ESCI 379 – Python Programming for Advanced Earth Sciences Applications
Lesson 9 – Maps and 3-D Plots

BASEMAP

- **`Basemap` is a `matplotlib` toolkit for plotting data on geographically-referenced map projections.**
- **The documentation for `basemap` can be found at http://matplotlib.github.com/basemap**
- **The first step to using `basemap` is to import it using the command**
  `from mpl_toolkits.basemap import Basemap`
- **We then create a map object by typing**
  `m = Basemap(projection = *proj_type*, *kwds/args*)`
  where *`proj_type`* is a valid projection type, and *`kwds/args`* are additional keywords and arguments (which depend on the projection type chosen).
- **As an example we will create an orthographic projection using the example code below:**

```
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
import numpy as np
m = Basemap(projection = 'ortho',lat_0 = 40, lon_0 = -80)
m.drawmapboundary(fill_color = 'white')
m.drawcoastlines(color = 'black',linewidth = 0.5)
m.fillcontinents(color = '0.85')
m.drawparallels(np.arange(-90, 91,30))
m.drawmeridians(np.arange(-180,180,30))
plt.show()
```

which creates the plot

- In this example, `proj_type` is `'ortho'` which is short for 'orthographic'.
- `lat_0` and `lon_0` are the central latitude and longitude.
- The map methods `drawmapboundary()`, `drawcoastlines()`, `fillcontinents()`, `drawparallels()`, and `drawmeridians()` are fairly self-evident, but we will revisit them shortly.

- There are many different projections available to `basemap`. The table below shows the available projections.

Projection	proj_type
Azimuthal Equidistant	`'aeqd'`
Polyconic	`'poly'`
Gnomonic	`'gnom'`
Mollweide	`'moll'`
Transverse Mercator	`'tmerc'`
North-Polar Lambert Azimuthal	`'nplaea'`
Gall Stereographic Cylindrical	`'gall'`
Miller Cylindrical	`'mill'`
Mercator	`'merc'`
Sterographic	`'stere'`
North-Polar Stereographic	`'npstere'`
Hammer	`'hammer'`
Geostationary	`'geos'`
Near-Sided Perspective	`'nsper'`
van der Grinten	`'vandg'`
Lambert Azimuthal Equal Area	`'laea'`
McBryde-Thomas Flat-Polar Quartic	`'mbtfpq'`
Sinusoidal	`'sinu'`
South-Polar Stereographic	`'spstere'`
Lambert Conformal	`'lcc'`
North-Polar Azimuthal Equidistant	`'npaeqd'`
Equidistant Conic	`'eqdc'`

Cylindrial Equidistant	**`cyl'**
Oblique Mercator	**`omerc'**
Albers Equal Area	**`aea'**
South-Polar Azimuthal Equidistant	**`spaeqd'**
Orthographic	**`ortho'**
Cassini-Soldner	**`cass'**
South-Polar Lambert Azimuthal	**`splaea'**
Robinson	**`robin'**

- **Depending on the map projection the region is specified by either the following keywords:**
    - **lon_0 – center longitude (degrees)**
    - **lat_0 – center latitude (degrees)**
    - **width – width of domain (meters)**
    - **height - height of domain (meters)**

  or
    - **llcrnrlon – lower-left corner longitude (degrees)**
    - **llcrnrlat – lower-left corner latitude (degrees)**
    - **urcrnrlon – upper-right corner longitude (degrees)**
    - **urcrnrlat – upper-right corner latitude (degrees)**
- **There are many additional basemap keywords used for controlling the drawing of the map and setting up projections. Some of these are:**

Keyword	**Values**	**Purpose**
resolution	**c, l, i, h, or f**	**Resolution of the database for continents, lakes, etc. Initials stand for crude, low, intermediate, high, and full. The default is crude.**
area_thresh	**number representing square kilometers**	**Will not draw lakes or coastal features that have an area smaller than this threshold**
rsphere	**radius of the globe in meters**	**Defaults to 6370997. Can be changed, or even given as major and minor axes for plotting on an ellipsoid.**

MAP METHODS FOR DRAWING FEATURES ON MAPS

- **There are many methods for plotting coastlines, continents, rivers, etc. Some of these are summarized below.**
  - **For many of the methods the `linewidth` and `color` can be specified, but not the line type. Solid is often the only option.**
- `drawcoastlines()` **– Draws coastlines.**
- `drawcountries()` **– Draws country boundaries.**
- `drawgreatcircle(lon1, lat1, lon2, lat2, del_s = f)` **– Draws a greatcircle between two lat/lon pairs.** `del_s` **is the spacing (in km) between points.**
- `drawmapboundary()` **– Draws boundary around map projection. The fill color is specified with the keyword** `fill_color`**.**
- `drawmapscale(lon, lat, lon0, lat0, length)` **– Draws a scale at the position given by** `lon`**,** `lat`**. The distance is for the position of** `lon0`**,** `lat0`**. Additional keywords are:**
  - `units` **– The units for the scale ('km' is default)**
  - `barstyle` **–** `'simple'` **or** `'fancy'`
  - `fontsize` **– default is 9**
  - `color` **– default is** `'black'`
  - `labelstype` **–** `'simple'` **or** `'fancy'`
  - `format` **– a string format statement of the type** `'%3.1f'` **and such.**
  - `yoffset` **– controls the scale height and annotation placement.**
  - `fillcolor1, fillcolor2` **– controls colors for alternating regions of scale for** `'fancy'` **style.**
- `drawmeridians(mlist)` **– Draws meridians with values given by** `mlist`**. In addition to** `color` **and** `linewidth`**, additional keywords are:**
  - `dashes` **– Pattern for dashed lines, of the form** `[on,off]` **where** `on` **is the number of adjacent pixels turned on, while** `off` **is the number that are turned off. The default is** `[1,1]`**.**

- `labels` – Four values in the form `[left, right, top, bottom]` that control meridian labeling. These are Boolean in that 1 is on and 0 is off. So, `[1,0,1,0]` would label the meridians at the left and top of the plot.
- `labelstyle` – Controls whether labels use +/- or E/W. The default is E/W unless `labelstyle = '+/-'`.
- `fmt` – This formats the labels using the format statements of the type `'%3.1f'` and such.
- `xoffset, yoffset` – Label offsets from edge of map.
- `latmax` – Controls maximum latitude for drawing meridians (default is 80).

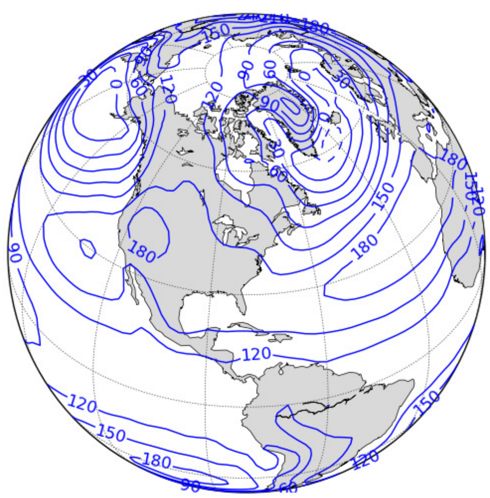
- `drawparallels(plist) )` – Draws latitude parallels with values given by `mlist`. Keywords are essentially the same as for `drawmeridians()`.
- `drawrivers()` – Draws rivers on map.
- `drawstates()` – Draws state boundaries.
- `fillcontinents(color = 'brown', lake_color = 'blue')` – Fills continents and lakes with specified colors.
- There is also a method for reading a GIS shapefile. This method is called `readshapefile()`. Details of its use can be found in the online documentation.

PLOTTING DATA ON MAPS

- Data can be plotted on maps by using the `contour()`, `contourf()`, `plot()`, `quiver()`, `barbs()`, and `drawgreatcircle()` methods for map objects.
  - These methods work pretty much just like the corresponding axes methods, with some exceptions.
- When plotting on the maps the latitudes and longitudes have to be converted into map coordinates.
  - This is accomplished by calling the map object with the longitude and latitude as arguments. This returns the x and y coordinates on the map projection.
    `x,y = m(lon,lat)`
  - To go from map coordinates back to longitude and latitude we use the `inverse` keyword.
    `lon, lat = m(x, y, inverse = True)`

- **The example below reads in 1000 mb height data, latitudes, and longitudes from a numpy NPZ file (`jan1000mb.npz`, available on the class website) and plots them on an orthographic projection using `contour()`.**

```
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
import numpy as np
m = Basemap(projection = 'ortho',lat_0 = 40, lon_0 = -80)
m.drawmapboundary(fill_color = 'white')
m.drawcoastlines(color = 'black',linewidth = 0.5)
m.fillcontinents(color = '0.85')
m.drawparallels(np.arange(-90, 91,30), color = '0.25', linewidth = 0.5)
m.drawmeridians(np.arange(-180,180,30), color = '0.25', linewidth = 0.5)
data = np.load('jan1000mb.npz')
lon = data['lon']
lat = data['lat']
z = data['z']
x,y = m(lon,lat)
cs = m.contour(x,y,z, levels = range(-180,360,30),colors = 'blue')
plt.clabel(cs, fmt = '%.0f', inline = True)
plt.show()
```

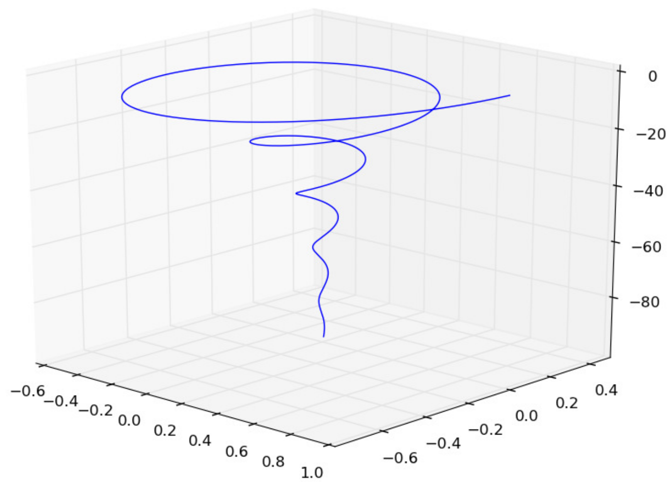


3-D PLOTTING

- **3-D plotting is accomplished using the `mpl_toolkits.mplot3d.axes3d` module.**
- **The functionality of this module is not complete, but is being worked on. Some features are not fully implemented.**
- **3-D plots can be rotated and viewed from different angles.**
- **The example below shows how to plot a 3-D spiraling line.**

```
import numpy as np
import matplotlib.pyplot as plt
```

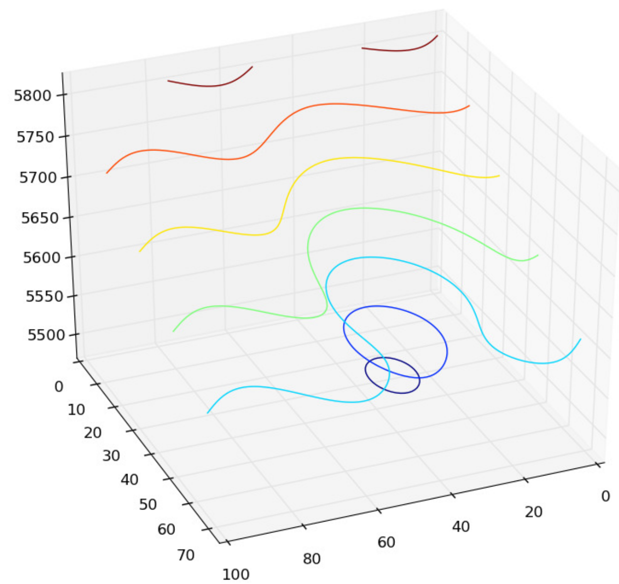
```
import mpl_toolkits.mplot3d.axes3d as ax3d
z = np.arange(0,-100.0, -0.1)
x = np.exp(z/20.0)*np.cos(2*np.pi*z/20.0)
y = np.exp(z/20.0)*np.sin(2*np.pi*z/20.0)
fig = plt.figure()
a = ax3d.Axes3D(fig)
a.plot(x,y,z)
plt.show()
```



- **The examples below show how to plot a 3-D contour plot, a 3-D surface plot, and a 3-D wireframe plot with contours overlain. Details can be found in the online documentation. All the examples use the `heights.npy` data file from the class website.**
- **3-D contour plot:**

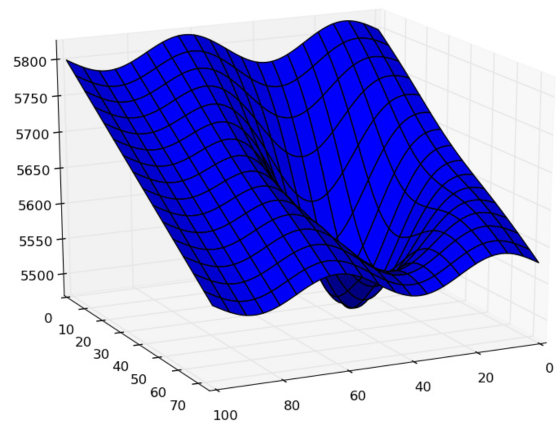
```
import numpy as np
import matplotlib.pyplot as plt
import mpl_toolkits.mplot3d.axes3d as ax3d
h = np.load('heights.npy')
shp = np.shape(h)
x = np.zeros_like(h)
y = np.zeros_like(h)
for i in range(0,shp[0]):
    for j in range(0, shp[1]):
        x[i,j] = i
        y[i,j] = j
fig = plt.figure()
a = ax3d.Axes3D(fig)
```

```
a.contour(x,y,h)
plt.show()
```



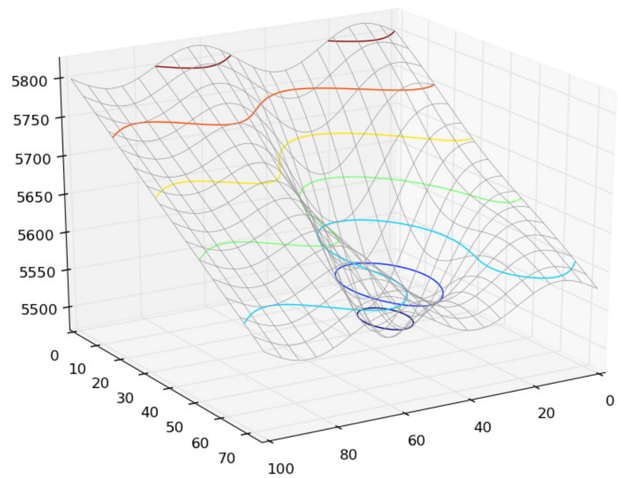
- **3-D surface plot:**

```
import numpy as np
import matplotlib.pyplot as plt
import mpl_toolkits.mplot3d.axes3d as ax3d
h = np.load('heights.npy')
shp = np.shape(h)
x = np.zeros_like(h)
y = np.zeros_like(h)
for i in range(0,shp[0]):
    for j in range(0, shp[1]):
        x[i,j] = i
        y[i,j] = j
fig = plt.figure()
a = ax3d.Axes3D(fig)
a.plot_surface(x,y,h,rstride = 5,cstride = 5)
plt.show()
```



- **3-D wireframe plot with contours:**

```
import numpy as np
import matplotlib.pyplot as plt
import mpl_toolkits.mplot3d.axes3d as ax3d
h = np.load('heights.npy')
shp = np.shape(h)
x = np.zeros_like(h)
y = np.zeros_like(h)
for i in range(0,shp[0]):
    for j in range(0, shp[1]):
        x[i,j] = i
        y[i,j] = j
fig = plt.figure()
a = ax3d.Axes3D(fig)
a.plot_wireframe(x,y,h,rstride = 5,cstride = 5,linewidth =
0.5, color = '0.6')
a.contour(x,y,h,linewidth = 2)
plt.show()
```



5800
5750
5700
5650
5600
5550
5500
0
10
20
30
40
50
60
70
100
80
60
40
20
0