rHEALPixDGGS Documentation

Release 0.5.1

Alexander Raichev

July 29, 2013

CONTENTS

CHAPTER

ONE

INTRODUCTION

rHEALPixDGGS is a Python 3.3 package that implements the rHEALPix discrete global grid system (DGGS). This documentation assumes you are familiar with the rHEALPix DGGS as described in [GRS2013] and familiar with basic Python 3.3 usage as described in The Python Tutorial.

1.1 Requirements

- Python >=3.3
- NumPy >=1.7 Base N-dimensional array package
- SciPy >=0.12 Fundamental library for scientific computing
- Matplotlib >=1.2.1 Comprehensive 2D Plotting
- Pyproj >=1.9.3 Python interface to the PROJ.4 cartographic library
- Sage >=5.10 (Optional) Comprehensive mathematics package. Needed only for a few optional graphics methods. To use the optional graphics methods, start a Sage notebook session and import/attach the Python module that contains the methods. For examples, see the Sage worksheet `rhealpix_dggs/tests/test_rhealpix_dggs.sws`.

1.2 Installation

rHEALPixDGGS is available for download at Landcare Research's git repository http://code.scenzgrid.org/index.php/p/scenzgrid-py/source/tree/master/rHEALPixDGGS-0.5 and can be cloned via the command `git clone git@code.scenzgrid.org:scenzgrid-py.git`.

To install Sage, download and install the appropriate binary file from sagemath.org. Then install Pyproj within Sage by downloading the Pyproj source, changing to the Pyproj directory, starting a Sage shell via `sage -sh`, then typing `python setup.py build`, then `python setup.py install`. If that doesn't work, try again but first start in superuser mode via `sudo su`.

1.3 Usage

To use rHEALPixDGGS after installing it, start a Python session in the directory where you downloaded the modules and import the modules. Here are some examples. For a list of all methods available, see the application programming interface (API) in the following chapters.

1.3.1 Using the `projection_wrapper` Module

The module `projection_wrapper` implements a wrapper for map projections of ellipsoids of revolution (which include spheres but not general triaxial ellipsoids) defined in `pj_healpix`, `pj_rhealpix`, and Pyproj. For brevity hereafter, the word 'ellipsoid' abbreviates 'ellipsoid of revolution'.

Currently `projection_wrapper` uses the HEALPix and rHEALPix projections defined in `pj_healpix` and `pj_rhealpix` and *not* the buggy versions defined in Pyproj 1.9.3 as `PJ_healpix.c`. Alternatively, you can download a corrected version of `PJ_healpix.c` from trac.osgeo.org/proj/changeset/2378, rebuild Pyproj with it, and use it in `rhealpix_dggs` by editing the `HOMEMADE_PROJECTIONS` line in `projection_wrapper`.

Import all the classes, methods, and constants from the module:

```
>>> from projection_wrapper import *
```

Create an ellipsoid, say, one with major radius 5 and eccentricity 0.8:

```
>>> from ellipsoids import *
>>> ellps_1 = Ellipsoid(a=5, e=0.8)
>>> print(ellps_1)
ellipsoid:
    R_A = 4.32200117119
    a = 5
    b = 3.0
    e = 0.8
    f = 0.4
    lat_0 = 0
    lon_0 = 0
    radians = False
    sphere = False
```

The names of the ellipsoid attributes agree with the names of the PROJ.4 ellipsoid parameters.

By default, angles are measured in degrees. If you prefer radians, then do:

```
>>> ellps_2 = Ellipsoid(a=5, e=0.8, radians=True)
>>> print(ellps_2)
ellipsoid:
    R_A = 4.32200117119
    a = 5
    b = 3.0
    e = 0.8
    f = 0.4
    lat_0 = 0
    lon_0 = 0
    radians = True
    sphere = False
```

Some common ellipsoids are predefined as constants.

```
>>> print(UNIT_SPHERE)
ellipsoid:
    R = 1
    R_A = 1
    a = 1
    b = 1
    e = 0
    f = 0
    lat_0 = 0
    lon_0 = 0
    radians = False
    sphere = True
>>> print(WGS84_ELLIPSOID)
ellipsoid:
    R_A = 6374581.4671
    a = 6378137.0
    b = 6356752.31414
    e = 0.0578063088401
    f = 0.00335281068118
    lat_0 = 0
    lon_0 = 0
    radians = False
    sphere = False
>>> print(WGS84_ELLIPSOID_RADIANS)
ellipsoid:
    R_A = 6374581.4671
    a = 6378137.0
    b = 6356752.31414
    e = 0.0578063088401
    f = 0.00335281068118
    lat_0 = 0
    lon_0 = 0
    radians = True
    sphere = False
```

Ellipsoid instances are parametrized by geographic longitude and latitude with the central meridian at `lon_0` and the parallel of origin at `lat_0`.

Project some points of the ellipsoid using the HEALPix and rHEALPix projections:

```
>>> h = Proj(ellps_1, 'healpix')
>>> rh = Proj(ellps_1, 'rhealpix', north_square=1, south_square=2)
>>> print(h(0, 60))
(0.0, 3.351278550178026)
>>> print(rh(0, 60))
(0.0, 3.351278550178026)
>>> print(h(0, 70))
(0.864006732389895, 4.2584985144432679)
>>> print(rh(0, 70))
(-0.86400673238989445, 4.2584985144432679)
```

1.3.2 Using the `rhealpix_dggs` Module

The module `rhealpix_dggs` implements the rHEALPix DGGS and various operations thereupon.

Import all the classes, methods, and constants from the module

```
>>> from rhealpix_dggs import *
```

Create the (0, 0)-rHEALPix DGGS with N_side=3 that is based upon the WGS84 ellipsoid:

```
>>> from ellipsoids import WGS84_ELLIPSOID
>>> E = WGS84_ELLIPSOID
>>> rdggs = RHEALPixDGGS(ellipsoid=E, north_square=0, south_square=0, N_side=3)
>>> print(rdggs)
rHEALPix DGGS:
    N_side = 3
    north_square = 0
    south_square = 0
    max_areal_res = 1
    max_level = 15
    ellipsoid:
        R_A = 6374581.4671
        a = 6378137.0
        b = 6356752.31414
        e = 0.0578063088401
        f = 0.00335281068118
        lat_0 = 0
        lon_0 = 0
        radians = False
        sphere = False
```

Some common rHEALPix DGGSs are predefined as constants:

```
>>> print(UNIT_003)
rHEALPix DGGS:
    N_side = 3
    north_square = 0
    south_square = 0
    max_areal_resolution = 1
    max_resolution = 1
    ellipsoid:
        R = 1
        R_A = 1
        a = 1
        b = 1
        e = 0
        f = 0
        lat_0 = 0
        lon_0 = 0
        radians = False
        sphere = True
>>> print(WGS84_003)
rHEALPix DGGS:
    N_side = 3
    north_square = 0
    south_square = 0
    max_areal_resolution = 1
    max_resolution = 15
    ellipsoid:
        R_A = 6374581.4671
        a = 6378137.0
        b = 6356752.314140356
        e = 0.0578063088401
        f = 0.003352810681182319
        lat_0 = 0
        lon_0 = 0
        radians = False
        sphere = False
>>> print(UNIT_003_RADIANS)
rHEALPix DGGS:
    N_side = 3
    north_square = 0
    south_square = 0
    max_areal_resolution = 1
    max_resolution = 1
    ellipsoid:
        R = 1
        R_A = 1
        a = 1
        b = 1
        e = 0
        f = 0
        lat_0 = 0
        lon_0 = 0
        radians = True
        sphere = True
```

Pick a (longitude-latitude) point on the ellipsoid and find the level 1 cell that contains it

```
>>> p = (0, 15)
>>> c = rdggs.cell_from_point(1, p, plane=False); print(c)
Q0
```

Find the ellipsoidal (edge) neighbors of this cell

```
>>> for (direction, cell) in c.neighbors(plane=False).items():
...     print(direction, cell)
west P2
east Q1
north N2
south Q3
```

Find the planar (edge) neighbors of this cell

```
>>> for (direction, cell) in c.neighbors('plane').items():
...     print(direction, cell)
down Q3
right Q1
up N2
left P2
```

Find all the level 1 cells intersecting the longitude-latitude aligned ellipsoidal quadrangle with given northwest and southeast corners

```
>>> nw = (0, 45)
>>> se = (90, 0)
>>> cells = rdggs.cells_from_region(1, nw, se, plane=False)
>>> for row in cells:
...     print([str(cell) for cell in row])
['N2', 'N1', 'N0']
['Q0', 'Q1', 'Q2', 'R0']
['Q3', 'Q4', 'Q5', 'R3']
```

Compute the ellipsoidal shape and ellipsoidal nuclei of these cells

```
>>> for row in cells:
...     for cell in row:
...         print(cell, cell.ellipsoidal_shape(), cell.nucleus_and_vertices(plane=False)[0])
N2 dart (5.0888874903416268e-14, 58.470677829627355)
N1 skew_quad (45.000000000000036, 58.470677829627355)
N0 dart (89.999999999999957, 58.470677829627363)
Q0 quad (14.999999999999998, 26.438744923100096)
Q1 quad (45.0, 26.438744923100096)
Q2 quad (74.999999999999986, 26.438744923100096)
R0 quad (105.0, 26.438744923100096)
Q3 quad (14.999999999999998, 3.560649871414923e-15)
Q4 quad (45.0, 3.560649871414923e-15)
Q5 quad (74.999999999999986, 3.560649871414923e-15)
R3 quad (105.0, 3.560649871414923e-15)
```

Create the (0, 0)-rHEALPix DGGS with N_side = 3 that is based on the WGS84 ellipsoid. Orient the DGGS so that the planar origin (0, 0) is on Auckland, New Zealand:

```
>>> p = (174, -37)  # Approximate Auckland lon-lat coordinates
>>> from projection_wrapper import *
>>> E = Ellipsoid(a=WGS84_A, f=WGS84_F, radians=False, lon_0=p[0], lat_0=p[1])
>>> rdggs = RHEALPixDGGS(E, N_side=3, north_square=0, south_square=0)
>>> print(rdggs)
rHEALPix DGGS:
    N_side = 3
    north_square = 0
    south_square = 0
    max_areal_res = 1
    max_level = 15
    ellipsoid:
        R_A = 6374581.4671
        a = 6378137.0
        b = 6356752.31414
        e = 0.0578063088401
        f = 0.00335281068118
        lat_0 = -37
        lon_0 = 174
        radians = False
        sphere = False
>>> print(rdggs.cell_from_point(1, p, plane=False))
Q3
```

1.3.3 Using the `distortion` Module

The module `distortion` computes distortions of map projections created via `projection_wrapper` and was used to produce the tables in [GRS2013]. The module is not necessary for manipulating the rHEALPix DGGS.

Import all the classes, methods, and constants from the module:

```
>>> from distortion import *
```

Import the WGS84 ellipsoid, define the rHEALPix projection on the ellipsoid, and compute linear distortion and areal distortion for the projection at a given point:

```
>>> from ellipsoids import WGS84_ELLIPSOID
>>> E = WGS84_ELLIPSOID
>>> from projection_wrapper import Proj
>>> f = Proj(ellipsoid=E, proj='rhealpix', north_square=0, south_square=0)
>>> p = (0, 30)
>>> print(distortion(f, *p))   # First entry of output is f(*p))
((0.0, 3748655.115049511), 6.976075406932126, 1.1295629192757011, 1.1780969079183845)
```

Sample 100 random points from the WGS84 ellipsoid and compute the sample minimum, sample maximum, sample median, sample mean, and sample standard deviation of the linear and area distortion functions of the rHEALPix projection of those points:

```
>>> sample = [E.random_point() for i in range(100)]
>>> print(distortion_stats(f, sample)[1])
[[14.918, 13.82, 0.124, 48.086, 8.623], [1.345, 0.371, 1.002, 2.375, 1.163], [1.178, 0.0, 1.178,
```

Do the same for 100 points chosen at random from an rHEALPix DGGS dart cell:

```
>>> from rhealpix_dggs import *
>>> rdggs = WGS84_003
>>> cell = rdggs.cell(['N', 6])
>>> sample = [cell.random_point(plane=False) for i in range(100)]
>>> print(distortion_stats(f, sample)[1])
[[38.907, 6.044, 28.189, 48.963, 38.701], [2.013, 0.226, 1.644, 2.415, 1.991], [1.178, 0.0, 1.178
```

CHAPTER

TWO

THE UTILS MODULE

This Python 3.3 module implements several helper functions for coding map projections.

CHANGELOG:

- Alexander Raichev (AR), 2012-01-26: Refactored code from release 0.3.
- AR, 2013-07-23: Ported to Python 3.3.

NOTE:

All lengths are measured in meters and all angles are measured in radians unless indicated otherwise.

`utils.`**`auth_lat`**(*phi*, *e*, *inverse=False*, *radians=False*)

Given a point of geographic latitude *phi* on an ellipse of eccentricity *e*, return the authalic latitude of the point. If *inverse* =True, then compute its inverse approximately.

EXAMPLES:

```
>>> beta = auth_lat(pi/4, 0.5, radians=True)
>>> print(my_round(beta, 15))
0.689518212435
>>> print(my_round(auth_lat(beta, 0.5, radians=True, inverse=True), 15))
0.785126523581
>>> print(my_round(pi/4, 15))
0.785398163397448
```

NOTES:

The power series approximation used for the inverse is standard in cartography (PROJ.4 uses it, for instance) and accurate for small eccentricities.

`utils.`**`auth_rad`**(*a*, *e*, *inverse=False*)

Return the radius of the authalic sphere of the ellipsoid with major radius *a* and eccentricity *e*. If *inverse* = True, then return the major radius of the ellipsoid with authalic radius *a* and eccentricity *e*.

EXAMPLES:

```
>>> auth_rad(1, 0)
1
>>> for i in range(2, 11):
...     e = 1.0/i**2
...     print(my_round((e, auth_rad(1, 1.0/i**2)), 15))
(0.25, 0.98939325967009495)
(0.111111111111111, 0.99793514742994305)
(0.0625, 0.99934823645582505)
(0.04, 0.99973321235361001)
(0.027777777777778, 0.99987137105187995)
(0.020408163265306, 0.99993057628561399)
(0.015625, 0.99995930708084702)
(0.012345679012346, 0.99997459627121099)
(0.01, 0.99998333286108898)
```

`utils.`**`my_round`**`(`*x*, *digits=0*`)`

Round the floating point number or list/tuple of floating point numbers to `digits` number of digits. Calls Python's `round()` function.

EXAMPLES:

```
>>> print(my_round(1./7, 6))
0.142857
>>> print(my_round((1./3, 1./7), 6))
(0.333333, 0.142857)
```

`utils.`**`wrap_latitude`**`(`*phi*, *radians=False*`)`

Given a point p on the unit circle at angle *phi* from the positive x-axis, if p lies in the right half of the circle, then return its angle that lies in the interval [-pi/2, pi/2]. If p lies in the left half of the circle, then reflect it through the origin, and return the angle of the reflected point that lies in the interval [-pi/2, pi/2]. If *radians* = True, then *phi* and the output are given in radians. Otherwise, they are given in degrees.

EXAMPLES:

```
>>> wrap_latitude(45, radians=False)
45.0
>>> wrap_latitude(-45, radians=False)
-45.0
>>> wrap_latitude(90, radians=False)
90.0
>>> wrap_latitude(-90, radians=False)
-90.0
>>> wrap_latitude(135, radians=False)
-45.0
>>> wrap_latitude(-135, radians=False)
45.0
```

`utils.`**`wrap_longitude`**`(`*lam*, *radians=False*`)`

Given a point p on the unit circle at angle *lam* from the positive x-axis, return its angle theta in the range -pi <= theta < pi. If *radians* = True, then *lam* and the output are given in radians. Otherwise, they are given in degrees.

EXAMPLES:

```
>>> wrap_longitude(2*pi + pi, radians=True)
-3.1415926535897931
>>> wrap_longitude(-185, radians=False)
175.0
>>> wrap_longitude(-180, radians=False)
-180.0
>>> wrap_longitude(185, radians=False)
-175.0
```

CHAPTER

THREE

THE PJ_HEALPIX MODULE

This Python 3.3 module implements the HEALPix map projection as described in [CaRo2007].

CHANGELOG:

- Alexander Raichev (AR), 2013-01-26: Refactored code from release 0.3.
- AR, 2013-03-05: In in_healpix_image() increased eps to 1e-10 to decrease out-of-bounds errors i was getting when drawing figures.
- AR, 2013-07-23: Ported to Python 3.3.

NOTE:

All lengths are measured in meters and all angles are measured in radians unless indicated otherwise. By 'ellipsoid' below, i mean an oblate ellipsoid of revolution.

`pj_healpix.`**`healpix`**`(`*a=1*, *e=0*`)`

Return a function object that wraps the HEALPix projection and its inverse of an ellipsoid with major radius *a* and eccentricity *e*.

EXAMPLES:

```
>>> f = healpix(a=2, e=0)
>>> print(my_round(f(0, pi/3, radians=True), 15))
(0.57495135977821499, 2.1457476865731109)
>>> p = (0, 60)
>>> q = f(*p, radians=False); print(my_round(q, 15))
(0.57495135977821499, 2.1457476865731109)
>>> print(my_round(f(*q, radians=False, inverse=True), 15))
(5.9999999999999997e-15, 59.999999999999986)
>>> print(my_round(p, 15))
(0, 60)
```

OUTPUT:

- A function object of the form f(u, v, radians=False, inverse=False).

`pj_healpix.`**`healpix_diagram`**`(`*a=1*, *e=0*, *shade_polar_region=True*`)`

Return a Sage Graphics object diagramming the HEALPix projection boundary and polar triangles for the ellipsoid with major radius *a* and eccentricity *e*. Inessential graphics method. Requires Sage graphics methods.

`pj_healpix.`**`healpix_ellipsoid`**`(`*lam*, *phi*, *e=0*`)`

Compute the signature functions of the HEALPix projection of an oblate ellipsoid with eccentricity *e* whose authalic sphere is the unit sphere. Works when *e* = 0 (spherical case) too.

INPUT:

- *lam, phi* - Geodetic longitude-latitude coordinates in radians. Assume -pi <= *lam* < pi and -pi/2 <= *phi* <= pi/2.
- *e* - Eccentricity of the oblate ellipsoid.

EXAMPLES:

```
>>> print(my_round(healpix_ellipsoid(0, pi/7), 15))
(0, 0.511157237746422)
>>> print(my_round(healpix_ellipsoid(0, pi/7, e=0.8), 15))
(0, 0.268484450857837)
```

pj_healpix.**healpix_ellipsoid_inverse**(*x*, *y*, *e=0*)
Compute the inverse of healpix_ellipsoid().

EXAMPLES:

```
>>> p = (0, pi/7)
>>> q = healpix_ellipsoid(*p)
>>> print(my_round(healpix_ellipsoid_inverse(*q), 15))
(0, 0.448798950512828)
>>> print(my_round(p, 15))
(0, 0.448798950512828)
```

pj_healpix.**healpix_sphere**(*lam*, *phi*)
Compute the signature function of the HEALPix projection of the unit sphere.

INPUT:

- *lam, phi* - Geodetic longitude-latitude coordinates in radians. Assume -pi <= *lam* < pi and -pi/2 <= *phi* <= pi/2.

EXAMPLES:

```
>>> print(healpix_sphere(0, arcsin(2.0/3)) == (0, pi/4))
True
```

pj_healpix.**healpix_sphere_inverse**(*x*, *y*)
Compute the inverse of the healpix_sphere().

INPUT:

- *x, y* - Planar coordinates in meters in the image of the HEALPix projection of the unit sphere.

EXAMPLES:

```
>>> print(healpix_sphere_inverse(0, pi/4) == (0, arcsin(2.0/3)))
True
```

pj_healpix.**healpix_vertices**()
Return a list of the planar vertices of the HEALPix projection of the unit sphere.

pj_healpix.**in_healpix_image**(*x*, *y*)
Return True if and only if *(x, y)* lies in the image of the HEALPix projection of the unit sphere.

EXAMPLES:

```
>>> eps = 0     # Test boundary points.
>>> hp = [
... (-pi - eps, pi/4),
... (-3*pi/4, pi/2 + eps),
... (-pi/2, pi/4 + eps),
... (-pi/4, pi/2 + eps),
... (0, pi/4 + eps),
... (pi/4, pi/2 + eps),
... (pi/2, pi/4 + eps),
... (3*pi/4, pi/2 + eps),
... (pi + eps, pi/4),
... (pi + eps,-pi/4),
... (3*pi/4,-pi/2 - eps),
... (pi/2,-pi/4 - eps),
... (pi/4,-pi/2 - eps),
... (0,-pi/4 - eps),
... (-pi/4,-pi/2 - eps),
... (-pi/2,-pi/4 - eps),
... (-3*pi/4,-pi/2 - eps),
... (-pi - eps,-pi/4)
... ]
>>> for p in hp:
...     if not in_healpix_image(*p):
...             print('Fail')
...
>>> in_healpix_image(0, 0)
True
>>> in_healpix_image(0, pi/4 + 0.1)
False
```

CHAPTER

FOUR

THE PJ_RHEALPIX MODULE

This Python 3.3 module implements the rHEALPix map projection.

CHANGELOG:

- Alexander Raichev (AR), 2013-01-26: Refactored code from release 0.3.
- AR, 2013-07-23: Ported to Python 3.3.

NOTE:

All lengths are measured in meters and all angles are measured in radians unless indicated otherwise. By 'ellipsoid' below, i mean an oblate ellipsoid of revolution.

`pj_rhealpix.`**`combine_triangles`**(*x*, *y*, *north_square=0*, *south_square=0*, *inverse=False*)

Rearrange point *(x, y)* in the HEALPix projection by combining the polar triangles into two polar squares. Put the north polar square in position *north_square* and the south polar square in position *south_square*. If *inverse* = True, uncombine the polar triangles.

INPUT:

- *x, y* - Coordinates in the HEALPix projection of the unit sphere.
- *north_square, south_square* - Integers between 0 and 3 indicating the positions of the north_square polar square and south_square polar square respectively. See rhealpix_sphere() docstring for a diagram.
- *inverse* - (Optional; default = False) Boolean. If False, then compute forward function. If True, then compute inverse function.

EXAMPLES:

```
>>> u, v = -pi/4, pi/3
>>> x, y = combine_triangles(u, v)
>>> print(my_round((x, y), 15))
(-1.8325957145940459, 1.5707963267948959)
>>> print(my_round(combine_triangles(x, y, inverse=True), 15))
(-0.78539816339744795, 1.0471975511965981)
>>> print(my_round((u, v), 15))
(-0.785398163397448, 1.047197551196598)
```

`pj_rhealpix.`**`in_rhealpix_image`**(*x*, *y*, *north_square=0*, *south_square=0*)

Return True if and only if the point *(x, y)* lies in the image of the rHEALPix projection of the unit sphere.

EXAMPLES:

```
>>> eps = 0      # Test boundary points.
>>> north_square, south_square = 0, 0
>>> rhp = [
... (-pi - eps, pi/4 + eps),
... (-pi + north_square*pi/2 - eps, pi/4 + eps),
... (-pi + north_square*pi/2 - eps, 3*pi/4 + eps),
... (-pi + (north_square + 1)*pi/2 + eps, 3*pi/4 + eps),
... (-pi + (north_square + 1)*pi/2 + eps, pi/4 + eps),
... (pi + eps, pi/4 + eps),
... (pi + eps,-pi/4 - eps),
... (-pi + (south_square + 1)*pi/2 + eps,-pi/4 - eps),
... (-pi + (south_square + 1)*pi/2 + eps,-3*pi/4 - eps),
... (-pi + south_square*pi/2 - eps,-3*pi/4 - eps),
... (-pi + south_square*pi/2 -eps,-pi/4 - eps),
... (-pi - eps,-pi/4 - eps)
... ]
>>> for p in rhp:
...     if not in_rhealpix_image(*p):
...             print('Fail')
...
>>> print(in_rhealpix_image(0, 0))
True
>>> print(in_rhealpix_image(0, pi/4 + 0.1))
False
```

pj_rhealpix.**rhealpix**(*a=1*, *e=0*, *north_square=0*, *south_square=0*)

Return a function object that wraps the rHEALPix projection and its inverse of an ellipsoid with major radius *a* and eccentricity *e*.

EXAMPLES:

```
>>> f = rhealpix(a=2, e=0, north_square=1, south_square=2)
>>> print(my_round(f(0, pi/3, radians=True), 15))
(-0.57495135977821499, 2.1457476865731109)
>>> p = (0, 60)
>>> q = f(*p, radians=False)
>>> print(my_round(q, 15))
(-0.57495135977821499, 2.1457476865731109)
>>> print(my_round(f(*q, radians=False, inverse=True), 15))
(5.9999999999999997e-15, 59.999999999999986)
>>> print(my_round(p, 15))
(0, 60)
```

OUTPUT:

- A function object of the form f(u, v, radians=False, inverse=False).

pj_rhealpix.**rhealpix_diagram**(*a=1*, *e=0*, *north_square=0*, *south_square=0*, *shade_polar_region=True*)

Return a Sage Graphics object diagramming the rHEALPix projection boundary and polar triangles for the ellipsoid with major radius *a* and eccentricity *e*. Inessential graphics method. Requires Sage graphics methods.

pj_rhealpix.**rhealpix_ellipsoid**(*lam*, *phi*, *e=0*, *north_square=0*, *south_square=0*)

Compute the signature functions of the rHEALPix map projection of an oblate ellipsoid with eccentricity *e* whose authalic sphere is the unit sphere. The north polar square is put in position *north_square*, and the south polar square is put in position *south_square*. Works when $e = 0$ (spherical case) too.

INPUT:

- *lam, phi* - Geographic longitude-latitude coordinates in radian. Assume -pi <= *lam* < pi and -pi/2 <= *phi* <= pi/2.
- *e* - Eccentricity of the ellipsoid.
- *north_square, south_square* - (Optional; defaults = 0, 0) Integers between 0 and 3 indicating positions of north polar and south polar squares, respectively. See rhealpix_sphere() docstring for a diagram.

EXAMPLES:

```
>>> from numpy import arcsin
>>> print(my_round(rhealpix_ellipsoid(0, arcsin(2.0/3)), 15))
(0, 0.78539816339744795)
```

pj_rhealpix.**rhealpix_ellipsoid_inverse**(*x*, *y*, *e=0*, *north_square=0*, *south_square=0*)

Compute the inverse of rhealpix_ellipsoid.

EXAMPLES:

```
>>> p = (0, pi/4)
>>> q = rhealpix_ellipsoid(*p)
>>> print(my_round(rhealpix_ellipsoid_inverse(*q), 15))
(0.0, 0.78539816339744795)
>>> print(my_round(p, 15))
(0, 0.785398163397448)
```

pj_rhealpix.**rhealpix_sphere**(*lam*, *phi*, *north_square=0*, *south_square=0*)

Compute the signature functions of the rHEALPix map projection of the unit sphere. The north polar square is put in position *north_square*, and the south polar square is put in position *south_square*.

INPUT:

- *lam, phi* -Geographic longitude-latitude coordinates in radians. Assume -pi <= *lam* < pi and -pi/2 <= *phi* <= pi/2.
- *north_square, south_square* - (Optional; defaults = 0, 0) Integers between 0 and 3 indicating positions of north polar and south polar squares, respectively.

EXAMPLES:

```
>>> print(my_round(rhealpix_sphere(0, pi/4), 15))
(-1.619978633413937, 2.307012183573304)
```

NOTE:

The polar squares are labeled 0, 1, 2, 3 from east to west like this:

```
east          west
*---*---*---*---*
| 0 | 1 | 2 | 3 |
*---*---*---*---*
|   |   |   |   |
*---*---*---*---*
| 0 | 1 | 2 | 3 |
*---*---*---*---*
```

pj_rhealpix.**rhealpix_sphere_inverse**(*x*, *y*, *north_square=0*, *south_square=0*)

Compute the inverse of rhealpix_sphere().

EXAMPLES:

```
>>> p = (0, pi/4)
>>> q = rhealpix_sphere(*p)
>>> print(my_round(rhealpix_sphere_inverse(*q), 15))
(0.0, 0.78539816339744795)
>>> print(my_round(p, 15))
(0, 0.785398163397448)
```

pj_rhealpix.**rhealpix_vertices**(*north_square=0*, *south_square=0*)

Return a list of the planar vertices of the rHEALPix projection of the unit sphere.

pj_rhealpix.**triangle**(*x*, *y*, *north_square=0*, *south_square=0*, *inverse=False*)

Return the number of the polar triangle and region that *(x, y)* lies in. If *inverse* = False, then assume *(x,y)* lies in the image of the HEALPix projection of the unit sphere. If *inverse* = True, then assume *(x,y)* lies in the image of the *(north_square, south_square)*-rHEALPix projection of the unit sphere.

INPUT:

- *x, y* - Coordinates in the HEALPix or rHEALPix (if *inverse* = True) projection of the unit sphere.
- *north_square, south_square* - Integers between 0 and 3 indicating the positions of the north_square pole square and south_square pole square respectively. See rhealpix_sphere() docstring for a diagram.

- *inverse* - (Optional; default = False) Boolean. If False, then compute forward function. If True, then compute inverse function.

OUTPUT:

The pair (triangle_number, region). Here region equals 'north_polar' (polar), 'south_polar' (polar), or 'equatorial', indicating where *(x, y)* lies. If region = 'equatorial', then triangle_number = None. Suppose now that region != 'equatorial'. If *inverse* = False, then triangle_number is the number (0, 1, 2, or 3) of the HEALPix polar triangle Z that *(x, y)* lies in. If *inverse* = True, then triangle_number is the number (0, 1, 2, or 3) of the HEALPix polar triangle that *(x, y)* will get moved into.

EXAMPLES:

```
>>> triangle(-pi/4, pi/4 + 0.1)
(1, 'north_polar')
>>> triangle(-3*pi/4 + 0.1, pi/2, inverse=True)
(1, 'north_polar')
```

NOTES:

In the HEALPix projection, the polar triangles are labeled 0–3 from east to west like this:

```
    *       *       *       *
  * 0 *   * 1 *   * 2 *   * 3 *
*-------*-------*-------*-------*
|       |       |       |       |
|       |       |       |       |
|       |       |       |       |
*-------*-------*-------*-------*
  * 0 *   * 1 *   * 2 *   * 3 *
    *       *       *       *
```

In the rHEALPix projection these polar triangles get rearranged into a square with the triangles numbered *north_square* and *south_square* remaining fixed. For example, if *north_square* = 1 and *south_square* = 3, then the triangles get rearranged this way:

```
North polar square:     *-------*
                        | * 3 * |
                        | 0 * 2 |
                        | * 1 * |
                    ----*-------*----

South polar square: ----*-------*----
                        | * 3 * |
                        | 2 * 0 |
                        | * 1 * |
                        *-------*
```

CHAPTER

FIVE

THE ELLIPSOIDS MODULE

This Python 3.3 code implements ellipsoids of revolution.

CHANGELOG:

- Alexander Raichev (AR), 2012-01-26: Refactored code from release 0.3.
- AR, 2013-07-23: Ported to Python 3.3.

NOTE:

All lengths are measured in meters and all angles are measured in radians unless indicated otherwise.

By 'ellipsoid' throughout, i mean an ellipsoid of revolution and *not* a general (triaxial) ellipsoid. Points lying on an ellipsoid are given in geodetic (longitude, latitude) coordinates.

class `ellipsoids.`**`Ellipsoid`**(*R=None*, *a=6378137.0*, *b=None*, *e=None*, *f=0.003352810681182319*, *lon_0=0*, *lat_0=0*, *radians=False*)

Bases: `object`

Represents an ellipsoid of revolution (possibly a sphere) with a geodetic longitude-latitude coordinate frame.

INSTANCE ATTRIBUTES:

- *sphere* - True if the ellipsoid is a sphere, and False otherwise.
- *R* - The radius of the ellipsoid in meters, implying that it is a sphere.
- *a* - Major radius of the ellipsoid in meters.
- *b* - Minor radius of the ellipsoid in meters.
- *e* - Eccentricity of the ellipsoid.
- *f* - Flattening of the ellipsoid.
- *R_A* - Authalic radius of the ellipsoid in meters.
- *lon_0* - Central meridian.
- *lat_0* - Latitude of origin.
- *radians* - If True, use angles measured in radians for all calculations. Use degrees otherwise.
- *phi_0* - The latitude separating the equatorial region and the north polar region in the context of the (r)HEALPix projection.

Except for phi_0, these attribute names match the names of the PROJ.4 ellipsoid parameters.

`get_points`(*filename*)

Return a list of longitude-latitude points contained in the file with filename *filename*. Assume the file is a text file containing at most one longitude-latitude point per line with the coordinates separated by whitespace and angles given in degrees.

`graticule`(*n=400*, *spacing=None*)

Return a list of longitude-latitude points sampled from a longitude-latitude graticule on this ellipsoid with the given spacing between meridians and between parallels. The number of points on longitude and latitude per pi radians is *n*. The spacing should be specified in the angle units used for this ellipsoid. If *spacing=None*, then a default spacing of pi/16 radians will be set.

EXAMPLES:

```
>>> E = UNIT_SPHERE
>>> print(len(E.graticule(n=400)))
25600
```

lattice (*n=90*)

Return a 2n x n square lattice of longitude-latitude points.

EXAMPLES:

```
>>> E = UNIT_SPHERE
>>> for p in E.lattice(n=3):
...     print(p)
(-150.0, -60.0)
(-150.0, 0.0)
(-150.0, 60.0)
(-90.0, -60.0)
(-90.0, 0.0)
(-90.0, 60.0)
(-30.0, -60.0)
(-30.0, 0.0)
(-30.0, 60.0)
(30.0, -60.0)
(30.0, 0.0)
(30.0, 60.0)
(90.0, -60.0)
(90.0, 0.0)
(90.0, 60.0)
(150.0, -60.0)
(150.0, 0.0)
(150.0, 60.0)
```

meridian (*lam*, *n=200*)

Return a list of *n* equispaced longitude-latitude points lying along the meridian of longitude *lam*. Avoid the poles.

parallel (*phi*, *n=200*)

Return a list of *2*n* equispaced longitude-latitude points lying along the parallel of latitude *phi*.

pi ()

Return pi if *self.radians* = True and 180 otherwise.

random_point (*lam_min=None*, *lam_max=None*, *phi_min=None*, *phi_max=None*)

Return a point (given in geodetic coordinates) sampled uniformly at random from the section of this ellipsoid with longitude in the range *lam_min* <= *lam* < *lam_max* and latitude in the range *phi_min* <= *phi* < *phi_max*. But avoid the poles.

EXAMPLES:

```
>>> E = UNIT_SPHERE
>>> print(E.random_point())
(-1.0999574573422948, 0.21029104897701129)
```

xyz (*lam*, *phi*)

Given a point on this ellipsoid with longitude-latitude coordinates *(lam, phi)*, return the point's 3D rectangular coordinates.

EXAMPLES:

```
>>> E = UNIT_SPHERE
>>> print(my_round(E.xyz(0, 45), 15))
(0.70710678118654802, 0.0, 0.70710678118654802)
```

CHAPTER

SIX

THE PROJECTION_WRAPPER MODULE

This Python 3.3 module implements a wrapper for map projections.

CHANGELOG:

- Alexander Raichev (AR), 2013-01-25: Refactored code from release 0.3.
- AR, 2013-07-23: Ported to Python 3.3.

NOTE:

All lengths are measured in meters and all angles are measured in radians unless indicated otherwise. By 'ellipsoid' below, i mean an oblate ellipsoid of revolution.

class `projection_wrapper.`**`Proj`**(*ellipsoid=<ellipsoids.Ellipsoid object at 0x102425610>, proj=None, **kwargs*)

Bases: `object`

Represents a map projection of a given ellipsoid.

INSTANCE ATTRIBUTES:

- *ellipsoid* - An ellipsoid (Ellipsoid instance) to project.
- *proj* - The name (string) of the map projection, either a valid PROJ.4 projection name or a valid homemade projection name.
- *kwargs* - Keyword arguments (dictionary) needed for the projection's definition, but not for the definition of the ellipsoid. For example, these could be {'north_square':1, 'south_square': 2} for the rhealpix projection.

EXAMPLES:

```
>>> from ellipsoids import WGS84_ELLIPSOID
>>> f = Proj(ellipsoid=WGS84_ELLIPSOID, proj='rhealpix', north_square=1, south_square=0)
>>> print(my_round(f(0, 30), 15))
(0.0, 3748655.1150495014)
>>> f = Proj(ellipsoid=WGS84_ELLIPSOID, proj='cea')
>>> print(my_round(f(0, 30), 15))
(0.0, 3180183.485774971)
```

NOTES:

When accessing a homemade map projection assume that it can be called via a function g(a, e), where a is the major radius of the ellipsoid to be projected and e is its eccentricity. The output of g should be a function object of the form f(u, v, radians=False, inverse=False). For example, see the healpix() function in `pj_healpix.py`.

CHAPTER

SEVEN

THE RHEALPIX_DGGS MODULE

This Python 3.3 module implements the rHEALPix discrete global grid system.

CHANGELOG:

- Alexander Raichev (AR), 2012-11-12: Initial version based upon grids.py.
- AR, 2012-12-10: Corrected centroid() and moved some methods from graphics.py to here.
- AR, 2012-12-19: Tested all the methods and added examples.
- AR, 2013-01-01: Added ellipsoidal functionality to neighbor() and neighbors().
- AR, 2013-01-14: Added intersects_meridian(), cell_latitudes(), cells_from_meridian(), cells_from_parallel(), cells_from_region().
- AR, 2013-01-16: Changed the string keyword 'surface' to a boolean keyword 'plane'.
- AR, 2013-03-11: Added minimal_cover(), boundary(), interior(), triangle(), nw_vertex().
- AR, 2013-03-14: Fixed bug in nw_vertex().
- AR, 2013-07-23: Ported to Python 3.3.

NOTES:

All lengths are measured in meters and all angles are measured in radians unless indicated otherwise.

By 'ellipsoid' throughout, i mean an ellipsoid of revolution and *not* a general (triaxial) ellipsoid.

Points lying on the plane are given in rectangular (horizontal, vertical) coordinates, and points lying on the ellipsoid are given in geodetic (longitude, latitude) coordinates unless indicated otherwise.

DGGS abbreviates 'discrete global grid system'.

Except when manipulating positive integers, I avoid the modulo function '%' and insted write everything in terms of 'floor()'. This is because Python interprets the sign of '%' differently than Java or C, and I don't want to confuse people who are translating this code to those languages.

EXAMPLES:

Create the (1, 2)-rHEALPix DGGS with N_side = 3 that is based on the WGS84 ellipsoid. Use degrees instead of the default radians for angular measurements

```
>>> from ellipsoids import WGS84_ELLIPSOID
>>> E = WGS84_ELLIPSOID
>>> rdggs = RHEALPixDGGS(ellipsoid=E, north_square=1, south_square=2, N_side=3)
>>> print(rdggs)
rHEALPix DGGS:
    N_side = 3
    north_square = 1
    south_square = 2
    max_areal_resolution = 1
    max_resolution = 15
    ellipsoid:
        R_A = 6374581.4671
        a = 6378137.0
        b = 6356752.314140356
        e = 0.0578063088401
        f = 0.003352810681182319
        lat_0 = 0
        lon_0 = 0
        radians = False
        sphere = False
```

Pick a (longitude-latitude) point on the ellipsoid and find the resolution 1 cell that contains it

```
>>> p = (0, 45)
>>> c = rdggs.cell_from_point(1, p, plane=False); print(c)
N8
```

Find the ellipsoidal (edge) neighbors of this cell

```
>>> for (direction, cell) in sorted(c.neighbors(plane=False).items()):
...     print(direction, cell)
east N5
south_east Q0
south_west P2
west N7
```

Find the planar (edge) neighbors of this cell

```
>>> for (direction, cell) in sorted(c.neighbors('plane').items()):
...     print(direction, cell)
down P2
left N7
right Q0
up N5
```

Find all the resolution 1 cells intersecting the longitude-latitude aligned ellipsoidal quadrangle with given northwest and southeast corners

```
>>> nw = (0, 45)
>>> se = (90, 0)
>>> cells = rdggs.cells_from_region(1, nw, se, plane=False)
>>> for row in cells:
...     print([str(cell) for cell in row])
['N8', 'N5', 'N2']
['Q0', 'Q1', 'Q2', 'R0']
['Q3', 'Q4', 'Q5', 'R3']
```

Compute the ellipsoidal nuclei of these cells

```
>>> for row in cells:
...     for cell in row:
...         print(cell, cell.nucleus(plane=False))
N8 (0.0, 58.470677829627363)
N5 (45.000000000000036, 58.470677829627363)
N2 (90.000000000000028, 58.470677829627355)
Q0 (14.999999999999998, 26.438744923100096)
Q1 (45.0, 26.438744923100096)
Q2 (74.999999999999986, 26.438744923100096)
R0 (105.00000000000001, 26.438744923100096)
Q3 (14.999999999999998, 3.560649871414923e-15)
Q4 (45.0, 3.560649871414923e-15)
Q5 (74.999999999999986, 3.560649871414923e-15)
R3 (105.00000000000001, 3.560649871414923e-15)
```

Create the (0, 0)-rHEALPix DGGS with N_side = 3 that is based on the WGS84 ellipsoid. Use degrees instead of the default radians for angular measurements and orient the DGGS so that the planar origin (0, 0) is on Auckland,

New Zealand

```
>>> p = (174, -37)  # Approximate Auckland lon-lat coordinates
>>> from ellipsoids import *
>>> E = Ellipsoid(a=WGS84_A, f=WGS84_F, radians=False, lon_0=p[0], lat_0=p[1])
>>> rdggs = RHEALPixDGGS(E, N_side=3, north_square=0, south_square=0)
>>> print(rdggs)
rHEALPix DGGS:
    N_side = 3
    north_square = 0
    south_square = 0
    max_areal_resolution = 1
    max_resolution = 15
    ellipsoid:
        R_A = 6374581.4671
        a = 6378137.0
        b = 6356752.314140356
        e = 0.0578063088401
        f = 0.003352810681182319
        lat_0 = -37
        lon_0 = 174
        radians = False
        sphere = False
>>> print(rdggs.cell_from_point(1, p, plane=False))
Q3
```

class `rhealpix_dggs.`**`Cell`**(*rdggs=<rhealpix_dggs.RHEALPixDGGS object at 0x103dab990>, suid=None, level_order_index=None, post_order_index=None*)

Bases: `object`

Represents a cell of the planar or ellipsoidal rHEALPix grid hierarchies. Cell identifiers are of the form (p_0, p_1,...,p_l), where p_0 is one of the characters 'A', 'B', 'C', 'D', 'E', 'F' and p_i for i > 0 is one of the integers 0, 1,..., N_side**2 - 1, where N_side is the instance attribute from RHEALPixDGGS (the number of children cells along a cell's side).

INSTANCE ATTRIBUTES:

- *rdggs* - The DGGS that the cell comes from.
- *ellipsoid* - The underlying ellipsoid of the DGGS.
- *N_side* - The N_side attribute of the DGGS
- *suid* - The cell's ID (tuple). SUID = spatially unique identifier. ('id' is a reserved word in Python)
- *resolution* - The cell's resolution (nonnegative integer).

NOTE:

Several Cell methods have the keyword argument 'plane'. Setting it to True indicates that all input and output points and cells are to be interpreted as lying in the planar DGGS. Setting it to False indicates that they are to be interpreted as lying in the ellipsoidal DGGS.

`area`(*plane=True*)

Return the area of this cell.

`boundary`(*n=2, plane=True, interior=False*)

Return a list of *4*n - 4* boundary points of this cell, *n* on each edge, where *n* >= 2. List the points in clockwise order starting from the cell's upper left corner if *plane* = True, or from the cell's northwest corner if *plane* = False.

If *n* = 2, then the output is the same as vertices(). If *interior* = True, then push the boundary points slighly into the interior of the cell, which is convenient for some graphics methods.

EXAMPLES:

```
>>> rdggs = UNIT_003
>>> c = rdggs.cell(['N', 6])
>>> c.boundary(n=2, plane=True) == c.vertices(plane=True)
True
>>> for p in c.boundary(n=3, plane=True):
...     print(my_round(p, 15))
(-3.1415926535897931, 1.308996938995747)
(-2.879793265790644, 1.308996938995747)
(-2.617993877991494, 1.308996938995747)
(-2.617993877991494, 1.0471975511965981)
(-2.617993877991494, 0.78539816339744795)
(-2.879793265790644, 0.78539816339744795)
(-3.1415926535897931, 0.78539816339744795)
(-3.1415926535897931, 1.0471975511965981)

>>> for p in c.boundary(n=3, plane=False):
...     print(my_round(p, 15))
(-180.0, 74.35752898700072)
(-157.50000000000003, 58.413661903472082)
(-150.0, 41.810314895778603)
(-165.00000000000003, 41.810314895778603)
(-180.0, 41.810314895778603)
(165.0, 41.810314895778603)
(149.99999999999997, 41.810314895778603)
(157.49999999999997, 58.413661903472082)
```

centroid(*plane=True*)

Return the centroid of this planar or ellipsoidal cell.

EXAMPLES:

```
>>> rdggs = RHEALPixDGGS()
>>> c = Cell(rdggs, ['P', 0, 2])
>>> centroid = c.centroid()
>>> nucleus = c.nucleus()
>>> print(centroid == nucleus)
True
```

color(*saturation=0.5*)

Return a unique RGB color tuple for this cell. Inessential graphics method.

contains(*p, plane=True*)

Return True if this cell contains point *p*, and return False otherwise.

EXAMPLES:

```
>>> rdggs = WGS84_003_RADIANS
>>> p = (pi/4, 0)
>>> c = rdggs.cell_from_point(2, p, plane=False)
>>> print(c)
Q44
>>> print(c.contains(p, plane=False))
True
```

ellipsoidal_shape()

Return the shape of this cell ('quad', 'cap', 'dart', or 'skew_quad') when viewed on the ellipsoid.

EXAMPLES:

```
>>> rdggs = RHEALPixDGGS()
>>> print(Cell(rdggs, ['P', 2]).ellipsoidal_shape())
quad
>>> print(Cell(rdggs, ['N', 2]).ellipsoidal_shape())
dart
```

index(*order='resolution'*)

Return the index of *self* when it's ordered according to *order*. Here *order* can be 'resolution' (default) or 'post'. Indices start at 0. The empty cell has index None.

The ordering comes from the way of traversing the tree T of all cells defined as follows. The root of T is a non-cell place holder. The children of the root are the cells A < B < ... < F. The children of a cell in T with suid s are s0 < s1 < ... < sn, where n = self.N_side**2.

The level order index of a nonempty cell is its position (starting from 0) in the level order traversal of T starting at cell A.

The post order index of a nonempty cell is its position (starting from 0) in the post order traversal of T.

EXAMPLES:

```
>>> rdggs = UNIT_003
>>> c = Cell(rdggs, ['N', 2])
>>> print(c.index(order='resolution'))
8
>>> print(c.index(order='post'))
2
```

interior(*n=2*, *plane=True*, *flatten=False*)

Return an *n* x *n* matrix of interior points of this cell. If the cell is planar, space the interior points on a regular square grid. List the points in standard, row-major matrix order. If the cell is ellipsoidal, project the matrix of points to the ellipsoid (longitude-latitude points). If *flatten* = True, then flatten the matrix into a one dimensional array of pairs.

EXAMPLES:

```
>>> rdggs = UNIT_003
>>> c = rdggs.cell(['N'])
>>> for p in c.interior(n=2, plane=False, flatten=True):
...     print(my_round(p, 15))
(90.0, 41.810380145353903)
(-180.0, 41.810380145353903)
(-1.3e-14, 41.810380145353903)
(-90.0, 41.810380145353903)
>>> all([c.contains(p) for p in c.interior(n=5, plane=True, flatten=True)])
True
```

intersects_meridian(*lam*)

Return True if this ellipsoidal cell's boundary intersects the meridian of longitude *lam*, and return False otherwise.

EXAMPLES:

```
>>> rdggs = WGS84_003_RADIANS
>>> c = rdggs.cell(['N', 6])
>>> print(c.intersects_meridian(-pi))
True
>>> print(c.intersects_meridian(-pi/2))
False
```

intersects_parallel(*phi*)

Return True if this cell's boundary intersects the parallel of latitude *phi*, and return False otherwise.

neighbor(*direction*, *plane=True*)

Return this cell's (edge) neighbor in the given direction. If *plane* = True, then the direction is one of the strings 'up', 'right', 'down', 'left', which indicates the desired neighbor relative to x-y coordinates in the following planar neighbor diagram, (drawn for self.N_side = 3) where *self* is the middle cell

```
         up
       *------*
       |      |
       |      |
              |      |
       *------*------*------*
       |      | 012  |      |
  left |      | 345  |      | right
       |      | 678  |      |
       *------*------*------*
              |      |
              |      |
              |      |
              *------*
               down
```

If *plane* = False, then the direction is relative to longitude-latitude coordinates and is one of the strings 'west', 'east', 'north', 'south' for a quad or skew quad cell; 'west', 'east', 'southwest', 'southeast' for a northern dart cell; 'west', 'east', 'northwest', 'northeast' for a southern dart cell; 'south_0', 'south_1', 'south_2', 'south_3' for a northern cap cell; 'north_0', 'north_1', 'north_2', 'north_3' for a southern cap cell; For a cap cell, neighbor directions are numbered in increasing longitude, so that the longitude of the (nucleus of) north_0 is less than the longitude of north_1 is less than the longitude of north_2 is less than the longitude of north_3, and the longitude of the south_0 is less than the longitude of south_1, etc.

The tricky part in the planar scenario is that the neighbor relationships of the six resolution 0 cells is determined by the positions of those cells on the surface of a cube, one cell on each face, and not on a plane. So sometimes rotating cells is needed to compute neighbors.

Return None if the given direction is invalid for this cell.

EXAMPLES:

```
>>> c = Cell(RHEALPixDGGS(), ['N', 0])
>>> print(c.neighbor('down'))
N3
```

neighbors (*plane=True*)

Return this cell's planar or ellipsoidal (edge) neighbors as a dictionary whose keys are the directions of the neighbors. See neighbor() for a list of valid directions.

EXAMPLES:

```
>>> c = Cell(RHEALPixDGGS(), ['N', 0])
>>> for k, v in sorted(c.neighbors().items()):
...     print(k, v)
...
down N3
left R0
right N1
up Q2
```

nucleus (*plane=True*)

Return the nucleus and vertices of this planar or ellipsoidal cell in the order (nucleus, upper left corner, lower left corner, lower right corner, upper right corner) with reference to the planar cell. The output for ellipsoidal cells is the projection onto the ellipsoid of the output for planar cells. In particular, while the nucleus of a planar cell is its centroid, the nucleus of an ellipsoidal cell is not its centroid. To compute the centroid of a cell, use centroid() below.

EXAMPLES:

```
>>> rdggs = UNIT_003
>>> c = rdggs.cell(['N'])
>>> print(my_round(c.nucleus(), 15))
(-2.3561944901923448, 1.5707963267948959)
```

nw_vertex (*plane=True*)

If *plane* = False, then return the northwest vertex of this ellipsoidal cell. If *plane* = True, then return the projection onto the plane of the ellipsoidal northwest vertex. On quad cells and cap cells, this function returns the same output as ul_vertex(). On skew quad cells and dart cells, this function returns output different from ul_vertex().

WARNING: The northwest vertex of a cell might not lie in the cell, because not all cells contain their boundary.

EXAMPLES::

```
>>> rdggs = RHEALPixDGGS()
>>> c = rdggs.cell(['P', 5, 7]) # Quad cell.
>>> print(my_round(c.ul_vertex(plane=True), 15))
(-2225148.7007489, -556287.17518722452)
>>> print(my_round(c.nw_vertex(plane=True), 15))
(-2225148.7007489, -556287.17518722452)


>>> c = rdggs.cell(['S', 4])  # Cap cell.
>>> print(my_round(c.ul_vertex(plane=True), 15))
(-16688615.255616743, -8344307.6278083706)
>>> print(my_round(c.nw_vertex(plane=True), 15))
(-16688615.255616743, -8344307.6278083706)


>>> c = rdggs.cell(['N', 4, 3]) # Skew quad cell.
>>> print(my_round(c.ul_vertex(plane=True), 15))
(-16688615.255616743, 10569456.328557272)
>>> print(my_round(c.nw_vertex(plane=True), 15))
(-15576040.905242294, 10569456.328557272)


>>> c = rdggs.cell(['S', 4, 3])  # Skew quad cell.
>>> print(my_round(c.ul_vertex(plane=True), 15))
(-16688615.255616743, -9456881.9781828206)
>>> print(my_round(c.nw_vertex(plane=True), 15))
(-16688615.255616743, -10569456.328557272)


>>> c = rdggs.cell(['N', 6, 2])  # Dart cell.
>>> print(my_round(c.ul_vertex(plane=True), 15))
(-17801189.605991192, 8344307.6278083716)
>>> print(my_round(c.nw_vertex(plane=True), 15))
(-16688615.255616743, 8344307.6278083716)


>>> c = rdggs.cell(['S', 6, 2])  # Dart cell.
>>> print(my_round(c.ul_vertex(plane=True), 15))
(-17801189.605991192, -11682030.678931719)
>>> print(my_round(c.nw_vertex(plane=True), 15))
(-16688615.255616743, -12794605.029306168)
```

predecessor(*resolution=None*)

Return the greatest resolution *resolution* cell less than *self*. Note: *self* need not be a resolution *resolution* cell.

EXAMPLES:

```
>>> c = Cell(RHEALPixDGGS(), ('N', 0, 8))
>>> print(c.predecessor())
N07
>>> print(c.predecessor(0))
None
>>> print(c.predecessor(1))
None
>>> print(c.predecessor(3))
N088
```

random_point (*plane=True*)

Return a random point in this cell. If *plane* = True, then choose the point from the planar cell. Otherwise, choose the point from the ellipsoidal cell.

EXAMPLES:

```
>>> c = Cell(RHEALPixDGGS(), ['N', 0])
>>> print(c.random_point(plane=False))
(1.4840291937583836, 0.90042819146088571)
```

region ()

Return the region of this cell: 'equatorial', 'north_polar', or 'south_polar'.

EXAMPLES:

```
>>> rdggs = RHEALPixDGGS()
>>> print(Cell(rdggs, ['P', 2]).region())
equatorial
>>> print(Cell(rdggs, ['N', 2]).region())
north_polar
```

rotate (*quarter_turns*)

Return the cell that is the result of rotating this cell's resolution 0 supercell by *quarter_turns* quarter turns anticlockwise. Used in neighbor().

EXAMPLES:

```
>>> c = Cell(RHEALPixDGGS(), ['N', 0])
>>> print([str(c.rotate(t)) for t in range(4)])
['N0', 'N2', 'N8', 'N6']
```

rotate_entry (*x*, *quarter_turns*)

Let N = self.N_side and rotate the N x N matrix of subcell numbers

```
0         1           ... N - 1
N         N+1         ... 2*N - 1
...
(N-1)*N   (N-1)*N+1   ... N**2-1
```

anticlockwise by *quarter_turns* quarter turns to obtain a new table with entries f(0), f(1), ..., f(N**2 - 1) read from left to right and top to bottom. Given entry number *x* in the original matrix, return *f(x)*. Used in rotate().

INPUT:

- *x* - A letter from RHEALPixDGGS.cells0 or one of the integers 0, 1, ..., N**2 - 1.
- *quarter_turns* - 0, 1, 2, or 3.

EXAMPLES:

```
>>> c = Cell(RHEALPixDGGS(), ['P', 2])
>>> print([c.rotate_entry(0, t) for t in range(4)])
[0, 2, 8, 6]
```

NOTES:

Operates on letters from RHEALPixDGGS.cells0 too. They stay fixed under f. Only depends on *self* through *self.N_side*.

subcell (*other*)

Subcell (subset) relation on cells.

EXAMPLES:

```
>>> a = Cell(RHEALPixDGGS(), ('N', 1))
>>> b = Cell(RHEALPixDGGS(), ['N'])
>>> print(a.subcell(b))
```

```
True
>>> print(b.subcell(a))
False
```

subcells(*resolution=None*)

Generator function for the set of all resolution *resolution* subcells of this cell. If *resolution=None*, then return a generator function for the children of this cell.

EXAMPLES:

```
>>> c = Cell(RHEALPixDGGS(), ['N'])
>>> print([str(cell) for cell in c.subcells()])
['N0', 'N1', 'N2', 'N3', 'N4', 'N5', 'N6', 'N7', 'N8']
```

successor(*resolution=None*)

Return the least resolution *resolution* cell greater than *self*. Note: *self* need not be a resolution *resolution* cell.

EXAMPLES:

```
>>> c = Cell(RHEALPixDGGS(), ('N', 8, 2))
>>> print(c.successor())
N83
>>> print(c.successor(0))
O
>>> print(c.successor(1))
O0
>>> print(c.successor(3))
N830
```

static **suid_from_index**(*rdggs*, *index*, *order='resolution'*)

Return the suid of a cell from its index. The index is according to the cell ordering *order*, which can be 'resolution' (default) or 'post'. See the *index()* docstring for more details on orderings. For internal use.

suid_rowcol()

Return the pair of row- and column-suids of *self*, each as tuples.

EXAMPLES:

```
>>> rdggs = RHEALPixDGGS()
>>> c = Cell(rdggs, ['N', 7, 3])
>>> rsuid, csuid = c.suid_rowcol()
>>> print(rsuid == ('N', 2, 1))
True
>>> print(csuid == ('N', 1, 0))
True
```

ul_vertex(*plane=True*)

If *plane* = True, then return the upper left vertex of this planar cell. If *plane* = False, then return the projection onto the ellipsoid of the planar upper left vertex. Note that for polar cells, this projection is not necessarily the northwest vertex. For the latter vertex use nw_vertex().

WARNING: The upper left vertex of a cell might not lie in the cell, because not all cells contain their boundary.

EXAMPLES:

```
>>> c = Cell(UNIT_003, ['N', 0])
>>> print(c.ul_vertex() == (-pi, 3*pi/4))
True
```

vertices(*plane=True*, *trim_dart=False*)

If *plane* = True, then assume this cell is planar and return its four vertices in the order (upper left corner, upper right corner, lower right corner, lower left corner). If *plane* = False, then assume this cell is ellipsoidal and return the projection of the planar vertices in the order (northwest, northeast, southeast, southwest). If *plane* = False, this cell is a dart cell, and *trim_dart* = True, then remove the one non-vertex point from the output. (Dart cells only have three vertices.)

EXAMPLES:

```
>>> rdggs = UNIT_003
>>> c = rdggs.cell(['N'])
>>> for p in c.vertices():
...     print(my_round(p, 15))
(-3.1415926535897931, 2.3561944901923448)
(-1.5707963267948959, 2.3561944901923448)
(-1.5707963267948959, 0.78539816339744795)
(-3.1415926535897931, 0.78539816339744795)

>>> rdggs = WGS84_003
>>> c = rdggs.cell(['N', 0])
>>> for p in c.vertices(plane=False):
...     print(my_round(p, 15))
(89.999999999999929, 74.39069094879062)
(119.99999999999997, 41.87385774220941)
(90.0, 41.87385774220941)
(60.000000000000007, 41.87385774220941)

>>> for p in c.vertices(plane=False, trim_dart=True):
...     print(my_round(p, 15))
(89.999999999999929, 74.39069094879062)
(119.99999999999997, 41.87385774220941)
(60.000000000000007, 41.87385774220941)

>>> c = rdggs.cell(['S', 0])
>>> for p in c.vertices(plane=False):
...     print(my_round(p, 15))
(149.99999999999997, -41.87385774220941)
(-180.0, -41.87385774220941)
(-150.0, -41.87385774220941)
(-180.0, -74.390690948790649)
>>> for p in c.vertices(plane=False, trim_dart=True):
...     print(my_round(p, 15))
(149.99999999999997, -41.87385774220941)
(-150.0, -41.87385774220941)
(-180.0, -74.390690948790649)
```

width(*plane=True*)

Return the width of this cell. If *plane* = False, then return None, because ellipsoidal cells don't have a fixed width.

EXAMPLES:

```
>>> c = Cell(UNIT_003, ('N', 8))
>>> print(c)
N8
>>> c.width() == pi/2*3**(-1)
True
```

xy_range()

Return the x- and y-coordinate extremes of the planar version of this cell in the format ((x_min, x_max), (y_min, y_max)).

EXAMPLES:

```
>>> rdggs = UNIT_003
>>> c = rdggs.cell(['N'])
>>> c.xy_range() == ((-pi, -pi/2), (pi/4, 3*pi/4))
True
```

class `rhealpix_dggs.`**`RHEALPixDGGS`**(*ellipsoid=<ellipsoids.Ellipsoid object at 0x102425610>, N_side=3, north_square=0, south_square=0, max_areal_resolution=1*)

Bases: `object`

Represents an rHEALPix DGGS on a given ellipsoid.

CLASS ATTRIBUTES:

- *cells0* - A list of the resolution 0 cell IDs (strings).

INSTANCE ATTRIBUTES:

- *ellipsoid* - The underlying ellipsoid (Ellipsoid instance).
- *N_side* - An integer of size at least 2. Each planar cell has N_side x N_side child cells.
- *(north_square, south_square)* - Integers between 0 and 3 indicating the positions of north polar and south polar squares, respectively, of the rHEALPix projection used.
- *max_areal_resolution* - An area measured in square meters that upper bounds the area of the smallest ellipsoidal grid cells.
- *max_resolution* - A nonnegative integer that is the maximum grid resolution needed to have ellipsoidal cells of area at most *max_areal_resolution*.
- *child_order* - A dictionary of the ordering (Morton order) of child cells of a cell in terms of the row-column coordinates in the matrix of child cells. Child cell are numbered 0 to *N_side**2 -1* from left to right and top to bottom.
- *ul_vertex* - A dictionary with key-value pairs (c, (x, y)), where c is an element of *cells0* and (x, y) is the upper left corner point of the resolution 0 planar cell c.
- *atomic_neighbors* - A dictionary with key-value pairs (n, {'up': a, 'down': b, 'left': c, 'right': d}), where n, a, b, c, and d are elements of *cells0* or {0, 1, ..., *N_side**2 -1*}. Describes the planar (edge) neighbors of cell0 letter / child cell number n.

NOTE:

Several RHEALPixDGGS methods have the keyword argument 'plane'. Setting it to True indicates that all input and output points and cells are interpreted as lying in the planar DGGS. Setting it to False indicates that they are interpreted as lying in the ellipsoidal DGGS.

`cell`(*suid=None, level_order_index=None, post_order_index=None*)

Return a cell (Cell instance) of this DGGS either from its ID or from its resolution and index.

EXAMPLES:

```
>>> rdggs = RHEALPixDGGS()
>>> c = rdggs.cell(('N', 4, 5))
>>> print(isinstance(c, Cell))
True
>>> print(c)
N45
```

`cell_area`(*resolution, plane=True*)

Return the area of a planar or ellipsoidal cell at the given resolution.

EXAMPLES:

```
>>> rdggs = UNIT_003
>>> a = rdggs.cell_area(1)
>>> print(a == (pi/6)**2)
True
>>> print(rdggs.cell_area(1, plane=False) == 8/(3*pi)*a)
True
```

cell_from_point (*resolution*, *p*, *plane=True*)

Return the resolution *resolution* cell that contains the point *p*. If *plane* = True, then *p* and the output cell lie in the planar DGGS. Otherwise, *p* and the output cell lie in the ellipsoidal DGGS.

EXAMPLES:

```
>>> rdggs = RHEALPixDGGS()
>>> p = (0, 0)
>>> c = rdggs.cell_from_point(1, p)
>>> print(c)
Q3
```

cell_from_region (*ul*, *dr*, *plane=True*)

Return the smallest planar or ellipsoidal cell wholly containing the region bounded by the axis-aligned rectangle with upper left and lower right vertices given by the the points *ul* and *dr*, respectively. If such as cell does not exist, then return None. If *plane* = True, then *ul* and *dr* and the returned cell lie in the planar DGGS. Otherwise, *ul* and *dr* and the returned cell lie in the ellipsoidal DGGS.

To specify an ellipsoidal cap region, set *ul* = (-pi, pi/2) and *dr* = (-pi, phi) for a northern cap from latitudes pi/2 to phi, or set *ul* = (-pi, phi) and *dr* = (-pi, -pi/2) for a southern cap from latitudes phi to -pi/2. (As usual, if *self.ellipsoid.radians* = False, then use degrees instead of radians when specifying ul and dr.)

EXAMPLES:

```
>>> rdggs = UNIT_003
>>> p = (0, pi/12)
>>> q = (pi/6 - 1e-6, 0)
>>> c = rdggs.cell_from_region(p, q)
>>> print(c)
Q3
```

cell_latitudes (*resolution*, *phi_min*, *phi_max*, *nucleus=True*, *plane=True*)

Return a list of every latitude phi whose parallel intersects a resolution *resolution* cell nucleus and satisfies *phi_min* < phi < *phi_max*. If *plane* = True, then use rHEALPix y-coordinates for *phi_min*, *phi_max*, and the result. Return the list in increasing order. If *nucleus* = False, then return a list of every latitude phi whose parallel intersects the north or south boundary of a resolution *resolution* cell and that satisfies *phi_min* < phi < *phi_max*.

NOTE:

By convention, the pole latitudes pi/2 and -pi/2 (or their corresponding rHEALPix y-coordinates) will be excluded.

There are 2*self.N_side**resolution - 1 nuclei latitudes between the poles if self.N_side is odd and 2*self.N_side**resolution if self.N_side is even. Consequently, there are 2*self.N_side**resolution boundary latitudes between the poles if self.N_side is odd and 2*self.N_side**resolution - 1 boundary latitudes if self.N_side is even.

EXAMPLES:

```
>>> rdggs = WGS84_003_RADIANS
>>> for phi in rdggs.cell_latitudes(1, -pi/2, pi/2, plane=False):
...     print(my_round(phi, 15))
-1.020505844
-0.461443149003
-0
0.461443149003
1.020505844
1.57079632679
>>> for phi in rdggs.cell_latitudes(1, -pi/2, pi/2, nucleus=False, plane=False):
...     print(my_round(phi, 15))
-1.29836248989
-0.730836688113
-0.224577156195
```

```
0.224577156195
0.730836688113
1.29836248989
```

cell_width(*resolution*, *plane=True*)

Return the width of a planar cell at the given resolution. If *plane* = False, then return None, because the ellipsoidal cells don't have constant width.

EXAMPLES:

```
>>> rdggs = UNIT_003
>>> print(rdggs.cell_width(0) == pi/2)
True
>>> print(rdggs.cell_width(1) == pi/6)
True
```

cells0 = ['N', 'O', 'P', 'Q', 'R', 'S']

cells_from_meridian(*resolution*, *lam*, *phi_min*, *phi_max*)

Return a list of the resolution *resolution* cells that intersect the meridian segment of longitude *lam* whose least latitude is *phi_min* and whose greatest latitude is *phi_max*. Sort the cells from north to south and west to east in case two cells with the same nucleus latitude intersect the meridian.

EXAMPLES:

```
>>> rdggs = WGS84_003_RADIANS
>>> cells = rdggs.cells_from_meridian(1, 0.1, -pi/2, pi/2)
>>> print([str(cell) for cell in cells])
['N4', 'N2', 'N1', 'Q0', 'Q3', 'Q6', 'S8', 'S7', 'S4']
```

cells_from_parallel(*resolution*, *phi*, *lam_min*, *lam_max*)

Return a list of the resolution *resolution* cells that intersect the parallel segment of latitude *phi* whose least longitude is *lam_min* and whose greatest longitude is *lam_max*. Sort the list from west to east.

EXAMPLES:

```
>>> rdggs = WGS84_003_RADIANS
>>> cells = rdggs.cells_from_parallel(1, pi/3, -pi, pi)
>>> print([str(cell) for cell in cells])
['N6', 'N7', 'N8', 'N5', 'N2', 'N1', 'N0', 'N3']
```

cells_from_region(*resolution*, *ul*, *dr*, *plane=True*)

If *plane* = True, then return a list of lists of resolution *resolution* cells that cover the axis-aligned rectangle whose upper left and lower right vertices are the points *ul* and *dr*, respectively. In the output, sort each sublist of cells from left to right (in the planar DGGS) and sort the sublists from top to bottom.

If *plane* = False, then return a list of lists of resolution *resolution* cells that cover the longitude-latitude aligned ellipsoidal quadrangle whose northwest and southeast vertices are the points *ul* and *dr*, respectively. Defunct quads with *ul* = (stuff, pi/2) or *dr* = (stuff, -pi/2) also work (and rely on the fact that the north and south pole can both be specified by infinitely many longitudes).

To specify an ellipsoidal cap region, set *ul* = (-pi, pi/2) and *dr* = (-pi, phi) for a northern cap from latitudes pi/2 to phi, or set *ul* = (-pi, phi) and *dr* = (-pi, -pi/2) for a southern cap from latitudes phi to -pi/2. (As usual, if *self.ellipsoid.radians* = False, then use degrees instead of radians when specifying ul and dr.)

In the output, sort each sublist of cells from west to east (in the ellipsoidal DGGS) and sort the sublists from north to south.

Return the empty list if if *ul[0]* > *dr[0]* or *ul[1]* < *dr[1]*.

NOTE:

If *plane* = True, then the resulting list is a matrix, that is, each sublist has the same length. This is not necessarily so if *plane* = False; see the examples below.

EXAMPLES:

```
>>> rdggs = WGS84_003_RADIANS
>>> R_A = rdggs.ellipsoid.R_A
>>> ul = R_A*array((-0.1, pi/4))
>>> dr = R_A*array((0.1, -pi/4))  # Rectangle
>>> M = rdggs.cells_from_region(1, ul, dr)
>>> for row in M:
...     print([str(cell) for cell in row])
['P2', 'Q0']
['P5', 'Q3']
['P8', 'Q6']

>>> ul = (0, pi/3)
>>> dr = (pi/2, 0)  # Quad
>>> M = rdggs.cells_from_region(1, ul, dr, plane=False)
>>> for row in M:
...     print([str(cell) for cell in row])
['N2', 'N1', 'N0']
['Q0', 'Q1', 'Q2', 'R0']
['Q3', 'Q4', 'Q5', 'R3']

>>> ul = (0, -pi/6)
>>> dr = (pi/2, -pi/2)  # Defunct quad / lune segment
>>> M = rdggs.cells_from_region(1, ul, dr, plane=False)
>>> for row in M:
...     print([str(cell) for cell in row])
['Q6', 'Q7', 'Q8', 'R6']
['S8', 'S7', 'S6']
['S4']

>>> ul = (-pi, -pi/5)
>>> dr = (-pi, -pi/2)  # Cap
>>> M = rdggs.cells_from_region(1, ul, dr, plane=False)
>>> for row in M:
...     print([str(cell) for cell in row])
['O6', 'O7', 'O8', 'P6', 'P7', 'P8', 'Q6', 'Q7', 'Q8', 'R6', 'R7', 'R8']
['S0', 'S1', 'S2', 'S5', 'S8', 'S7', 'S6', 'S3']
['S4']
```

combine_triangles(*u*, *v*, *inverse=False*)

Return the combine_triangles() transformation of the point *(u, v)* (or its inverse if *inverse* = True) appropriate to the underlying ellipsoid. It maps the HEALPix projection to the rHEALPix projection.

EXAMPLES:

```
>>> rdggs = UNIT_003
>>> p = (0, 0)
>>> q = (-pi/4, pi/2)
>>> print(rdggs.combine_triangles(*p))
(0.0, 0.0)
>>> print(my_round(rdggs.combine_triangles(*q), 15))
(-2.3561944901923448, 1.5707963267948959)
```

grid(*resolution*)

Generator function for all the cells at resolution *resolution*.

EXAMPLES:

```
>>> rdggs = RHEALPixDGGS()
>>> grid0 = rdggs.grid(0)
>>> print([str(x) for x in grid0])
['N', 'O', 'P', 'Q', 'R', 'S']
```

healpix(*u*, *v*, *inverse=False*)

Return the HEALPix projection of point *(u, v)* (or its inverse if *inverse* = True) appropriate to this rHEALPix DGGS.

EXAMPLES:

```
>>> rdggs = UNIT_003_RADIANS
>>> print(my_round(rdggs.healpix(-pi, pi/2), 15))
(-2.3561944901923448, 1.5707963267948959)
```

NOTE:

Uses `pj_healpix` instead of the PROJ.4 version of HEALPix.

`interval`(*a*, *b*)
Generator function for all the resolution *max(a.resolution, b.resolution)* cells between cell *a* and cell *b* (inclusive and with respect to the postorder ordering on cells). Note that *a* and *b* don't have to lie at the same resolution.

EXAMPLES:

```
>>> rdggs = RHEALPixDGGS()
>>> a = rdggs.cell(('N', 1))
>>> b = rdggs.cell(('N',))
>>> print([str(c) for c in list(rdggs.interval(a, b))])
['N1', 'N2', 'N3', 'N4', 'N5', 'N6', 'N7', 'N8']
```

`minimal_cover`(*resolution*, *points*, *plane=True*)
Find the minimal set of resolution *resolution* cells that covers the list of points *points*. If *plane* = True, then assume *points* is a list of x-y coordinates in the planar DGGS. If *plane* = False, then assume *points* is a list of longitude-latitude coordinates in the ellipsoidal DGGS. This method will be made redundant by standard GIS rasterization tools that implement the rHEALPix projection.

EXAMPLES:

```
>>> rdggs = RHEALPixDGGS()
>>> c1 = rdggs.cell(['N', 0, 2, 1])
>>> c2 = rdggs.cell(['P', 7, 3, 3])
>>> points = [c.nucleus() for c in [c1, c2]]
>>> for r in range(5):
...     cover = sorted(rdggs.minimal_cover(r, points))
...     print([str(c) for c in cover])
['N', 'P']
['N0', 'P7']
['N02', 'P73']
['N021', 'P733']
['N0214', 'P7334']
```

`num_cells`(*res_1*, *res_2=None*, *subcells=False*)
Return the number of cells of resolutions *res_1* to *res_2* (inclusive). Assume *res_1* <= *res_2*. If *subcells* = True, then return the number of subcells at resolutions *res_1* to *res_2* (inclusive) of a cell at resolution *res_1*. If *res_2=None* and *subcells=False, then return the number of cells at resolution 'res_1*. If *res_2=None* and *subcells* = True, then return the number of subcells from resolution *res_1* to resolution *self.max_resolution*.

EXAMPLES:

```
>>> rdggs = RHEALPixDGGS()
>>> rdggs.num_cells(0)
6
>>> rdggs.num_cells(0, 1)
60
>>> rdggs.num_cells(0, subcells=True)
231627523606480
>>> rdggs.num_cells(0, 1, subcells=True)
10
```

```
>>> rdggs.num_cells(5, 6, subcells=True)
10
```

plot_cells (*cells*, *surface='plane'*, *label=True*, *fontsize=15*, *saturation=0.5*)

Plot the given list of cells on the given surface. The cells should all come from the same rHEALPix DGGS. Inessential graphics method. Requires Sage graphics methods.

INPUT:

- *cells* - A list of cells from a common rHEALPix DGGS.
- *surface* - (Optional; default = 'plane'). One of the strings 'plane', 'plane_lonlat', 'cube', or 'ellipsoid'. Surface to draw cells on.
- *label* - (Optional; default = True). If True, then label cells with their names. If False, then don't.
- *saturation* - (Optional) Number between 0 and 1 indicating the saturation value of the cell color.

random_cell (*resolution=None*)

Return a cell of the given resolution chosen uniformly at random from all cells at that resolution. If *resolution=None*, then the cell resolution is first chosen uniformly at random from [0,..,self.max_resolution].

EXAMPLES:

```
>>> print(RHEALPixDGGS().random_cell())
S480586367780080
```

random_point (*plane=True*)

Return a point in this DGGS sampled uniformly at random from the plane or from the ellipsoid.

EXAMPLES:

```
>>> rdggs = RHEALPixDGGS()
>>> print(E.random_point())
(-1.0999574573422948, 0.21029104897701129)
```

rhealpix (*u*, *v*, *inverse=False*)

Return the rHEALPix projection of the point *(u, v)* (or its inverse if *inverse* = True) appropriate to this rHEALPix DGGS.

EXAMPLES:

```
>>> rdggs = UNIT_003_RADIANS
>>> print(my_round(rdggs.rhealpix(0, pi/3), 15))
(-1.8582720066840039, 2.0687188103032379)
```

NOTE:

Uses `pj_rhealpix` instead of the PROJ.4 version of rHEALPix.

triangle (*x*, *y*, *inverse=True*)

If *inverse* = False, then assume *(x,y)* lies in the image of the HEALPix projection that comes with this DGGS, and return the number of the HEALPix polar triangle (0, 1, 2, 3, or None) and the region ('north_polar', 'south_polar', or 'equatorial') that *(x, y)* lies in. If *inverse* = True, then assume *(x, y)* lies in the image of the rHEALPix projection that comes with this DGGS, map *(x, y)* to its HEALPix image (x', y'), and return the number of the HEALPix polar triangle and the region that (x', y') lies in. If *(x, y)* lies in the equatorial region, then the triangle number returned is None.

OUTPUT:

The pair (triangle_number, region).

NOTES:

This is a wrapper for pj_rhealpix.triangle().

EXAMPLES:

```
>>> rdggs = RHEALPixDGGS()
>>> c = rdggs.cell(['N', 7])
>>> print(rdggs.triangle(*c.nucleus(), inverse=True))
(0, 'north_polar')

>>> c = rdggs.cell(['N', 3])
>>> print(rdggs.triangle(*c.nucleus(), inverse=True))
(3, 'north_polar')

>>> c = rdggs.cell(['P', 3])
>>> print(rdggs.triangle(*c.nucleus(), inverse=True))
(None, 'equatorial')

>>> c = rdggs.cell(['S', 5, 2])
>>> print(rdggs.triangle(*c.nucleus(), inverse=True))
(1, 'south_polar')
```

xyz(*u*, *v*, *lonlat=False*)

Given a point *(u, v)* in the planar image of the rHEALPix projection, project it back to the ellipsoid and return its 3D rectangular coordinates. If *lonlat* = True, then assume *(u, v)* is a longitude-latitude point.

EXAMPLES:

```
>>> rdggs = UNIT_003_RADIANS
>>> print(my_round(rdggs.xyz(0, pi/4, lonlat=True), 15))
(0.70710678118654802, 0.0, 0.70710678118654802)
```

xyz_cube(*u*, *v*, *lonlat=False*)

Given a point *(u, v)* in the planar version of this rHEALPix DGGS, fold the rHEALPix image into a cube centered at the origin, and return the resulting point's 3D rectangular coordinates. If *lonlat* = True, then assume *(u, v)* is a longitude-latitude point.

EXAMPLES:

```
>>> rdggs = UNIT_003
>>> print(my_round(rdggs.xyz_cube(0, 0), 15))
(0.78539816339744795, 0.0, -0.78539816339744795)
```

CHAPTER

EIGHT

THE DISTORTION MODULE

This Python 3.3 module computes linear and areal distortion statistics of map projections. It is used for analysis only and so is not essential to manipulating the rHEALPix discrete global grid system.

CHANGELOG:

- Alexander Raichev (AR), 2012-10-01: Initial version.
- AR, 2013-01-21: Wasn't working in degrees mode. Fixed that bug.
- AR, 2013-07-23: Ported to Python 3.3.

distortion.**distortion**(*T*, *lam*, *phi*)

Return ((x, y), mad, ld, ad) Here *(x, y)* = *T(lam, phi)*, the image under the map projection *T* (an projection_tools.Proj or projection_tools.Proj4 instance) of the longitude-latitude point *(lam, phi)*; mad = maximum angular distortion at (x, y) = 2*arcsin((A - B)/(A + B)) ld = linear distortion at (x, y) = A/B; ad = areal distortion at (x, y) = AB; A and B are the major and minor radii, respectively, of the Tissot ellipse at (x, y).

EXAMPLES:

```
>>> from projection_wrapper import Proj
>>> from ellipsoids import WGS84_ELLIPSOID_RADIANS
>>> f = Proj(ellipsoid=WGS84_ELLIPSOID_RADIANS, proj='healpix')
>>> print(my_round(distortion(f, 0, pi/6), 15))
((0.0, 3748655.1150495014), 0.121755482930707, 1.1295629172526771, 1.1780969100283301)
```

distortion.**distortion_stats**(*T*, *sample*, *my_round_numbers=3*)

Return the sample minimum, sample maximum, sample median, sample mean, and sample standard deviation of the maximum angular distortion, linear distortion, and area distortion functions for the map projection *T* (an projection_tools.Proj or projection_tools.Proj4 instance) of the list *sample* of longitude-latitude points chosen from the surface *T.ellipsoid*. Most likely you will want sample to comprise points sampled uniformly at random from the surface of *T.ellipsoid* (and not simply sampled uniformly at random from the rectangle (-pi, pi) x (-pi/2, pi/2)).

OUTPUT:

(distortions, stats), where distortions is a list of distortion() outputs for each longitude- latitude point sampled; stats is the list of lists [maximum angular distortion stats, linear distortion stats, area distortion stats], where each stats sublist is of the form [sample mean, sample standard deviation, sample minimum, sample maximum, sample median].

EXAMPLES:

```
>>> from projection_wrapper import Proj
>>> from ellipsoids import WGS84_ELLIPSOID_RADIANS
>>> E = WGS84_ELLIPSOID_RADIANS
>>> f = Proj(ellipsoid=E, proj='healpix')
>>> sample = [E.random_point() for i in range(100)]
>>> print(distortion_stats(f, sample)[1])
[[0.309, 0.238, 0.001, 0.838, 0.178], [1.41, 0.375, 1.001, 2.372, 1.195], [1.178, 0.0, 1.178,
```

distortion.**fff_coeffs**(*T*, *lam*, *phi*)

Return numerical approximations of the first fundamental form coefficients E, F, and G (in that order) of map projection *T* (an projection_tools.Proj or projection_tools.Proj4 instance) at longitude *lam* and latitude *phi*.

EXAMPLES:

```
>>> from projection_wrapper import Proj
>>> from ellipsoids import WGS84_ELLIPSOID_RADIANS
>>> f = Proj(ellipsoid=WGS84_ELLIPSOID_RADIANS, proj='healpix')
>>> print(my_round(fff_coeffs(f, 0, pi/6), 15))
(40635288880650.484, 0.0, 42251277118793.328)
```

distortion.**scale_factors**(*T*, *lam*, *phi*)

Return numerical approximations of the local scale factors s_M, s_P, and s_A of the map projection *T* (an projection_tools.Proj or projection_tools.Proj4 instance) at longitude *lam* and latitude *phi*, where s_M is the local linear scale along meridians, s_P is the local linear scale along parallels, and s_A is the local area scale. Also return theta (in radians), the angle between the vectors (delxdellam, delydellam) and (delxdelphi, delydelphi).

OUTPUT:

(s_M, s_P, s_A, theta)

EXAMPLES:

```
>>> from projection_wrapper import Proj
>>> from ellipsoids import WGS84_ELLIPSOID_RADIANS
>>> f = Proj(ellipsoid=WGS84_ELLIPSOID_RADIANS, proj='healpix')
>>> print(my_round(scale_factors(f, 0, pi/6), 15))
(1.0212575853790069, 1.1535746974071359, 1.1780969100283301, 1.5707963267948959)
```

distortion.**utm_zone**(*lam*, *phi*)

Return the Universal Transverse Mercator zone and hemisphere for longitude *lam* and latitude *phi* given in radians. Based on the WGS84 ellipsoid. Return None if *phi* is out of bounds, that is, if *phi* is greater than 84 degrees or less than 80 degrees.

EXAMPLES:

```
>>> print(utm_zone(0, 84*pi/180))
31
>>> print(utm_zone(0, 85*pi/180))
None
```

CHAPTER

NINE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

BIBLIOGRAPHY

[GRS2013] Robert Gibb, Alexander Raichev, Michael Speth, The rHEALPix discrete global grid system, in preparation, 2013.

[CaRo2007] Mark R. Calabretta and Boudewijn F. Roukema, Mapping on the healpix grid, Monthly Notices of the Royal Astronomical Society 381 (2007), no. 2, 865–872.

PYTHON MODULE INDEX

INDEX