<code>graticule([dpar, dmer, coord, local, verbose])</code>	Draw a graticule.
<code>projmap(map[, nest])</code>	
<code>projplot(*args, **kws)</code>	projplot is a wrapper around <code>matplotlib.Axes.plot()</code> to take into account
<code>projscatter(theta[, phi])</code>	Projscatter is a wrapper around <code>matplotlib.Axes.scatter()</code> to take into
<code>projtext(theta, phi, s, **kws)</code>	Projtext is a wrapper around <code>matplotlib.Axes.text()</code> to take into account
<code>set_coordprec(n)</code>	Set the number of digits after floating point for coord display.
<code>set_format(f)</code>	Set the format string for value display

### healpy.projaxes.HpxMollweideAxes.delgraticules

HpxMollweideAxes.**delgraticules** ()  
Delete all graticules previously created on the Axes.

### healpy.projaxes.HpxMollweideAxes.format\_coord

HpxMollweideAxes.**format\_coord** (*x, y*)  
Format the coordinate for display in status bar. Take projection into account.

### healpy.projaxes.HpxMollweideAxes.get\_lonlat

HpxMollweideAxes.**get\_lonlat** (*x, y*)  
Get the coordinate in the coord system of the image, in lon/lat in deg.

### healpy.projaxes.HpxMollweideAxes.get\_meridian\_interval

HpxMollweideAxes.**get\_meridian\_interval** (*vx, vy=None, vz=None*)  
Get the min and max value of phi of the meridians to cover the field of view.

**Input:**

- the normalized vector of the direction of the center of the projection, in the reference frame of the graticule.

**Return:**

- **vmin,vmax** [the interval of phi for the] meridians crossing the field of view.

### healpy.projaxes.HpxMollweideAxes.get\_parallel\_interval

HpxMollweideAxes.**get\_parallel\_interval** (vx, vy=None, vz=None)

Get the min and max value of theta of the parallel to cover the field of view.

**Input:**

- the normalized vector of the direction of the center of the projection, in the reference frame of the graticule.

**Return:**

- **vmin,vmax** [between 0 and pi, vmin<vmax, the interval of theta] for the parallels crossing the field of view

### healpy.projaxes.HpxMollweideAxes.get\_value

HpxMollweideAxes.**get\_value** (x, y)

Get the value of the map at position x,y

### healpy.projaxes.HpxMollweideAxes.graticule

HpxMollweideAxes.**graticule** (dpar=None, dmer=None, coord=None, local=None, verbose=True, \*\*kws)

Draw a graticule.

**Input:**

- dpar: angular separation between parallels in degree
- dmer: angular separation between meridians in degree
- coord: coordinate system of the graticule ('G', 'E' or 'C')
- local: if True, no rotation performed at all

### healpy.projaxes.HpxMollweideAxes.projmap

HpxMollweideAxes.**projmap** (map, nest=False, \*\*kws)

### healpy.projaxes.HpxMollweideAxes.projplot

HpxMollweideAxes.**projplot** (\*args, \*\*kws)

projplot is a wrapper around matplotlib.Axes.plot() to take into account the spherical projection.

You can call this function as:

```

projplot(theta, phi)          # plot a line going through points at coord (theta, phi)
projplot(theta, phi, 'bo')    # plot 'o' in blue at coord (theta, phi)
projplot(thetaphi)           # plot a line going through points at coord (thetaphi[0], thetaphi[1])
projplot(thetaphi, 'bx')      # idem but with blue 'x'

```

**Parameters** *theta, phi* : float, array-like

Coordinates of point to plot. Can be put into one 2-d array, first line is then *theta* and second line is *phi*. See *lonlat* parameter for unit.

**fmt** : str

A format string (see `matplotlib.Axes.plot()` for details)

**lonlat** : bool, optional

If True, *theta* and *phi* are interpreted as longitude and latitude in degree, otherwise, as colatitude and longitude in radian

**coord** : {'E', 'G', 'C', None}

The coordinate system of the points, only used if the coordinate system of the Axes has been defined and in this case, a rotation is performed

**rot** : None or sequence

rotation to be applied =(lon, lat, psi) : lon, lat will be position of the new Z axis, and psi is rotation around this axis, all in degree. if None, no rotation is performed

**direct** : bool

if True, the rotation to center the projection is not taken into account

**See Also:**

`projscatter`, `projtext`

**Notes**

Other keywords are passed to `matplotlib.Axes.plot()`.

### healpy.projaxes.HpxMollweideAxes.projscatter

`HpxMollweideAxes.projscatter` (*theta, phi=None, \*args, \*\*kws*)

`Projscatter` is a wrapper around `matplotlib.Axes.scatter()` to take into account the spherical projection.

You can call this function as:

```

projscatter(theta, phi)      # plot points at coord (theta, phi)
projplot(thetaphi)          # plot points at coord (thetaphi[0], thetaphi[1])

```

**Parameters** *theta, phi* : float, array-like

Coordinates of point to plot. Can be put into one 2-d array, first line is then *theta* and second line is *phi*. See *lonlat* parameter for unit.

**lonlat** : bool, optional

If True, theta and phi are interpreted as longitude and latitude in degree, otherwise, as colatitude and longitude in radian

**coord** : {'E', 'G', 'C', None}, optional

The coordinate system of the points, only used if the coordinate coordinate system of the axes has been defined and in this case, a rotation is performed

**rot** : None or sequence, optional

rotation to be applied =(lon, lat, psi) : lon, lat will be position of the new Z axis, and psi is rotation around this axis, all in degree. if None, no rotation is performed

**direct** : bool, optional

if True, the rotation to center the projection is not taken into account

**See Also:**

[projplot](#), [projtext](#)

**Notes**

Other keywords are passed to `matplotlib.Axes.plot()`.

**healpy.projaxes.HpxMollweideAxes.projtext**

`HpxMollweideAxes.projtext(theta, phi, s, **kws)`

`Projtext` is a wrapper around `matplotlib.Axes.text()` to take into account the spherical projection.

**Parameters** **theta, phi** : float, array-like

Coordinates of point to plot. Can be put into one 2-d array, first line is then *theta* and second line is *phi*. See *lonlat* parameter for unit.

**text** : str

The text to be displayed.

**lonlat** : bool, optional

If True, theta and phi are interpreted as longitude and latitude in degree, otherwise, as colatitude and longitude in radian

**coord** : {'E', 'G', 'C', None}, optional

The coordinate system of the points, only used if the coordinate coordinate system of the axes has been defined and in this case, a rotation is performed

**rot** : None or sequence, optional

rotation to be applied =(lon, lat, psi) : lon, lat will be position of the new Z axis, and psi is rotation around this axis, all in degree. if None, no rotation is performed

**direct** : bool, optional

if True, the rotation to center the projection is not taken into account

**See Also:**

[projplot](#), [projscatter](#)

## Notes

Other keywords are passed to `matplotlib.Axes.text()`.

### `healpy.projaxes.HpxMollweideAxes.set_coordprec`

`HpxMollweideAxes.set_coordprec(n)`

Set the number of digits after floating point for coord display.

### `healpy.projaxes.HpxMollweideAxes.set_format`

`HpxMollweideAxes.set_format(f)`

Set the format string for value display

## `healpy.projaxes.CartesianAxes`

**class** `healpy.projaxes.CartesianAxes(*args, **kwargs)`

Define a cylindrical Axes to handle cylindrical projection.

## Methods

<code>delgraticules()</code>	Delete all graticules previously created on the Axes.
<code>format_coord(x, y)</code>	Format the coordinate for display in status bar.
<code>get_lonlat(x, y)</code>	Get the coordinate in the coord system of the image, in lon/lat in deg.
<code>get_meridian_interval(vx[, vy, vz])</code>	Get the min and max value of phi of the meridians to cover the field of view.
<code>get_parallel_interval(vx[, vy, vz])</code>	Get the min and max value of theta of the parallel to cover the field of view.
<code>get_value(x, y)</code>	Get the value of the map at position x,y
<code>graticule([dpar, dmer, coord, local, verbose])</code>	Draw a graticule.
<code>projmap(map, vec2pix_func[, xsize, ysize, ...])</code>	
<code>projplot(*args, **kwargs)</code>	<code>projplot</code> is a wrapper around <code>matplotlib.Axes.plot()</code> to take into account projection.
<code>projscatter(theta[, phi])</code>	<code>Projscatter</code> is a wrapper around <code>matplotlib.Axes.scatter()</code> to take into account projection.
<code>projtext(theta, phi, s, **kwargs)</code>	<code>Projtext</code> is a wrapper around <code>matplotlib.Axes.text()</code> to take into account projection.
<code>set_coordprec(n)</code>	Set the number of digits after floating point for coord display.
<code>set_format(f)</code>	Set the format string for value display

### `healpy.projaxes.CartesianAxes.delgraticules`

`CartesianAxes.delgraticules()`

Delete all graticules previously created on the Axes.

### `healpy.projaxes.CartesianAxes.format_coord`

`CartesianAxes.format_coord(x, y)`

Format the coordinate for display in status bar. Take projection into account.

### healpy.projaxes.CartesianAxes.get\_lonlat

`CartesianAxes.get_lonlat(x, y)`

Get the coordinate in the coord system of the image, in lon/lat in deg.

### healpy.projaxes.CartesianAxes.get\_meridian\_interval

`CartesianAxes.get_meridian_interval(vx, vy=None, vz=None)`

Get the min and max value of phi of the meridians to cover the field of view.

#### Input:

- the normalized vector of the direction of the center of the projection, in the reference frame of the graticule.

#### Return:

- **vmin,vmax** [the interval of phi for the] meridians crossing the field of view.

### healpy.projaxes.CartesianAxes.get\_parallel\_interval

`CartesianAxes.get_parallel_interval(vx, vy=None, vz=None)`

Get the min and max value of theta of the parallel to cover the field of view.

#### Input:

- the normalized vector of the direction of the center of the projection, in the reference frame of the graticule.

#### Return:

- **vmin,vmax** [between 0 and pi, vmin<vmax, the interval of theta] for the parallels crossing the field of view

### healpy.projaxes.CartesianAxes.get\_value

`CartesianAxes.get_value(x, y)`

Get the value of the map at position x,y

### healpy.projaxes.CartesianAxes.graticule

`CartesianAxes.graticule(dpar=None, dmer=None, coord=None, local=None, verbose=True, **kwds)`

Draw a graticule.

#### Input:

- dpar: angular separation between parallels in degree
- dmer: angular separation between meridians in degree
- coord: coordinate system of the graticule ('G', 'E' or 'C')
- local: if True, no rotation performed at all



### healpy.projaxes.CartesianAxes.projmap

`CartesianAxes.projmap` (*map*, *vec2pix\_func*, *xsize*=800, *ysize*=None, *lonra*=None, *latra*=None, *\*\*kws*)

### healpy.projaxes.CartesianAxes.projplot

`CartesianAxes.projplot` (*\*args*, *\*\*kws*)

`projplot` is a wrapper around `matplotlib.Axes.plot()` to take into account the spherical projection.

You can call this function as:

```
projplot(theta, phi)           # plot a line going through points at coord (theta, phi)
projplot(theta, phi, 'bo')     # plot 'o' in blue at coord (theta, phi)
projplot(thetaphi)            # plot a line going through points at coord (thetaphi[0], thetaphi[1])
projplot(thetaphi, 'bx')       # idem but with blue 'x'
```

**Parameters** *theta, phi* : float, array-like

Coordinates of point to plot. Can be put into one 2-d array, first line is then *theta* and second line is *phi*. See *lonlat* parameter for unit.

**fmt** : str

A format string (see `matplotlib.Axes.plot()` for details)

**lonlat** : bool, optional

If True, *theta* and *phi* are interpreted as longitude and latitude in degree, otherwise, as colatitude and longitude in radian

**coord** : {'E', 'G', 'C', None}

The coordinate system of the points, only used if the coordinate system of the Axes has been defined and in this case, a rotation is performed

**rot** : None or sequence

rotation to be applied =(lon, lat, psi) : lon, lat will be position of the new Z axis, and psi is rotation around this axis, all in degree. if None, no rotation is performed

**direct** : bool

if True, the rotation to center the projection is not taken into account

**See Also:**

`projscatter`, `projtext`

### Notes

Other keywords are passed to `matplotlib.Axes.plot()`.

### healpy.projaxes.CartesianAxes.projscatter

`CartesianAxes.projscatter` (*theta, phi*=None, *\*args*, *\*\*kws*)

`Projscatter` is a wrapper around `matplotlib.Axes.scatter()` to take into account the spherical projection.

You can call this function as:

```
projscatter(theta, phi)      # plot points at coord (theta, phi)
projplot(thetaphi)          # plot points at coord (thetaphi[0], thetaphi[1])
```

**Parameters** *theta, phi* : float, array-like

Coordinates of point to plot. Can be put into one 2-d array, first line is then *theta* and second line is *phi*. See *lonlat* parameter for unit.

**lonlat** : bool, optional

If True, theta and phi are interpreted as longitude and latitude in degree, otherwise, as colatitude and longitude in radian

**coord** : { 'E', 'G', 'C', None }, optional

The coordinate system of the points, only used if the coordinate coordinate system of the axes has been defined and in this case, a rotation is performed

**rot** : None or sequence, optional

rotation to be applied =(lon, lat, psi) : lon, lat will be position of the new Z axis, and psi is rotation around this axis, all in degree. if None, no rotation is performed

**direct** : bool, optional

if True, the rotation to center the projection is not taken into account

**See Also:**

`projplot`, `projtext`

## Notes

Other keywords are passed to `matplotlib.Axes.plot()`.

## healpy.projaxes.CartesianAxes.projtext

`CartesianAxes.projtext` (*theta, phi, s, \*\*kws*)

`Projtext` is a wrapper around `matplotlib.Axes.text()` to take into account the spherical projection.

**Parameters** *theta, phi* : float, array-like

Coordinates of point to plot. Can be put into one 2-d array, first line is then *theta* and second line is *phi*. See *lonlat* parameter for unit.

**text** : str

The text to be displayed.

**lonlat** : bool, optional

If True, theta and phi are interpreted as longitude and latitude in degree, otherwise, as colatitude and longitude in radian

**coord** : { 'E', 'G', 'C', None }, optional

The coordinate system of the points, only used if the coordinate coordinate system of the axes has been defined and in this case, a rotation is performed

**rot** : None or sequence, optional

rotation to be applied =(lon, lat, psi) : lon, lat will be position of the new Z axis, and psi is rotation around this axis, all in degree. if None, no rotation is performed

**direct** : bool, optional

if True, the rotation to center the projection is not taken into account

**See Also:**

`projplot`, `projscatter`

## Notes

Other keywords are passed to `matplotlib.Axes.text()`.

## healpy.projaxes.CartesianAxes.set\_coordprec

`CartesianAxes.set_coordprec(n)`

Set the number of digits after floating point for coord display.

## healpy.projaxes.CartesianAxes.set\_format

`CartesianAxes.set_format(f)`

Set the format string for value display

## healpy.projaxes.HpxCartesianAxes

`class healpy.projaxes.HpxCartesianAxes(*args, **kws)`

## Methods

<code>delgraticules()</code>	Delete all graticules previously created on the Axes.
<code>format_coord(x, y)</code>	Format the coordinate for display in status bar.
<code>get_lonlat(x, y)</code>	Get the coordinate in the coord system of the image, in lon/lat in deg.
<code>get_meridian_interval(vx[, vy, vz])</code>	Get the min and max value of phi of the meridians to cover the field of view.
<code>get_parallel_interval(vx[, vy, vz])</code>	Get the min and max value of theta of the parallel to cover the field of view.
<code>get_value(x, y)</code>	Get the value of the map at position x,y
<code>graticule([dpar, dmer, coord, local, verbose])</code>	Draw a graticule.
<code>projmap(map[, nest])</code>	
<code>projplot(*args, **kws)</code>	<code>projplot</code> is a wrapper around <code>matplotlib.Axes.plot()</code> to take into account
<code>projscatter(theta[, phi])</code>	<code>Projscatter</code> is a wrapper around <code>matplotlib.Axes.scatter()</code> to take into
<code>projtext(theta, phi, s, **kws)</code>	<code>Projtext</code> is a wrapper around <code>matplotlib.Axes.text()</code> to take into account
<code>set_coordprec(n)</code>	Set the number of digits after floating point for coord display.
<code>set_format(f)</code>	Set the format string for value display

## healpy.projaxes.HpxCartesianAxes.delgraticules

`HpxCartesianAxes.delgraticules()`

Delete all graticules previously created on the Axes.

### healpy.projaxes.HpxCartesianAxes.format\_coord

HpxCartesianAxes.**format\_coord**(x, y)

Format the coordinate for display in status bar. Take projection into account.

### healpy.projaxes.HpxCartesianAxes.get\_lonlat

HpxCartesianAxes.**get\_lonlat**(x, y)

Get the coordinate in the coord system of the image, in lon/lat in deg.

### healpy.projaxes.HpxCartesianAxes.get\_meridian\_interval

HpxCartesianAxes.**get\_meridian\_interval**(vx, vy=None, vz=None)

Get the min and max value of phi of the meridians to cover the field of view.

#### Input:

- the normalized vector of the direction of the center of the projection, in the reference frame of the graticule.

#### Return:

- **vmin,vmax** [the interval of phi for the] meridians crossing the field of view.

### healpy.projaxes.HpxCartesianAxes.get\_parallel\_interval

HpxCartesianAxes.**get\_parallel\_interval**(vx, vy=None, vz=None)

Get the min and max value of theta of the parallel to cover the field of view.

#### Input:

- the normalized vector of the direction of the center of the projection, in the reference frame of the graticule.

#### Return:

- **vmin,vmax** [between 0 and pi, vmin<vmax, the interval of theta] for the parallels crossing the field of view

### healpy.projaxes.HpxCartesianAxes.get\_value

HpxCartesianAxes.**get\_value**(x, y)

Get the value of the map at position x,y

### healpy.projaxes.HpxCartesianAxes.graticule

HpxCartesianAxes.**graticule**(dpar=None, dmer=None, coord=None, local=None, verbose=True, \*\*kws)

Draw a graticule.

#### Input:

- dpar: angular separation between parallels in degree
- dmer: angular separation between meridians in degree

- coord: coordinate system of the graticule ('G', 'E' or 'C')
- local: if True, no rotation performed at all

### healpy.projaxes.HpxCartesianAxes.projmap

HpxCartesianAxes.**projmap** (*map*, *nest=False*, *\*\*kws*)

### healpy.projaxes.HpxCartesianAxes.projplot

HpxCartesianAxes.**projplot** (*\*args*, *\*\*kws*)

projplot is a wrapper around `matplotlib.Axes.plot()` to take into account the spherical projection.

You can call this function as:

```
projplot(theta, phi)           # plot a line going through points at coord (theta, phi)
projplot(theta, phi, 'bo')    # plot 'o' in blue at coord (theta, phi)
projplot(thetaphi)           # plot a line going through points at coord (thetaphi[0], thetaphi[1])
projplot(thetaphi, 'bx')      # idem but with blue 'x'
```

**Parameters** **theta, phi** : float, array-like

Coordinates of point to plot. Can be put into one 2-d array, first line is then *theta* and second line is *phi*. See *lonlat* parameter for unit.

**fmt** : str

A format string (see `matplotlib.Axes.plot()` for details)

**lonlat** : bool, optional

If True, theta and phi are interpreted as longitude and latitude in degree, otherwise, as colatitude and longitude in radian

**coord** : {'E', 'G', 'C', None}

The coordinate system of the points, only used if the coordinate coordinate system of the Axes has been defined and in this case, a rotation is performed

**rot** : None or sequence

rotation to be applied =(lon, lat, psi) : lon, lat will be position of the new Z axis, and psi is rotation around this axis, all in degree. if None, no rotation is performed

**direct** : bool

if True, the rotation to center the projection is not taken into account

**See Also:**

`projscatter`, `projtext`

### Notes

Other keywords are passed to `matplotlib.Axes.plot()`.

### healpy.projaxes.HpxCartesianAxes.projscatter

HpxCartesianAxes.**projscatter** (*theta*, *phi*=None, \*args, \*\*kws)

Projscatter is a wrapper around `matplotlib.Axes.scatter()` to take into account the spherical projection.

You can call this function as:

```
projscatter(theta, phi)           # plot points at coord (theta, phi)
projplot(thetaphi)               # plot points at coord (thetaphi[0], thetaphi[1])
```

**Parameters** *theta*, *phi* : float, array-like

Coordinates of point to plot. Can be put into one 2-d array, first line is then *theta* and second line is *phi*. See *lonlat* parameter for unit.

**lonlat** : bool, optional

If True, *theta* and *phi* are interpreted as longitude and latitude in degree, otherwise, as colatitude and longitude in radian

**coord** : {'E', 'G', 'C', None}, optional

The coordinate system of the points, only used if the coordinate coordinate system of the axes has been defined and in this case, a rotation is performed

**rot** : None or sequence, optional

rotation to be applied =(lon, lat, psi) : lon, lat will be position of the new Z axis, and psi is rotation around this axis, all in degree. if None, no rotation is performed

**direct** : bool, optional

if True, the rotation to center the projection is not taken into account

**See Also:**

`projplot`, `projtext`

#### Notes

Other keywords are passed to `matplotlib.Axes.plot()`.

### healpy.projaxes.HpxCartesianAxes.projtext

HpxCartesianAxes.**projtext** (*theta*, *phi*, *s*, \*\*kws)

Projtext is a wrapper around `matplotlib.Axes.text()` to take into account the spherical projection.

**Parameters** *theta*, *phi* : float, array-like

Coordinates of point to plot. Can be put into one 2-d array, first line is then *theta* and second line is *phi*. See *lonlat* parameter for unit.

**text** : str

The text to be displayed.

**lonlat** : bool, optional

If True, theta and phi are interpreted as longitude and latitude in degree, otherwise, as colatitude and longitude in radian

**coord** : {'E', 'G', 'C', None}, optional

The coordinate system of the points, only used if the coordinate coordinate system of the axes has been defined and in this case, a rotation is performed

**rot** : None or sequence, optional

rotation to be applied =(lon, lat, psi) : lon, lat will be position of the new Z axis, and psi is rotation around this axis, all in degree. if None, no rotation is performed

**direct** : bool, optional

if True, the rotation to center the projection is not taken into account

**See Also:**

`projplot`, `projscatter`

### Notes

Other keywords are passed to `matplotlib.Axes.text()`.

#### `healpy.projaxes.HpxCartesianAxes.set_coordprec`

`HpxCartesianAxes.set_coordprec(n)`

Set the number of digits after floating point for coord display.

#### `healpy.projaxes.HpxCartesianAxes.set_format`

`HpxCartesianAxes.set_format(f)`

Set the format string for value display

## 3.8 zoomtool – Interactive visualisation

### 3.8.1 Interactive map visualization

`mollzoom`([map, fig, rot, coord, unit, ...]) Interactive mollweide plot with zoomed gnomview.

#### `healpy.zoomtool.mollzoom`

`healpy.zoomtool.mollzoom`(map=None, fig=None, rot=None, coord=None, unit='', xsize=800, title='Mollweide view', nest=False, min=None, max=None, flip='astro', remove\_dip=False, remove\_mono=False, gal\_cut=0, format='%g', cmap=None, norm=None, hold=False, margins=None, sub=None)

Interactive mollweide plot with zoomed gnomview.





# INDICES AND TABLES

- *genindex*
- *modindex*
- *search*