rHEALPixDGGS Documentation

Release 0.5.4

Alexander Raichev

Apr 10, 2025

CONTENTS

CHAPTER

ONE

INTRODUCTION

rHEALPixDGGS is a Python 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 >=3.11
- NumPy >=1.25.2 Base N-dimensional array package
- SciPy >=1.11.2 Fundamental library for scientific computing
- Matplotlib >=3.7.2 Comprehensive 2D Plotting
- Pyproj >=3.6 Python interface to the PROJ.4 cartographic library

1.2 Installation

The package is available on PyPI, the Package Index from where it can be installed as follows:

pip install rhealpixdggs

rHEALPixDGGS is also available for download from Landcare Research's github repository https://github.com/manaakiwhenua/rhealpixdggs-py from wher the latest version can be cloned.

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 `ellipsoids` and `projection_wrapper` Modules

The `ellipsoids` module implements functions and constants dealing with ellipsoids of revolution (which include spheres but not general triaxial ellipsoids). For brevity hereafter, the word 'ellipsoid' abbreviates 'ellipsoid of revolution'.

The module `projection_wrapper` implements a wrapper for the map projections of ellipsoids defined in `pj_healpix`, `pj_rhealpix`, and Pyproj.

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 `dggs` by editing the `HOMEMADE_PROJECTIONS` line in `projection_wrapper`.

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

```
>>> from rhealpixdggs.projection_wrapper import *
>>> from rhealpixdggs.ellipsoids import *
```

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

```
>>> ellps_1 = Ellipsoid(a=5, e=0.8)
>>> print(ellps_1)
ellipsoid:
    R_A = 4.322001171188888
    a = 5
    b = 2.999999999999999
    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. For example, `R_A` is the authalic radius of the ellipsoid, the radius of the sphere that has the same area as the ellipsoid.

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 = 4.322001171188888
    a = 5
    b = 2.999999999999999
    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 = 6371007.180918476
    a = 6378137.0
    b = 6356752.314245179
    e = 0.081819190842621
    f = 0.003352810664747
    lat_0 = 0
```

(continues on next page)

(continued from previous page)

```
    lon_0 = 0
    radians = False
    sphere = False
>>> print(WGS84_ELLIPSOID_RADIANS)
ellipsoid:
    R_A = 6371007.180918476
    a = 6378137.0
    b = 6356752.314245179
    e = 0.081819190842621
    f = 0.003352810664747
    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:

```
>>> from numpy.testing import assert_allclose
>>> h = Projection(ellps_1, 'healpix')
>>> rh = Projection(ellps_1, 'rhealpix', north_square=1, south_square=2)
>>> assert_allclose(h(0, 60), (0.0, 3.35127855017803), rtol=1e-14, atol=0) == None
True
>>> assert_allclose(rh(0, 60), (0.0, 3.35127855017803), rtol=1e-14, atol=0) == None
True
>>> assert_allclose(h(0, 70), (0.864006732389895, 4.258498514443268), rtol=1e-14,␣
↪atol=0) == None
True
>>> assert_allclose(rh(0, 70), (-0.864006732389895, 4.258498514443268), rtol=1e-14,␣
↪atol=0) == None
True
```

1.3.2 Using the dggs Module

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

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

```
>>> from rhealpixdggs.dggs import *
```

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

```
>>> from rhealpixdggs.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_resolution = 1
    max_resolution = 15
    ellipsoid:
        R_A = 6371007.180918476
        a = 6378137.0
```

(continues on next page)

(continued from previous page)

```
        b = 6356752.314245179
        e = 0.08181919084262149
        f = 0.003352810664747480 5
        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 = 6371007.180918476
        a = 6378137.0
        b = 6356752.314245179
        e = 0.08181919084262149
        f = 0.003352810664747480 5
        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
```

(continues on next page)

(continued from previous page)

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

Find the planar (edge) neighbors of this cell

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

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

```
>>> expected_results = [
...     [
...         (0.0, 58.52801748206219),
...         (44.999999999999964, 58.52801748206219),
...         (90.0, 58.52801748206219)
...     ], [
...         (14.999999999999998, 26.490118751439734),
...         (45.0, 26.490118751439734),
...         (74.99999999999999, 26.490118751439734),
...         (105.00000000000001, 26.490118751439734)
...     ], [
```

(continues on next page)

(continued from previous page)

```
...         (14.999999999999998, 0),
...         (45.0, 0),
...         (74.99999999999999, 0),
...         (105.00000000000001, 0)
...     ]]
>>> for i, row in enumerate(cells):
...     for j, cell in enumerate(row):
...         print(cell, cell.ellipsoidal_shape(), assert_allclose(cell.
↪nucleus(plane=False), expected_results[i][j], rtol=1e-15, atol=0) == None)
N2 dart True
N1 skew_quad True
N0 dart True
Q0 quad True
Q1 quad True
Q2 quad True
R0 quad True
Q3 quad True
Q4 quad True
Q5 quad True
R3 quad True
```

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 rhealpixdggs.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_resolution = 1
    max_resolution = 15
    ellipsoid:
        R_A = 6371007.180918476
        a = 6378137.0
        b = 6356752.314245179
        e = 0.08181919084262149
        f = 0.003352810664747480 5
        lat_0 = -37
        lon_0 = 174
        radians = False
        sphere = False
>>> print(rdggs.cell_from_point(1, p, plane=False))
Q3
```

CHAPTER

TWO

THE UTILS MODULE

This Python 3.11 module implements several helper functions for coding map projections.

- Alexander Raichev (AR), 2012-01-26: Refactored code from release 0.3.

NOTE:

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

rhealpixdggs.utils.**auth_lat**(*phi: float*, *e: float*, *inverse: bool = False*, *radians: bool = False*) → float

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/3, 0.08181919104281579, radians=True)
>>> print(my_round(beta, 15))
1.045256493205824

>>> print(my_round(auth_lat(beta, 0.08181919104281579, radians=True,␣
↪inverse=True), 15))
1.047197551196598

>>> print(my_round(pi/3, 15))
1.047197551196598
```

NOTES:

For small flattenings f (f < 1/150), when calculating authalic from common latitude, power series approximation (from https://doi.org/10.48550/arXiv.2212.05818) gives more accurate results than direct formula (for double-precission accuracy). For the inverse, again power series approximation is used, which is standard in cartography for small flattenings. The one used in this case is from https://doi.org/10.48550/arXiv.2212.05818

rhealpixdggs.utils.**auth_rad**(*a: float*, *e: float*, *inverse: bool = False*) → float

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, 0.989393259670095)
(0.111111111111111, 0.997935147429943)
(0.0625, 0.999348236455825)
(0.04, 0.99973321235361)
```

(continues on next page)

(continued from previous page)

```
(0.027777777777778, 0.99987137105188)
(0.020408163265306, 0.999930576285614)
(0.015625, 0.999959307080847)
(0.012345679012346, 0.999974596271211)
(0.01, 0.999983332861089)
```

rhealpixdggs.utils.**my_round**(*x: Any*, *digits: int = 0*) → Any

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)
```

rhealpixdggs.utils.**wrap_latitude**(*phi: float*, *radians: bool = False*) → float

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(45.0, radians=False)
45.0
>>> wrap_latitude(-45.0, radians=False)
-45.0
>>> wrap_latitude(90.0, radians=False)
90.0
>>> wrap_latitude(-90.0, radians=False)
-90.0
>>> wrap_latitude(135.0, radians=False)
-45.0
>>> wrap_latitude(-135.0, radians=False)
45.0
```

rhealpixdggs.utils.**wrap_longitude**(*lam: float*, *radians: bool = False*) → float

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)
-3.141592653589793
```

CHAPTER

THREE

THE PJ_HEALPIX MODULE

This Python 3.11 module implements the HEALPix map projection as described in [CaRo2007].

- Alexander Raichev (AR), 2013-01-26: Refactored code from release 0.3.

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.

rhealpixdggs.pj_healpix.**healpix**(*a: float = 1*, *e: float = 0*) → Callable[[float, float, bool, bool], tuple[float, float]]

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(tuple(x.tolist() for x in my_round(f(0, pi/3, radians=True), 15)))
(0.574951359778215, 2.145747686573111)
>>> p = (0, 60)
>>> q = f(*p, radians=False); print(tuple(x.tolist() for x in my_round(q, 15)))
(0.574951359778215, 2.145747686573111)
>>> print(tuple(x.tolist() for x in my_round(f(*q, radians=False, inverse=True),␣
↪15)))
(6e-15, 59.999999999999986)
>>> print(my_round(p, 15))
(0, 60)
```

OUTPUT:

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

rhealpixdggs.pj_healpix.**healpix_ellipsoid**(*lam: float*, *phi: float*, *e: float = 0*) → tuple[float, float]

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(tuple(x if type(x) is int else x.tolist() for x in my_round(healpix_
↪ellipsoid(0, pi/7), 15)))
(0, 0.511157237746422)
>>> print(tuple(x if type(x) is int else x.tolist() for x in my_round(healpix_
```

(continues on next page)

(continued from previous page)

```
↪ellipsoid(0, pi/7, e=0.8), 15)))
(0, 0.268484450857837)
```

rhealpixdggs.pj_healpix.**healpix_ellipsoid_inverse**(*x: float*, *y: float*, *e: float = 0*) → tuple[float, float]

Compute the inverse of healpix_ellipsoid().

EXAMPLES:

```
>>> p = (0, pi/7)
>>> q = healpix_ellipsoid(*p)
>>> print(tuple(x if type(x) is int else x.tolist() for x in my_round(healpix_
↪ellipsoid_inverse(*q), 15)))
(0, 0.448798950512828)
>>> print(my_round(p, 15))
(0, 0.448798950512828)
```

rhealpixdggs.pj_healpix.**healpix_sphere**(*lam: float*, *phi: float*) → tuple[float, float]

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
```

rhealpixdggs.pj_healpix.**healpix_sphere_inverse**(*x: float*, *y: float*) → tuple[float, float]

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
```

rhealpixdggs.pj_healpix.**healpix_vertices**() → list[tuple[float, float, float]]

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

rhealpixdggs.pj_healpix.**in_healpix_image**(*x: float*, *y: float*) → bool

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),
```

(continues on next page)

(continued from previous page)

```
... (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 3.11 module implements the rHEALPix map projection.

- Alexander Raichev (AR), 2013-01-26: Refactored code from release 0.3.

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.

rhealpixdggs.pj_rhealpix.**combine_triangles**(*x: float*, *y: float*, *north_square: int = 0*, *south_square: int = 0*, *inverse: bool = False*) → tuple[float, float]

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(tuple(x.tolist() for x in my_round((x, y), 15)))
(-1.832595714594046, 1.570796326794896)
>>> print(tuple(x.tolist() for x in my_round(combine_triangles(x, y,␣
↪inverse=True), 15)))
(-0.785398163397448, 1.047197551196598)
>>> print(my_round((u, v), 15))
(-0.785398163397448, 1.047197551196598)
```

rhealpixdggs.pj_rhealpix.**in_rhealpix_image**(*x: float*, *y: float*, *north_square: int = 0*, *south_square: int = 0*) → bool

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),
```

(continues on next page)

(continued from previous page)

```
... (-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
```

rhealpixdggs.pj_rhealpix.**rhealpix**(*a: float = 1*, *e: float = 0*, *north_square: int = 0*, *south_square: int = 0*, *region: str = 'none'*) → Callable[[float, float, bool, bool], tuple[float, float]]

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(tuple(x.tolist() for x in my_round(f(0, pi/3, radians=True), 15)))
(-0.574951359778215, 2.145747686573111)
>>> p = (0, 60)
>>> q = f(*p, radians=False)
>>> print(tuple(x.tolist() for x in my_round(q, 15)))
(-0.574951359778215, 2.145747686573111)
>>> print(tuple(x.tolist() for x in my_round(f(*q, radians=False, inverse=True),␣
↪15)))
(6e-15, 59.999999999999986)
>>> print(my_round(p, 15))
(0, 60)
```

OUTPUT:

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

rhealpixdggs.pj_rhealpix.**rhealpix_ellipsoid**(*lam: float*, *phi: float*, *e: float = 0*, *north_square: int = 0*, *south_square: int = 0*, *region: str = 'none'*) → tuple[float, float]

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(tuple(x if type(x) is int else x.tolist() for x in my_round(rhealpix_
↪ellipsoid(0, arcsin(2.0/3)), 15)))
(0, 0.785398163397448)
```

rhealpixdggs.pj_rhealpix.**rhealpix_ellipsoid_inverse**(*x: float*, *y: float*, *e: float = 0*, *north_square: int = 0*, *south_square: int = 0*, *region: str = 'none'*) → tuple[float, float]

Compute the inverse of rhealpix_ellipsoid().

EXAMPLES:

```
>>> p = (0, pi/4)
>>> q = rhealpix_ellipsoid(*p)
>>> print(tuple(x.tolist() for x in my_round(rhealpix_ellipsoid_inverse(*q),␣
↪15)))
(0.0, 0.785398163397448)
>>> print(my_round(p, 15))
(0, 0.785398163397448)
```

rhealpixdggs.pj_rhealpix.**rhealpix_sphere**(*lam: float*, *phi: float*, *north_square: int = 0*, *south_square: int = 0*, *region: str = 'none'*) → tuple[float, float]

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(tuple(x.tolist() for x in 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 |
*---*---*---*---*
```

rhealpixdggs.pj_rhealpix.**rhealpix_sphere_inverse**(*x: float*, *y: float*, *north_square: int = 0*, *south_square: int = 0*, *region: str = 'none'*) → tuple[float, float]

Compute the inverse of rhealpix_sphere().

EXAMPLES:

```
>>> p = (0, pi/4)
>>> q = rhealpix_sphere(*p)
>>> print(tuple(x.tolist() for x in my_round(rhealpix_sphere_inverse(*q), 15)))
(0.0, 0.785398163397448)
>>> print(my_round(p, 15))
(0, 0.785398163397448)
```

rhealpixdggs.pj_rhealpix.**rhealpix_vertices**(*north_square: int = 0*, *south_square: int = 0*) → list[tuple[float, float]]

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

rhealpixdggs.pj_rhealpix.**triangle**(*x: float*, *y: float*, *north_square: int = 0*, *south_square: int = 0*, *inverse: bool = False*) → tuple[int, str]

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 3.11 code implements ellipsoids of revolution.

- Alexander Raichev (AR), 2012-01-26: Refactored code from release 0.3.

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 rhealpixdggs.ellipsoids.**Ellipsoid**(*R=None*, *a=6378137.0*, *b=None*, *e=None*, *f=0.0033528106647474805*, *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 ellipsoid parameters <https://proj.org/en/9.4/usage/ellipsoids.html#ellipsoid-spherification-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(tuple(x.tolist() for x in my_round(E.xyz(0, 45), 15)))
(0.707106781186548, 0.0, 0.707106781186548)
```

CHAPTER

SIX

THE PROJECTION_WRAPPER MODULE

This Python 3.11 module implements a wrapper for map projections.

- Alexander Raichev (AR), 2013-01-25: Refactored code from release 0.3.

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 rhealpixdggs.projection_wrapper.**Projection**(*ellipsoid=<rhealpixdggs.ellipsoids.Ellipsoid object>*, *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 rhealpixdggs.ellipsoids import WGS84_ELLIPSOID
>>> f = Projection(ellipsoid=WGS84_ELLIPSOID, proj='rhealpix', north_square=1,␣
↪south_square=0)
>>> print(tuple(x.tolist() for x in my_round(f(0, 30), 15)))
(0.0, 3740232.8933662786)
>>> f = Projection(ellipsoid=WGS84_ELLIPSOID, proj='cea')
>>> print(my_round(f(0, 30), 15))
(0.0, 3171259.315518537)
```

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 DGGS MODULE

This Python 3.11 module implements the rHEALPix discrete global grid system.

- Alexander Raichev (AR), 2012-11-12: Initial version based upon grids.py.

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 rhealpixdggs.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 = 6371007.180918476
        a = 6378137.0
        b = 6356752.314245179
        e = 0.08181919084262149
        f = 0.003352810664747480 5
        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

```
>>> expected_results = [
...     [
...         (1.90833280887811e-14, 58.52801748206219),
...         (45.00000000000002, 58.52801748206219),
...         (89.99999999999997, 58.52801748206219)
...     ], [
...         (14.999999999999998, 26.490118751439734),
...         (45.0, 26.490118751439734),
...         (74.99999999999999, 26.490118751439734),
...         (105.00000000000001, 26.490118751439734)
...     ], [
...         (14.999999999999998, 0),
...         (45.0, 0),
...         (74.99999999999999, 0),
...         (105.00000000000001, 0)
...     ]]
>>> for i, row in enumerate(cells):
...     for j, cell in enumerate(row):
...         print(cell, assert_allclose(cell.nucleus(plane=False), expected_
↪results[i][j], rtol=1e-15, atol=0) == None)
N8 True
N5 True
N2 True
Q0 True
Q1 True
Q2 True
```

(continues on next page)

(continued from previous page)

```
R0 True
Q3 True
Q4 True
Q5 True
R3 True
```

Create a (0, 0)-rHEALPix DGGS with N_side = 3 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 rhealpixdggs.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 = 6371007.180918476
        a = 6378137.0
        b = 6356752.314245179
        e = 0.08181919084262149
        f = 0.003352810664747480 5
        lat_0 = -37
        lon_0 = 174
        radians = False
        sphere = False

>>> print(rdggs.cell_from_point(1, p, plane=False))
Q3
```

class rhealpixdggs.dggs.**RHEALPixDGGS**(*ellipsoid=<rhealpixdggs.ellipsoids.Ellipsoid object>*, *N_side=3*, *north_square=0*, *south_square=0*, *max_areal_resolution=1*)

Bases: `object`

Represents an rHEALPix DGGS on a given ellipsoid.

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
>>> rdggs = RHEALPixDGGS(N_side=15)
>>> p = (80, -20)
>>> c = rdggs.cell_from_point(1, p, plane=False)
>>> print(c)
(Q, 178)
```

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, 14))
-1.02150660972679
-0.46233979145691
0.0
0.46233979145691
1.02150660972679

>>> for phi in rdggs.cell_latitudes(1, -pi/2, pi/2, nucleus=False,␣
↪plane=False):
...     print(my_round(phi, 14))
-1.29894395947616
-0.73195363195267
-0.22506566919844
0.22506566919844
0.73195363195267
1.29894395947616
```

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
```

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)
```

(continues on next page)

(continued from previous page)

```
>>> 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*, *region='none'*)

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(tuple(x.tolist() for x in rdggs.combine_triangles(*p)))
(0.0, 0.0)
>>> print(tuple(x.tolist() for x in my_round(rdggs.combine_triangles(*q),␣
↪14)))
(-2.35619449019234, 1.5707963267949)
```

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(tuple(x.tolist() for x in my_round(rdggs.healpix(-pi, pi/2), 14)))
(-2.35619449019234, 1.5707963267949)
```

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)
```

(continues on next page)

(continued from previous page)

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

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*, *region='none'*)

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(tuple(x.tolist() for x in my_round(rdggs.rhealpix(0, pi/3), 14)))
(-1.858272006684, 2.06871881030324)
```

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 pjr.triangle().

EXAMPLES:

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

>>> c = rdggs.cell(['N', 3])
```

(continues on next page)

(continued from previous page)

```
>>> 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(tuple(x.tolist() for x in my_round(rdggs.xyz(0, pi/4, lonlat=True),
↪ 14)))
(0.70710678118655, 0.0, 0.70710678118655)
```

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(tuple(x.tolist() for x in my_round(rdggs.xyz_cube(0, 0), 14)))
(0.78539816339745, 0.0, -0.78539816339745)
```

class rhealpixdggs.dggs.**RhealPolygon**(*rdggs=<rhealpixdggs.dggs.RHEALPixDGGS object>*, *suid_list=None*)

Bases: `object`

CHAPTER

EIGHT

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

r

INDEX

R

T

W

X