astropy-healpix Documentation

Release 0.3.1

Christoph Deil, Thomas Robitaille, and Dustin Lang

Oct 24, 2018

Contents

Warning: This **astropy-healpix** package is in an early stage of development. It should not be considered feature complete or API stable. Feedback and contributions welcome!

HEALPix (Hierarchical Equal Area isoLatitude Pixelisation) is an algorithm for pixellizing a sphere that is sometimes used in Astronomy to store data from all-sky surveys, but the general algorithm can apply to any field that has to deal with representing data on a sphere.

More information about the HEALPix algorithm can be found here:

- http://healpix.jpl.nasa.gov/
- http://adsabs.harvard.edu/abs/2005ApJ...622..759G
- http://adsabs.harvard.edu/abs/2007MNRAS.381..865C

Part I

About this package

astropy-healpix is a new BSD-licensed implementation that is separate from the original GPL-licensed HEALPix library and associated healpy Python wrapper. See about for further information about the difference between this new implementation and the original libraries.

The code can be found on GitHub, along with the list of Contributors.

Part II

User documentation

CHAPTER 1

Installation

1.1 Dependencies

1.1.1 Required dependencies

The **astropy-healpix** package works with Python 2.7 or 3.5 and later (on Linux, MacOS X and Windows), and requires the following dependencies:

- Numpy 1.10 or later
- Astropy 1.2 or later

If you use *Using pip* or *Using conda*, these will be installed automatically.

1.1.2 Optional dependencies

The following packages are optional dependencies, which can be installed if needed:

- pytest for testing
- healpy for testing (but this is not required and the tests that require healpy will be skipped if healpy is not installed)
- hypothesis for the healpy-related tests.

1.2 Stable version

Installing the latest stable version is possible either using pip or conda.

1.2.1 Using pip

To install **astropy-healpix** with pip from PyPI simply run:

```
pip install --no-deps astropy-healpix
```

Note: The `--no-deps` flag is optional, but highly recommended if you already have Numpy installed, since otherwise pip will sometimes try to "help" you by upgrading your Numpy installation, which may not always be desired.

1.2.2 Using conda

To install healpix with Anaconda from the conda-forge channel on anaconda.org simply run:

```
conda install -c conda-forge astropy-healpix
```

1.2.3 Testing installation

To check that you have this package installed and which version you're using, start Python and execute the following code:

```
$ python
Python 3.6.2 |Continuum Analytics, Inc.| (default, Jul 20 2017, 13:14:59)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import astropy_healpix
>>> astropy_healpix.__version__
0.1
```

To make sure that all functionality is working OK on your system, you can run the automated tests of this package by executing the `test` function:

```
python -c 'import astropy_healpix; astropy_healpix.test()'
```

1.3 Development version

Install the latest development version from https://github.com/astropy/astropy-healpix :

```
git clone https://github.com/astropy/astropy-healpix
cd astropy-healpix
pip install .
```

1.4 Contributing

This section contains some tips how to hack on **astropy-healpix**.

One quick way to get a Python environment with everything needed to work on `astropy-healpix` (code, run tests, build docs) is like this:

```
git clone https://github.com/astropy/astropy-healpix
cd astropy-healpix
conda env create -f environment-dev.yml
conda activate astropy-healpix
```

Run this command to do an in-place build and put this local version on your Python `sys.path`:

```
python setup.py develop
```

To run the tests, use `pytest` directly:

```
python -m pytest -v astropy_healpix
```

To build the docs, use this command:

```
python setup.py build_docs
open docs/_build/html/index.html
```

If you have any questions, just open an issue on Github and we'll help.

CHAPTER 2

Getting started

The cleanest way to use the functionality in **healpix** is to make use of the high-level `HEALPix` class. The `HEALPix` class should be initialized with the `nside` parameter which controls the resolution of the pixellization - it is the number of pixels on the side of each of the 12 top-level HEALPix pixels:

```
>>> from astropy_healpix import HEALPix
>>> hp = HEALPix(nside=16)
```

As described in the references above, HEALPix pixel indices can follow two different ordering conventions - the *nested* convention and the *ring* convention. By default, the `HEALPix` class assumes the ring ordering convention, but it is possible to explicitly specify the convention to use using the `order` argument, for example:

```
>>> hp = HEALPix(nside=16, order='ring')
```

or:

```
>>> hp = HEALPix(nside=16, order='nested')
```

Once this class has been set up, you can access various properties and methods related to the HEALPix pixellization. For example, you can calculate the number of pixels as well as the pixel area or resolution:

```
>>> hp.npix
3072
>>> hp.pixel_area
<Quantity 0.0040906154343617095 sr>
>>> hp.pixel_resolution
<Quantity 219.87113035631398 arcmin>
```

As you can see, when appropriate the properties and the methods on the `HEALPix` class return Astropy high-level classes such as `Quantity`, `Longitude`, and so on.

For example, the `healpix_to_lonlat()` method can be used to convert HEALPix indices to `Longitude` and `Latitude` objects:

```
>>> lon, lat = hp.healpix_to_lonlat([1, 442, 2200])
>>> lon
<Longitude [ 0.83448555, 1.63624617, 0.4712389 ] rad>
>>> lat
<Latitude [ 0.08343009, 0.94842784,-0.78529135] rad>
```

The `HEALPix` class includes methods that take or return `SkyCoord` objects (we will take a look at this in the *Celestial coordinates* section).

In the subsequent sections of the documentation, we will take a closer look at converting between coordinate systems, as well as more advanced features such as interpolation and cone searches.

CHAPTER 3

Coordinate conversions

3.1 Converting between pixel indices and spherical coordinates

As described in *Getting started*, coordinates in a HEALPix pixellization can follow either the 'ring' or 'nested' convention. Let's start by setting up an example pixellization:

```
>>> from astropy_healpix import HEALPix
>>> hp = HEALPix(nside=16, order='nested')
```

The `healpix_to_lonlat()` method can be used to convert HEALPix indices to `Longitude` and `Latitude` objects:

```
>>> lon, lat = hp.healpix_to_lonlat([1, 442, 2200])
>>> lon
<Longitude [ 0.83448555, 1.63624617, 0.4712389 ] rad>
>>> lat
<Latitude [ 0.08343009, 0.94842784,-0.78529135] rad>
```

The `Longitude` and `Latitude` objects are fully-fledged `Quantity` objects and also include shortcuts to get the values in various units:

```
>>> lon.hourangle
array([ 3.1875,  6.25  ,  1.8   ])
>>> lat.degree
array([  4.78019185,  54.3409123 , -44.99388015])
```

Conversely, given longitudes and latitudes as `Quantity` objects, it is possible to recover HEALPix pixel indices:

```
>>> from astropy import units as u
>>> print(hp.lonlat_to_healpix([1, 3, 4] * u.deg, [5, 6, 9] * u.deg))
[1217 1217 1222]
```

In these examples, what is being converted is the position of the center of each pixel. In fact, the `lonlat_to_healpix()` method can also take or give the fractional position inside each HEALPix pixel, e.g.:

```
>>> index, dx, dy = hp.lonlat_to_healpix([1, 3, 4] * u.deg, [5, 6, 9] * u.deg,
...                                      return_offsets=True)
>>> print(index)
[1217 1217 1222]
>>> dx
array([ 0.22364669,  0.78767489,  0.58832469])
>>> dy
array([ 0.86809114,  0.72100823,  0.16610247])
```

and the `healpix_to_lonlat()` method can take offset positions - for example we can use this to find the position of the corners of a given pixel:

```
>>> dx = [0., 1., 1., 0.]
>>> dy = [0., 0., 1., 1.]
>>> lon, lat = hp.healpix_to_lonlat([133, 133, 133, 133], dx=dx, dy=dy)
>>> lon
<Longitude [ 0.53996124, 0.58904862, 0.53996124, 0.49087385] rad>
>>> lat
<Latitude [ 0.47611906, 0.52359878, 0.57241857, 0.52359878] rad>
```

3.2 Celestial coordinates

For cases where the HEALPix pixellization is of the celestial sphere, a `frame` argument can be passed to `HEALPix`. This argument should specify the celestial frame (using an astropy.coordinates frame) in which the HEALPix pixellization is defined:

```
>>> from astropy_healpix import HEALPix
>>> from astropy.coordinates import Galactic
>>> hp = HEALPix(nside=16, order='nested', frame=Galactic())
```

Each method defined in `HEALPix` and ending in `lonlat` has an equivalent method ending in `skycoord` which can be used if the frame is set. For example, to convert from HEALPix indices to celestial coordinates, you can use the `healpix_to_skycoord()` method:

```
>>> hp.healpix_to_skycoord([144, 231])
<SkyCoord (Galactic): (l, b) in deg
    [( 33.75      ,  32.7971683 ), ( 32.14285714,  69.42254649)]>
```

and to convert from celestial coordinates to HEALPix indices you can use the `skycoord_to_healpix()` method, e.g:

```
>>> from astropy.coordinates import SkyCoord
>>> coord = SkyCoord('00h42m44.3503s +41d16m08.634s')
>>> hp.skycoord_to_healpix(coord)
2537
```

3.3 Converting between ring and nested conventions

The `HEALPix` class has methods that can be used to convert HEALPix pixel indices between the ring and nested convention. These are `nested_to_ring()`:

```
>>> print(hp.nested_to_ring([30]))
[873]
```

and ring_to_nested():

```
>>> print(hp.ring_to_nested([1, 2, 3]))
[ 511  767 1023]
```

CHAPTER 4

Pixel corners and edges

In some cases, you may need to find out the longitude/latitude or celestial coordinates of the corners or edges of HEALPix pixels.

The `boundaries_lonlat()` method can be used to sample points long the edge of one or more HEALPix pixels:

```
>>> from astropy_healpix import HEALPix
>>> hp = HEALPix(nside=16, order='nested')
>>> hp.boundaries_lonlat([120], step=1)
(<Longitude [[ 1.17809725, 1.08747438, 1.12199738, 1.20830487]] rad>, <Latitude [[ 0.94842784, 0.
↪89458259, 0.84022258, 0.89458259]] rad>)
```

This method takes a `step` argument which specifies how many points to sample along each edge. Setting `step` to 1 returns the corner positions, while setting e.g. 2 returns the corners and points along the middle of each edge, and larger values can be used to get the precise curved edges of the pixels.

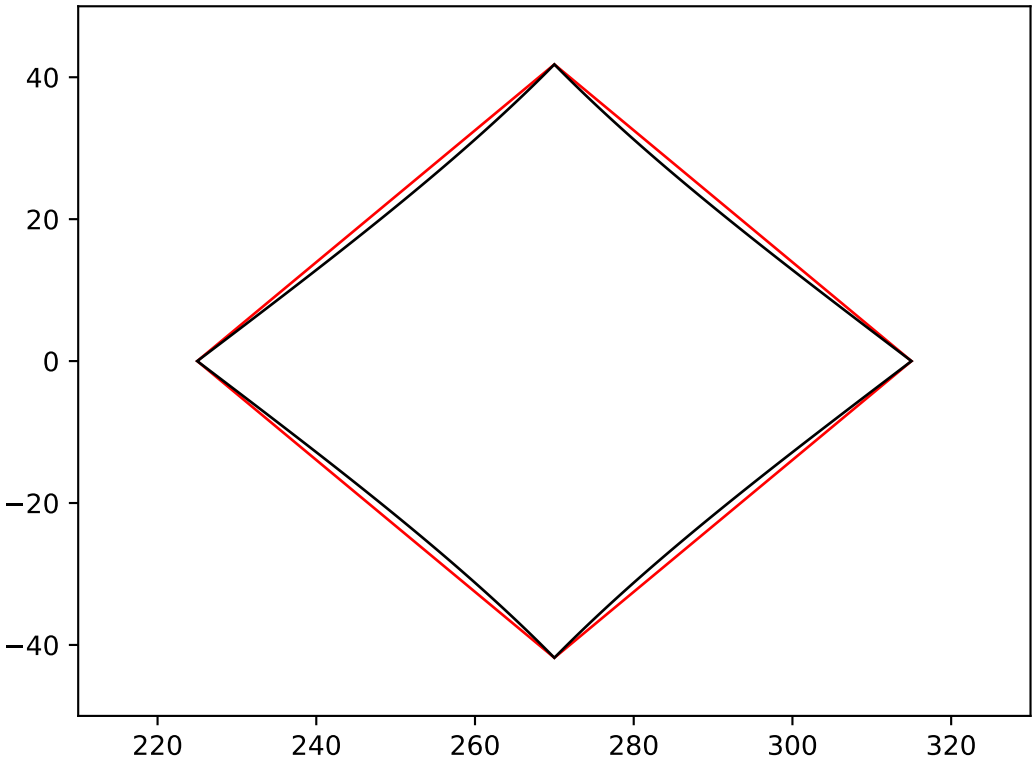
The following example shows the difference between the boundary constructed from just the corners (in red) and a much higher-resolution boundary computed with 100 steps on each side (in black):

```
import numpy as np
from astropy import units as u
import matplotlib.pyplot as plt
from matplotlib.patches import Polygon
from astropy_healpix.core import boundaries_lonlat

ax = plt.subplot(1, 1, 1)

for step, color in [(1, 'red'), (100, 'black')]:
    lon, lat = boundaries_lonlat([7], nside=1, step=step)
    lon = lon.to(u.deg).value
    lat = lat.to(u.deg).value
    vertices = np.vstack([lon.ravel(), lat.ravel()]).transpose()
    p = Polygon(vertices, closed=True, edgecolor=color, facecolor='none')
    ax.add_patch(p)
```

(continues on next page)

(continued from previous page)

```
plt.xlim(210, 330)
plt.ylim(-50, 50)
```



As for other methods, the `HEALPix` class has an equivalent `boundaries_skycoord()` method that can return the celestial coordinates of the boundaries as a `SkyCoord` object if the `frame` is set:

```
>>> from astropy.coordinates import Galactic
>>> hp = HEALPix(nside=16, order='nested', frame=Galactic())
>>> hp.boundaries_skycoord([120], step=1)
<SkyCoord (Galactic): (l, b) in deg
    [[( 67.5       ,  54.3409123 ), ( 62.30769231,  51.25580695),
      ( 64.28571429,  48.14120779), ( 69.23076923,  51.25580695)]]>
```

CHAPTER 5

Seaching for pixels around a position (cone search)

A common operation when using HEALPix maps is to try and find all pixels that lie within a certain radius of a given longitude/latitude. One way to do this would be to simply find the longitude/latitude of all pixels in the HEALPix map then find the spherical distance to the requested longitude and latitude, but in practice this would be very inefficient for high resolution HEALPix maps where the number of pixels may become arbitrarily large.

Instead, the `cone_search_lonlat()` method can be used to efficiently find all HEALpix pixels within a certain radius from a longitude/latitude:

```
>>> from astropy import units as u
>>> from astropy_healpix import HEALPix
>>> hp = HEALPix(nside=16, order='nested')
>>> print(hp.cone_search_lonlat(10 * u.deg, 30 * u.deg, radius=10 * u.deg))
[1269  160  162 1271 1270 1268 1246 1247  138  139  161 1245  136  137
  140  142  130  131 1239 1244 1238 1241 1243 1265 1267 1276 1273 1277
  168  169  163  166  164]
```

Likewise, if a celestial frame was specified using the `frame` keyword arguent to `HEALPix`, you can use the `cone_search_skycoord()` method to query around specific celestial coordinates:

```
>>> from astropy.coordinates import Galactic
>>> hp = HEALPix(nside=16, order='nested', frame=Galactic())
>>> from astropy.coordinates import SkyCoord
>>> coord = SkyCoord('00h42m44.3503s +41d16m08.634s')
>>> print(hp.cone_search_skycoord(coord, radius=5 * u.arcmin))
[2537]
```

CHAPTER 6

Interpolating values from a HEALPix map

6.1 Main methods

While all the functionality we have seen so far in the remainder of the documentation is concerned with the geometry of the HEALPix pixellization, the main purpose of HEALPix is to actually tabulate values in each pixel to represent a physical quantity over a sphere (e.g. flux over the celestial sphere). We will refer to this as a HEALPix map.

These maps are stored using a 1-dimensional vector with as many elements as pixels in the HEALPix pixellization, and either in the 'ring' or 'nested' order.

If you are interested in finding the value in a HEALPix map at a given longitude/latitude on the sphere, there are two main options:

- Convert the longitude/latitude to the HEALPix pixel that the position falls inside (e.g. `index`) using `lonlat_to_healpix()` or `skycoord_to_healpix()`, and extract the value of the array of map values at that index (e.g. `values[index]`). This is essentially equivalent to a nearest-neighbour interpolation.
- Convert the longitude/latitude to the HEALPix pixel that the position falls inside then find the other neighboring pixels and carry out a bilinear interpolation. This is trickier to do by hand, and we therefore provide the methods `interpolate_bilinear_lonlat()` and `interpolate_bilinear_skycoord()` methods to faciliate this. If you are not already familiar with how to access HEALPix data from FITS files, we have provided a *Full example* in the following section.

6.2 Full example

To illustrate this, we use an example map from the WMAP mission, specifically the map **K Band Map for the Full Five Years**. We start off by downloading and opening this map with Astropy:

```
>>> from astropy.io import fits
>>> hdulist = fits.open('https://lambda.gsfc.nasa.gov/data/map/dr3/skymaps/5yr//wmap_band_imap_r9_5yr_K_
↪v3.fits')
Downloading https://lambda.gsfc.nasa.gov/data/map/dr3/skymaps/5yr//wmap_band_imap_r9_5yr_K_v3.fits␣
↪[Done]
```

(continues on next page)

(continued from previous page)

```
>>> hdulist.info()
Filename: ...
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU      19   ()
  1  Archive Map Table    1 BinTableHDU     20   3145728R x 2C   [E, E]
```

Since HEALPix maps are stored in tabular form, the data is contained in HDU 1 (primary HDUs cannot contain tabular data).

Let's now take a look at the header:

```
>>> hdulist[1].header
XTENSION= 'BINTABLE'           /binary table extension
BITPIX  =                    8 /8-bit bytes
NAXIS   =                    2 /2-dimensional binary table
NAXIS1  =                    8 /width of table in bytes
NAXIS2  =              3145728 /number of rows in table
PCOUNT  =                    0 /size of special data area
GCOUNT  =                    1 /one data group (required keyword)
TFIELDS =                    2 /number of fields in each row
TTYPE1  = 'TEMPERATURE '       /label for field 1
TFORM1  = 'E       '           /data format of field: 4-byte REAL
TUNIT1  = 'mK      '           /physical unit of field 1
TTYPE2  = 'N_OBS   '           /label for field 2
TFORM2  = 'E       '           /data format of field: 4-byte REAL
TUNIT2  = 'counts  '           /physical unit of field 2
EXTNAME = 'Archive Map Table'  /name of this binary table extension
PIXTYPE = 'HEALPIX '           /Pixel algorigthm
ORDERING= 'NESTED  '           /Ordering scheme
NSIDE   =                  512 /Resolution parameter
FIRSTPIX=                    0 /First pixel (0 based)
LASTPIX =              3145727 /Last pixel (0 based)
```

Of particular interest to us are the `NSIDE` and `ORDERING` keywords:

```
>>> hdulist[1].header['NSIDE']
512
>>> hdulist[1].header['ORDERING']
'NESTED'
```

The data itself can be accessed using:

```
>>> hdulist[1].data['TEMPERATURE']
array([ 16.28499985,  16.8025322 ,  15.32036781, ...,  15.0780201 ,
    15.36229229,  15.23281574], dtype=float32)
```

The last piece of information we need is that the map is in Galactic coordinates, which is unfortunately not encoded in the header but can be found here.

We can now instantiate a `HEALPix` object:

```
>>> from astropy_healpix import HEALPix
>>> from astropy.coordinates import Galactic
>>> nside = hdulist[1].header['NSIDE']
>>> order = hdulist[1].header['ORDERING']
>>> hp = HEALPix(nside=nside, order=order, frame=Galactic())
```

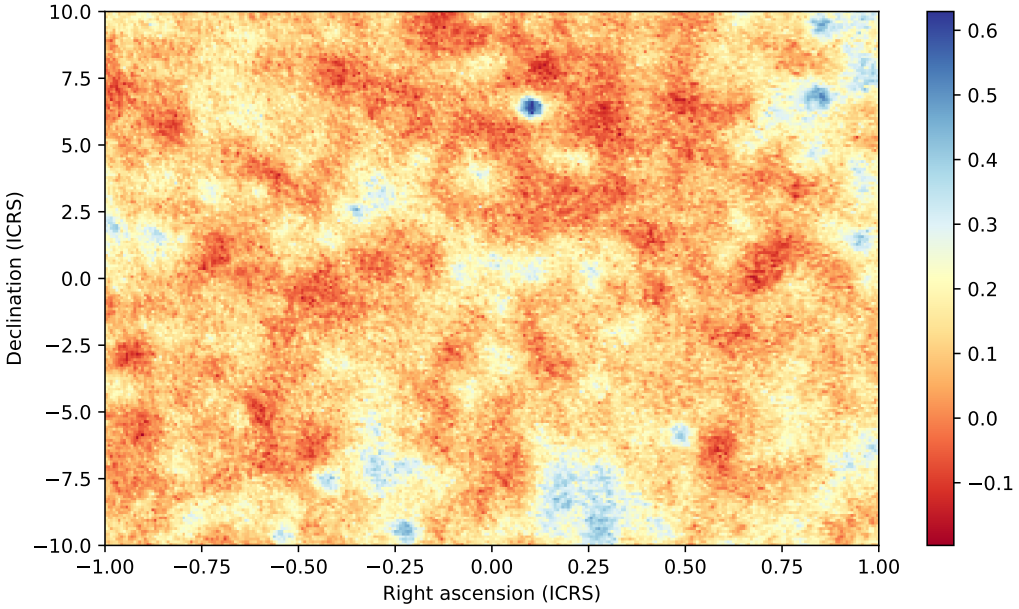
and we can now use `interpolate_bilinear_skycoord()` to interpolate the temperature at a given position on the sky:

```
>>> from astropy.coordinates import SkyCoord
>>> coord = SkyCoord('00h42m44.3503s +41d16m08.634s', frame='icrs')
>>> temperature = hdulist[1].data['temperature']
>>> hp.interpolate_bilinear_skycoord(coord, temperature)
array([ 0.41296058])
```

Here is a full example that uses this to make a map of a section of the sky:

```
# Get the data
from astropy.io import fits
hdulist = fits.open('https://lambda.gsfc.nasa.gov/data/map/dr3/skymaps/5yr//wmap_band_imap_r9_5yr_K_v3.
↪fits')

# Set up the HEALPix projection
from astropy_healpix import HEALPix
from astropy.coordinates import Galactic
nside = hdulist[1].header['NSIDE']
order = hdulist[1].header['ORDERING']
hp = HEALPix(nside=nside, order=order, frame=Galactic())

# Sample a 300x200 grid in RA/Dec
from astropy import units as u
ra = np.linspace(-15., 15., 300) * u.deg
dec = np.linspace(-10., 10., 200) * u.deg
ra_grid, dec_grid = np.meshgrid(ra, dec)

# Set up Astropy coordinate objects
from astropy.coordinates import SkyCoord
coords = SkyCoord(ra_grid.ravel(), dec_grid.ravel(), frame='icrs')

# Interpolate values
temperature = hdulist[1].data['temperature']
tmap = hp.interpolate_bilinear_skycoord(coords, temperature)
tmap = tmap.reshape((200, 300))

# Make a plot of the interpolated temperatures
plt.figure(figsize=(9, 5))
im = plt.imshow(tmap, extent=[-1, 1, -10, 10], cmap=plt.cm.RdYlBu, aspect='auto')
plt.colorbar(im)
plt.xlabel('Right ascension (ICRS)')
plt.ylabel('Declination (ICRS)')
plt.show()
```

In practice, for the common case of reprojecting a HEALPix map to a regular gridded image, you can use the reproject package which provides high-level reprojection functions that use **healpix** behind the scenes.


10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
Declination (ICRS)
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
Right ascension (ICRS)
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−0.1

CHAPTER 7

Performance

At this time, we have focused mostly on implementing functionality into the **astropy-healpix** package, performance is not as good in most cases as the healpy library. Once the API is stable, we will focus on improving performance.

7.1 Benchmark

To get an idea of how the performance of the two packages compare, we have included some simple benchmarks that compare the healpy-compatible interface of **astropy-healpix** with healpy itself. These benchmarks are run with:

```
$ python -m astropy_healpix.bench
Running benchmarks...

  fct    nest nside   size  time_healpy time_self   ratio
------- ----- ----- ------- ----------- ---------- -------
pix2ang  True     1      10   0.0000081  0.0003575   43.91
pix2ang  True   128      10   0.0000082  0.0003471   42.52
pix2ang  True     1    1000   0.0000399  0.0004751   11.92
pix2ang  True   128    1000   0.0000345  0.0004575   13.28
pix2ang  True     1 1000000   0.0434032  0.1589150    3.66
pix2ang  True   128 1000000   0.0364285  0.1383810    3.80
pix2ang False     1      10   0.0000080  0.0004040   50.30
pix2ang False   128      10   0.0000082  0.0003322   40.63
pix2ang False     1    1000   0.0000400  0.0005005   12.50
pix2ang False   128    1000   0.0000548  0.0005045    9.21
pix2ang False     1 1000000   0.0342841  0.1429310    4.17
pix2ang False   128 1000000   0.0478645  0.1405270    2.94
```

For small arrays, `pix2ang` in **astropy-healpix** performs worse, but in both caes the times are less than a millisecond, and such differences may therefore not matter. For larger arrays, the difference is a factor of a few at most. We will add more benchmarks over time to provide a more complete picture.

CHAPTER 8

Healpy-compatible interface

In addition to the above high- and low-level interfaces, we have provided a healpy-compatible interface in `astropy_healpix.healpy`. Note that this only includes a subset of the healpy functions. This is not the recommended interface, and is only provided as a convenience for packages that want to support both healpy and this package.

8.1 Example

As an example, the `pix2ang()` function can be used to get the longitude/latitude of a given HEALPix pixel (by default using the 'ring' convention):

```
>>> from astropy_healpix.healpy import pix2ang
>>> pix2ang(16, [100, 120])
(array([ 0.35914432,  0.41113786]), array([ 3.70259134,  1.6689711 ]))
```

which agrees exactly with the healpy function:

```
.. doctest-requires:: healpy
```

```
>>> from healpy import pix2ang
>>> pix2ang(16, [100, 120])
(array([ 0.35914432,  0.41113786]), array([ 3.70259134,  1.6689711 ]))
```

8.2 Migrate

To migrate a script or package from using `healpy` to this `healpix` package, to check if the required functionality is available by changing all:

```
import healpy as hp
```

to:

```
from astropy_healpix import healpy as hp
```

and see what's missing or breaks. Please file issues or feature requests!

As mentioned above, we then recommend that when you actually make the change, you use the main API of this package instead of the `healpy`-compatible interface.

CHAPTER 9

Reference/API

9.1 astropy_healpix Package

9.1.1 Functions

bilinear_interpolation_weights(lon, lat, nside)	Get the four neighbours for each (lon, lat) position and the weight associated with each one for bilinear interpolation.
healpix_to_lonlat(healpix_index, nside[, ...])	Convert HEALPix indices (optionally with offsets) to longitudes/latitudes.
interpolate_bilinear_lonlat(lon, lat, values)	Interpolate values at specific longitudes/latitudes using bilinear interpolation
level_ipix_to_uniq(level, ipix)	Convert a level and HEALPix index into a uniq number representing the cell.
level_to_nside(level)	Find the pixel dimensions of the top-level HEALPix tiles.
lonlat_to_healpix(lon, lat, nside[, ...])	Convert longitudes/latitudes to HEALPix indices
neighbours(healpix_index, nside[, order])	Find all the HEALPix pixels that are the neighbours of a HEALPix pixel
npix_to_nside(npix)	Find the number of pixels on the side of one of the 12 'top-level' HEALPix tiles given a total number of pixels.
nside_to_level(nside)	Find the HEALPix level for a given nside.
nside_to_npix(nside)	Find the number of pixels corresponding to a HEALPix resolution.
nside_to_pixel_area(nside)	Find the area of HEALPix pixels given the pixel dimensions of one of the 12 'top-level' HEALPix tiles.
nside_to_pixel_resolution(nside)	Find the resolution of HEALPix pixels given the pixel dimensions of one of the 12 'top-level' HEALPix tiles.

Continued on next page

Table 1 – continued from previous page

pixel_resolution_to_nside(resolution[, round])	Find closest HEALPix nside for a given angular resolution.
test([package, test_path, args, plugins, ...])	Run the tests using py.test.
uniq_to_level_ipix(uniq)	Convert a HEALPix cell uniq number to its (level, ipix) equivalent.

bilinear_interpolation_weights

astropy_healpix.**bilinear_interpolation_weights**(*lon*, *lat*, *nside*, *order='ring'*)

Get the four neighbours for each (lon, lat) position and the weight associated with each one for bilinear interpolation.

Parameters

lon, lat : Quantity

The longitude and latitude values as Quantity instances with angle units.

nside : int

Number of pixels along the side of each of the 12 top-level HEALPix tiles

order : { 'nested' | 'ring' }

Order of HEALPix pixels

Returns

indices : ndarray

2-D array with shape (4, N) giving the four indices to use for the interpolation

weights : ndarray

2-D array with shape (4, N) giving the four weights to use for the interpolation

healpix_to_lonlat

astropy_healpix.**healpix_to_lonlat**(*healpix_index*, *nside*, *dx=None*, *dy=None*, *order='ring'*)

Convert HEALPix indices (optionally with offsets) to longitudes/latitudes.

If no offsets (dx and dy) are provided, the coordinates will default to those at the center of the HEALPix pixels.

Parameters

healpix_index : int or ndarray

HEALPix indices (as a scalar or array)

nside : int

Number of pixels along the side of each of the 12 top-level HEALPix tiles

dx, dy : float or ndarray, optional

Offsets inside the HEALPix pixel, which must be in the range [0:1], where 0.5 is the center of the HEALPix pixels (as scalars or arrays)

order : { 'nested' | 'ring' }, optional

Order of HEALPix pixels

Returns

lon : Longitude

The longitude values

lat : Latitude

The latitude values

interpolate_bilinear_lonlat

astropy_healpix.**interpolate_bilinear_lonlat**(*lon*, *lat*, *values*, *order='ring'*)

Interpolate values at specific longitudes/latitudes using bilinear interpolation

Parameters

lon, lat : Quantity

The longitude and latitude values as Quantity instances with angle units.

values : ndarray

Array with the values in each HEALPix pixel. The first dimension should have length 12 * nside ** 2 (and nside is determined automatically from this).

order : { 'nested' | 'ring' }

Order of HEALPix pixels

Returns

result : float ndarray

The interpolated values

level_ipix_to_uniq

astropy_healpix.**level_ipix_to_uniq**(*level*, *ipix*)

Convert a level and HEALPix index into a uniq number representing the cell.

This function is the inverse of uniq_to_level_ipix.

Parameters

level : int

The level of the HEALPix cell

ipix : int

The index of the HEALPix cell

Returns

uniq : int

The uniq number representing the HEALPix cell.

level_to_nside

astropy_healpix.**level_to_nside**(*level*)

Find the pixel dimensions of the top-level HEALPix tiles.

This is given by `nside = 2**level`.

Parameters

level : int

The resolution level

Returns

nside : int

The number of pixels on the side of one of the 12 'top-level' HEALPix tiles.

lonlat_to_healpix

astropy_healpix.**lonlat_to_healpix**(*lon*, *lat*, *nside*, *return_offsets=False*, *order='ring'*)

Convert longitudes/latitudes to HEALPix indices

Parameters

lon, lat : `Quantity`

The longitude and latitude values as `Quantity` instances with angle units.

nside : int

Number of pixels along the side of each of the 12 top-level HEALPix tiles

order : { 'nested' | 'ring' }

Order of HEALPix pixels

return_offsets : bool, optional

If `True`, the returned values are the HEALPix pixel indices as well as `dx` and `dy`, the fractional positions inside the pixels. If `False` (the default), only the HEALPix pixel indices is returned.

Returns

healpix_index : int or `ndarray`

The HEALPix indices

dx, dy : `ndarray`

Offsets inside the HEALPix pixel in the range [0:1], where 0.5 is the center of the HEALPix pixels

neighbours

astropy_healpix.**neighbours**(*healpix_index*, *nside*, *order='ring'*)

Find all the HEALPix pixels that are the neighbours of a HEALPix pixel

Parameters

healpix_index : `ndarray`

Array of HEALPix pixels

nside : int

Number of pixels along the side of each of the 12 top-level HEALPix tiles

order : { 'nested' | 'ring' }

Order of HEALPix pixels

Returns

neigh : `ndarray`

Array giving the neighbours starting SW and rotating clockwise. This has one extra dimension compared to `healpix_index` - the first dimension - which is set to 8. For example if healpix_index has shape (2, 3), `neigh` has shape (8, 2, 3).

npix_to_nside

astropy_healpix.**npix_to_nside**(*npix*)

Find the number of pixels on the side of one of the 12 'top-level' HEALPix tiles given a total number of pixels.

Parameters

npix : int

The number of pixels in the HEALPix map.

Returns

nside : int

The number of pixels on the side of one of the 12 'top-level' HEALPix tiles.

nside_to_level

astropy_healpix.**nside_to_level**(*nside*)

Find the HEALPix level for a given nside.

This is given by `level = log2(nside)`.

This function is the inverse of `level_to_nside`.

Parameters

nside : int

The number of pixels on the side of one of the 12 'top-level' HEALPix tiles. Must be a power of two.

Returns

level : int

The level of the HEALPix cells

nside_to_npix

astropy_healpix.**nside_to_npix**(*nside*)

Find the number of pixels corresponding to a HEALPix resolution.

Parameters

nside : int

The number of pixels on the side of one of the 12 'top-level' HEALPix tiles.

Returns

npix : int

The number of pixels in the HEALPix map.

nside_to_pixel_area

astropy_healpix.**nside_to_pixel_area**(*nside*)

Find the area of HEALPix pixels given the pixel dimensions of one of the 12 'top-level' HEALPix tiles.

Parameters

nside : int

The number of pixels on the side of one of the 12 'top-level' HEALPix tiles.

Returns

pixel_area : Quantity

The area of the HEALPix pixels

nside_to_pixel_resolution

astropy_healpix.**nside_to_pixel_resolution**(*nside*)

Find the resolution of HEALPix pixels given the pixel dimensions of one of the 12 'top-level' HEALPix tiles.

Parameters

nside : int

The number of pixels on the side of one of the 12 'top-level' HEALPix tiles.

Returns

resolution : Quantity

The resolution of the HEALPix pixels

See also:

pixel_resolution_to_nside

pixel_resolution_to_nside

astropy_healpix.**pixel_resolution_to_nside**(*resolution*, *round='nearest'*)

Find closest HEALPix nside for a given angular resolution.

This function is the inverse of nside_to_pixel_resolution, for the default rounding scheme of `round='nearest'`.

If you choose `round='up'`, you'll get HEALPix pixels that have at least the requested resolution (usually a bit better due to rounding).

Pixel resolution is defined as square root of pixel area.

Parameters

resolution : Quantity

Angular resolution

round : {'up', 'nearest', 'down'}

Which way to round

Returns

nside : int

The number of pixels on the side of one of the 12 'top-level' HEALPix tiles. Always a power of 2.

Examples

```
>>> from astropy import units as u
>>> from astropy_healpix import pixel_resolution_to_nside
>>> pixel_resolution_to_nside(13 * u.arcmin)
256
>>> pixel_resolution_to_nside(13 * u.arcmin, round='up')
512
```

test

astropy_healpix.**test**(*package=None*, *test_path=None*, *args=None*, *plugins=None*, *verbose=False*, *pastebin=None*, *remote_data=False*, *pep8=False*, *pdb=False*, *coverage=False*, *open_files=False*, ***kwargs*)

Run the tests using py.test. A proper set of arguments is constructed and passed to pytest.main.

Parameters

package : str, optional

The name of a specific package to test, e.g. 'io.fits' or 'utils'. If nothing is specified all default tests are run.

test_path : str, optional

Specify location to test by path. May be a single file or directory. Must be specified absolutely or relative to the calling directory.

args : str, optional

Additional arguments to be passed to pytest.main in the `args` keyword argument.

plugins : list, optional

Plugins to be passed to pytest.main in the `plugins` keyword argument.

verbose : bool, optional

Convenience option to turn on verbose output from py.test. Passing True is the same as specifying `'-v'` in `args`.

pastebin : {'failed','all',None}, optional

Convenience option for turning on py.test pastebin output. Set to `'failed'` to upload info for failed tests, or `'all'` to upload info for all tests.

remote_data : bool, optional

Controls whether to run tests marked with @remote_data. These tests use online data and are not run by default. Set to True to run these tests.

pep8 : bool, optional

Turn on PEP8 checking via the pytest-pep8 plugin and disable normal tests. Same as specifying `'--pep8 -k pep8'` in `args`.

pdb : bool, optional

Turn on PDB post-mortem analysis for failing tests. Same as specifying `'--pdb'` in `args`.

coverage : bool, optional

Generate a test coverage report. The result will be placed in the directory htmlcov.

open_files : bool, optional

Fail when any tests leave files open. Off by default, because this adds extra run time to the test suite. Requires the psutil package.

parallel : int, optional

When provided, run the tests in parallel on the specified number of CPUs. If parallel is negative, it will use the all the cores on the machine. Requires the pytest-xdist plugin installed. Only available when using Astropy 0.3 or later.

kwargs

Any additional keywords passed into this function will be passed on to the astropy test runner. This allows use of test-related functionality implemented in later versions of astropy without explicitly updating the package template.

uniq_to_level_ipix

astropy_healpix.**uniq_to_level_ipix**(*uniq*)

Convert a HEALPix cell uniq number to its (level, ipix) equivalent.

A uniq number is a 64 bits integer equaling to : ipix + 4*(4**level). Please read this paper for more details about uniq numbers.

Parameters

uniq : int

The uniq number of a HEALPix cell.

Returns

level, ipix: int, int

The level and index of the HEALPix cell computed from `uniq`.

9.1.2 Classes

HEALPix([nside, order, frame])	A HEALPix pixellization.

HEALPix

class astropy_healpix.**HEALPix**(*nside=None*, *order='ring'*, *frame=None*)

Bases: object

A HEALPix pixellization.

Parameters

nside : int

Number of pixels along the side of each of the 12 top-level HEALPix tiles

order : { 'nested' | 'ring' }

Order of HEALPix pixels

frame : BaseCoordinateFrame, optional

The celestial coordinate frame of the pixellization. This can be ommitted, in which case the pixellization will not be attached to any particular celestial frame, and the methods ending in _skycoord will not work (but the _lonlat methods will still work and continue to return generic longitudes/latitudes).

Attributes Summary

npix	The number of pixels in the pixellization of the sphere.
pixel_area	The area of a single HEALPix pixel.
pixel_resolution	The resolution of a single HEALPix pixel.

Methods Summary

bilinear_interpolation_weights(lon, lat)	Get the four neighbours for each (lon, lat) position and the weight associated with each one for bilinear interpolation.
boundaries_lonlat(healpix_index, step)	Return the longitude and latitude of the edges of HEALPix pixels
boundaries_skycoord(healpix_index, step)	Return the celestial coordinates of the edges of HEALPix pixels
cone_search_lonlat(lon, lat, radius)	Find all the HEALPix pixels within a given radius of a longitude/latitude.
cone_search_skycoord(skycoord, radius)	Find all the HEALPix pixels within a given radius of a celestial position.
healpix_to_lonlat(healpix_index[, dx, dy])	Convert HEALPix indices (optionally with offsets) to longitudes/latitudes
healpix_to_skycoord(healpix_index[, dx, dy])	Convert HEALPix indices (optionally with offsets) to celestial coordinates.
interpolate_bilinear_lonlat(lon, lat, values)	Interpolate values at specific longitudes/latitudes using bilinear interpolation
interpolate_bilinear_skycoord(skycoord, values)	Interpolate values at specific celestial coordinates using bilinear interpolation.
lonlat_to_healpix(lon, lat[, return_offsets])	Convert longitudes/latitudes to HEALPix indices (optionally with offsets)
neighbours(healpix_index)	Find all the HEALPix pixels that are the neighbours of a HEALPix pixel
nested_to_ring(nested_index)	Convert a healpix 'nested' index to a healpix 'ring' index
ring_to_nested(ring_index)	Convert a healpix 'ring' index to a healpix 'nested' index
skycoord_to_healpix(skycoord[, return_offsets])	Convert celestial coordinates to HEALPix indices (optionally with offsets).

Attributes Documentation

npix
The number of pixels in the pixellization of the sphere.

pixel_area
The area of a single HEALPix pixel.

pixel_resolution
The resolution of a single HEALPix pixel.

Methods Documentation

bilinear_interpolation_weights(*lon*, *lat*)
Get the four neighbours for each (lon, lat) position and the weight associated with each one for bilinear interpolation.

Parameters
lon, lat : Quantity

The longitude and latitude values as Quantity instances with angle units.

Returns

indices : `ndarray`

2-D array with shape (4, N) giving the four indices to use for the interpolation

weights : `ndarray`

2-D array with shape (4, N) giving the four weights to use for the interpolation

boundaries_lonlat(*healpix_index*, *step*)

Return the longitude and latitude of the edges of HEALPix pixels

This returns the longitude and latitude of points along the edge of each HEALPIX pixel. The number of points returned for each pixel is `4 * step`, so setting `step` to 1 returns just the corners.

Parameters

healpix_index : `ndarray`

1-D array of HEALPix pixels

step : int

The number of steps to take along each edge.

Returns

lon, lat : `Quantity`

The longitude and latitude, as 2-D arrays where the first dimension is the same as the `healpix_index` input, and the second dimension has size `4 * step`.

boundaries_skycoord(*healpix_index*, *step*)

Return the celestial coordinates of the edges of HEALPix pixels

This returns the celestial coordinates of points along the edge of each HEALPIX pixel. The number of points returned for each pixel is `4 * step`, so setting `step` to 1 returns just the corners.

This method requires that a celestial frame was specified when initializing HEALPix. If you don't know or need the celestial frame, you can instead use `boundaries_lonlat()`.

Parameters

healpix_index : `ndarray`

1-D array of HEALPix pixels

step : int

The number of steps to take along each edge.

Returns

skycoord : `SkyCoord`

The celestial coordinates of the HEALPix pixel boundaries

cone_search_lonlat(*lon*, *lat*, *radius*)

Find all the HEALPix pixels within a given radius of a longitude/latitude.

Note that this returns all pixels that overlap, including partially, with the search cone. This function can only be used for a single lon/lat pair at a time, since different calls to the function may result in a different number of matches.

Parameters

lon, lat : `Quantity`

The longitude and latitude to search around

radius : `Quantity`

The search radius

Returns

healpix_index : `ndarray`

1-D array with all the matching HEALPix pixel indices.

cone_search_skycoord(*skycoord*, *radius*)

Find all the HEALPix pixels within a given radius of a celestial position.

Note that this returns all pixels that overlap, including partially, with the search cone. This function can only be used for a single celestial position at a time, since different calls to the function may result in a different number of matches.

This method requires that a celestial frame was specified when initializing HEALPix. If you don't know or need the celestial frame, you can instead use `cone_search_lonlat()`.

Parameters

skycoord : `SkyCoord`

The celestial coordinates to use for the cone search

radius : `Quantity`

The search radius

Returns

healpix_index : `ndarray`

1-D array with all the matching HEALPix pixel indices.

healpix_to_lonlat(*healpix_index*, *dx=None*, *dy=None*)

Convert HEALPix indices (optionally with offsets) to longitudes/latitudes

Parameters

healpix_index : `ndarray`

1-D array of HEALPix indices

dx, dy : `ndarray`, optional

1-D arrays of offsets inside the HEALPix pixel, which must be in the range [0:1] (0.5 is the center of the HEALPix pixels). If not specified, the position at the center of the pixel is used.

Returns

lon : `Longitude`

The longitude values

lat : `Latitude`

The latitude values

healpix_to_skycoord(*healpix_index*, *dx=None*, *dy=None*)

Convert HEALPix indices (optionally with offsets) to celestial coordinates.

Note that this method requires that a celestial frame was specified when initializing HEALPix. If you don't know or need the celestial frame, you can instead use `healpix_to_lonlat()`.

Parameters

healpix_index : `ndarray`

1-D array of HEALPix indices

dx, dy : `ndarray`, optional

1-D arrays of offsets inside the HEALPix pixel, which must be in the range [0:1] (0.5 is the center of the HEALPix pixels). If not specified, the position at the center of the pixel is used.

Returns

coord : `SkyCoord`

The resulting celestial coordinates

interpolate_bilinear_lonlat(*lon*, *lat*, *values*)

Interpolate values at specific longitudes/latitudes using bilinear interpolation

If a position does not have four neighbours, this currently returns NaN.

Parameters

lon, lat : `Quantity`

The longitude and latitude values as `Quantity` instances with angle units.

values : `ndarray`

1-D array with the values in each HEALPix pixel. This must have a length of the form 12 * nside ** 2 (and nside is determined automatically from this).

Returns

result : `ndarray`

1-D array of interpolated values

interpolate_bilinear_skycoord(*skycoord*, *values*)

Interpolate values at specific celestial coordinates using bilinear interpolation.

If a position does not have four neighbours, this currently returns NaN.

Note that this method requires that a celestial frame was specified when initializing HEALPix. If you don't know or need the celestial frame, you can instead use `interpolate_bilinear_lonlat()`.

Parameters

skycoord : `SkyCoord`

The celestial coordinates at which to interpolate

values : `ndarray`

1-D array with the values in each HEALPix pixel. This must have a length of the form 12 * nside ** 2 (and nside is determined automatically from this).

Returns

result : `ndarray`

1-D array of interpolated values

lonlat_to_healpix(*lon*, *lat*, *return_offsets=False*)

Convert longitudes/latitudes to HEALPix indices (optionally with offsets)

Parameters

lon, lat : `Quantity`

The longitude and latitude values as `Quantity` instances with angle units.

return_offsets : bool

If `True`, the returned values are the HEALPix pixel as well as `dx` and `dy`, the fractional positions inside the pixel. If `False` (the default), only the HEALPix pixel is returned.

Returns

healpix_index : ndarray

1-D array of HEALPix indices

dx, dy : ndarray

1-D arrays of offsets inside the HEALPix pixel in the range [0:1] (0.5 is the center of the HEALPix pixels). This is returned if `return_offsets` is `True`.

neighbours(*healpix_index*)

Find all the HEALPix pixels that are the neighbours of a HEALPix pixel

Parameters

healpix_index : ndarray

Array of HEALPix pixels

Returns

neigh : ndarray

Array giving the neighbours starting SW and rotating clockwise. This has one extra dimension compared to `healpix_index` - the first dimension - which is set to 8. For example if healpix_index has shape (2, 3), `neigh` has shape (8, 2, 3).

nested_to_ring(*nested_index*)

Convert a healpix 'nested' index to a healpix 'ring' index

Parameters

nested_index : ndarray

Healpix index using the 'nested' ordering

Returns

ring_index : ndarray

Healpix index using the 'ring' ordering

ring_to_nested(*ring_index*)

Convert a healpix 'ring' index to a healpix 'nested' index

Parameters

ring_index : ndarray

Healpix index using the 'ring' ordering

Returns

nested_index : ndarray

Healpix index using the 'nested' ordering

skycoord_to_healpix(*skycoord*, *return_offsets=False*)

Convert celestial coordinates to HEALPix indices (optionally with offsets).

Note that this method requires that a celestial frame was specified when initializing HEALPix. If you don't know or need the celestial frame, you can instead use `lonlat_to_healpix()`.

Parameters

skycoord : SkyCoord

The celestial coordinates to convert

return_offsets : bool

If `True`, the returned values are the HEALPix pixel as well as `dx` and `dy`, the fractional positions inside the pixel. If `False` (the default), only the HEALPix pixel is returned.

Returns

healpix_index : ndarray

1-D array of HEALPix indices

dx, dy : ndarray

1-D arrays of offsets inside the HEALPix pixel in the range [0:1] (0.5 is the center of the HEALPix pixels). This is returned if `return_offsets` is `True`.

9.2 astropy_healpix.healpy Module

9.2.1 Functions

nside2resol(nside[, arcmin])	Drop-in replacement for healpy nside2resol.
nside2pixarea(nside[, degrees])	Drop-in replacement for healpy nside2pixarea.
nside2npix(nside)	Drop-in replacement for healpy nside2npix.
npix2nside(npix)	Drop-in replacement for healpy npix2nside.
pix2ang(nside, ipix[, nest, lonlat])	Drop-in replacement for healpy pix2ang.
ang2pix(nside, theta, phi[, nest, lonlat])	Drop-in replacement for healpy ang2pix.
pix2vec(nside, ipix[, nest])	Drop-in replacement for healpy pix2vec.
vec2pix(nside, x, y, z[, nest])	Drop-in replacement for healpy vec2pix.
order2nside(order)	Drop-in replacement for healpy order2nside.
nest2ring(nside, ipix)	Drop-in replacement for healpy nest2ring.
ring2nest(nside, ipix)	Drop-in replacement for healpy ring2nest.
boundaries(nside, pix[, step, nest])	Drop-in replacement for healpy boundaries.
vec2ang(vectors[, lonlat])	Drop-in replacement for healpy vec2ang.
ang2vec(theta, phi[, lonlat])	Drop-in replacement for healpy ang2vec.
get_interp_weights(nside, theta[, phi, ...])	Drop-in replacement for healpy get_interp_weights.
get_interp_val(m, theta, phi[, nest, lonlat])	Drop-in replacement for healpy get_interp_val.

nside2resol

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

Drop-in replacement for healpy nside2resol.

nside2pixarea

astropy_healpix.healpy.**nside2pixarea**(*nside*, *degrees=False*)

Drop-in replacement for healpy nside2pixarea.

nside2npix

astropy_healpix.healpy.**nside2npix**(*nside*)

Drop-in replacement for healpy nside2npix.

npix2nside

astropy_healpix.healpy.**npix2nside**(*npix*)

Drop-in replacement for healpy npix2nside.

pix2ang

astropy_healpix.healpy.**pix2ang**(*nside*, *ipix*, *nest=False*, *lonlat=False*)
Drop-in replacement for healpy pix2ang.

ang2pix

astropy_healpix.healpy.**ang2pix**(*nside*, *theta*, *phi*, *nest=False*, *lonlat=False*)
Drop-in replacement for healpy ang2pix.

pix2vec

astropy_healpix.healpy.**pix2vec**(*nside*, *ipix*, *nest=False*)
Drop-in replacement for healpy pix2vec.

vec2pix

astropy_healpix.healpy.**vec2pix**(*nside*, *x*, *y*, *z*, *nest=False*)
Drop-in replacement for healpy vec2pix.

order2nside

astropy_healpix.healpy.**order2nside**(*order*)
Drop-in replacement for healpy order2nside.

nest2ring

astropy_healpix.healpy.**nest2ring**(*nside*, *ipix*)
Drop-in replacement for healpy nest2ring.

ring2nest

astropy_healpix.healpy.**ring2nest**(*nside*, *ipix*)
Drop-in replacement for healpy ring2nest.

boundaries

astropy_healpix.healpy.**boundaries**(*nside*, *pix*, *step=1*, *nest=False*)
Drop-in replacement for healpy boundaries.

vec2ang

astropy_healpix.healpy.**vec2ang**(*vectors*, *lonlat=False*)
Drop-in replacement for healpy vec2ang.

ang2vec

astropy_healpix.healpy.**ang2vec**(*theta*, *phi*, *lonlat=False*)
Drop-in replacement for healpy ang2vec.

get_interp_weights

astropy_healpix.healpy.**get_interp_weights**(*nside*, *theta*, *phi=None*, *nest=False*, *lonlat=False*)
Drop-in replacement for healpy get_interp_weights.

Although note that the order of the weights and pixels may differ.

get_interp_val

astropy_healpix.healpy.**get_interp_val**(*m*, *theta*, *phi*, *nest=False*, *lonlat=False*)
Drop-in replacement for healpy get_interp_val.

CHAPTER 10

Changes

10.1 0.3.1 (2018-10-24)

- Ensure .c files are included in tar file.

10.2 0.3 (2018-10-24)

- Remove OpenMP from astropy-healpix [#108]
- Fix bilinear interpolation of invalid values [#106]
- Add uniq to (level, ipix) and inverse function [#105]
- compute z more stably; improve on z2dec [#101]
- use more stable cos(Dec) term [#94]
- Fix get_interp_weights for phi=None case [#89]
- Add pix2vec, vec2pix, ang2vec [#73]
- Add `pixel_resolution_to_nside` function. [#31]

10.3 0.2 (2017-10-15)

- Expand benchmarks to include ang2pix, nest2ring and ring2nest. [#62]
- Use OpenMP to parallelize the Cython wrappers. [#59]
- Renamed the `healpix_neighbours` function to `neighbours` and added a wrapper to the high-level class. [#61]
- Fix bilinear interpolation which was being done incorrectly, and added a new `bilinear_interpolation_weights` function to get the interpolation weights. [#63]

10.4 0.1 (2017-10-01)

- Initial release

Python Module Index

a

Index

R

S

T

U

V