pygeogrids Documentation

Release

Christoph Paulik

May 12, 2015

pygeogrids is a package for creation and handling of Discrete Global Grids.

It can be used to define a grid on the globe using numpy arrays of longitude and latitude. These grids can also have unique grid point numbers. The grids must not be valid globally but can e.g. only cover the Continents.

When a grid is defined it can be used to quickly find the nearest neigbor of a a given lat, lon coordinate on the grid. For that the lon, lat coordinates are converted to Cartesian coordinates. This approach is of limited use for high resolution data which might rely on a specific geodetic datum.

The class `pygeogrids.grids.CellGrid` extends this basic grid with the ability to store a additional cell number for each grid point. This can be used to tile a grid in e.g. 5x5° cells. We often store remote sensing data in cells to partition a dataset into manageable parts. This link with the grid class enables us to easily find the link between a grid point and the cell file in which the relevant data is stored.

Please see the examples in this documentation as well as the pytesmo code for real world usage examples.

CHAPTER 1

Contents

1.1 Examples

```
import pygeogrids.grids as grids
import numpy as np
```

Let's create a simple regular 10x10 degree grid with grid points at the center of each 10x10 degree cell.

First by hand to understand what is going on underneath

```
# create the longitudes
lons = np.arange(-180 + 5, 180, 10)
print(lons)
lats = np.arange(90 - 5, -90, -10)
print(lats)
```

```
[-175 -165 -155 -145 -135 -125 -115 -105  -95  -85  -75  -65  -55  -45  -35
  -25  -15   -5    5   15   25   35   45   55   65   75   85   95  105  115
  125  135  145  155  165  175]
[ 85  75  65  55  45  35  25  15   5  -5 -15 -25 -35 -45 -55 -65 -75 -85]
```

These are just the dimensions or we can also call them the "sides" of the array that defines all the gridpoints.

```
# create all the grid points by using the numpy.meshgrid function
longrid, latgrid = np.meshgrid(lons, lats)
```

now we can create a BasicGrid. We can also define the shape of the grid. The first part of the shape must be in longitude direction.

```
manualgrid = grids.BasicGrid(longrid.flatten(), latgrid.flatten(), shape=(36, 18))

# Each point of the grid automatically got a grid point number
gpis, gridlons, gridlats = manualgrid.get_grid_points()
print(gpis[:10], gridlons[:10], gridlats[:10])
```

```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]), array([-175, -165, -155, -145, -135, -125, -115, -105,  -95,
```

The grid point indices or numbers are useful when creating lookup tables between grids.

We can now use the manualgrid instance to find the nearest gpi to any longitude and latitude

```
ngpi, distance = manualgrid.find_nearest_gpi(15.84, 28.76)
print(ngpi, distance)
```

```
# convert the gpi to longitude and latitude
print(manualgrid.gpi2lonlat(ngpi))
```

```
(235, array([ 426227.83684784]))
(15, 25)
```

The same grid can also be created by a method for creating regular grids

```
autogrid = grids.genreg_grid(10, 10)
autogrid == manualgrid
```

```
True
```

If your grid has a 2D shape like the ones we just created then you can also get the row and the column of a grid point. This can be useful if you know that you have data stored on a specific grid and you want to read the data from a grid point.

```
row, col = autogrid.gpi2rowcol(ngpi)
print(row, col)
```

```
(6, 19)
```

1.1.1 Iteration over gridpoints

```
for i, (gpi, lon, lat) in enumerate(autogrid.grid_points()):
    print(gpi, lon, lat)
    if i==10: # this is just to keep the example output short
        break
```

```
(0, -175.0, 85.0)
(1, -165.0, 85.0)
(2, -155.0, 85.0)
(3, -145.0, 85.0)
(4, -135.0, 85.0)
(5, -125.0, 85.0)
(6, -115.0, 85.0)
(7, -105.0, 85.0)
(8, -95.0, 85.0)
(9, -85.0, 85.0)
(10, -75.0, 85.0)
```

1.1.2 Calculation of lookup tables

If you have a two grids and you know that you want to get the nearest neighbors for all of its grid points in the second grid you can calculate a lookup table once and reuse it later.

```
# lets generate a second grid with 10 random points on the Earth surface.

randlat = np.random.random(10) * 180 - 90
randlon = np.random.random(10) * 360 - 180
print(randlat)
print(randlon)
# This grid has no meaningful 2D shape so none is given
randgrid = grids.BasicGrid(randlon, randlat)
```

```
[ -1.54836122e+01  -1.35887580e+00  -4.78383560e-02   4.78709682e+00
   1.11369002e+01   3.12077283e+01  -8.71094503e+01   2.20725149e+00
   6.64563916e+01   7.87705109e+01]
[ 121.05539154  111.61193334 -131.22315478  -46.73641204   15.1126928
 -165.14751769 -115.24386825  -56.14158745    9.9695647   143.61467711]
```

Now lets calculate a lookup table to the regular 10x10° grid we created earlier

```
lut = randgrid.calc_lut(autogrid)
print(lut)
```

```
[390 353 328 301 271 181 618 300  90  68]
```

The lookup table contains the grid point indices of the other grid, autogrid in this case.

```
lut_lons, lut_lats = autogrid.gpi2lonlat(lut)
print(lut_lats)
print(lut_lons)
```

```
[-15.  -5.  -5.   5.  15.  35. -85.   5.  65.  75.]
[ 125.  115. -135.  -45.   15. -165. -115.  -55.    5.  145.]
```

1.1.3 Storing and loading grids

Grids can be stored to disk as CF compliant netCDF files

```
import pygeogrids.netcdf as nc
nc.save_grid('example.nc', randgrid)
```

```
loadedgrid = nc.load_grid('example.nc')
```

```
loadedgrid == randgrid
```

```
True
```

1.2 License

1.3 pygeogrids

1.3.1 pygeogrids package

Submodules

pygeogrids.grids module

pygeogrids.nearest_neighbor module

pygeogrids.netcdf module

Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search