Geographic Data with Basemap

Geographic data visualization is a prominent tool in the field of data science. The Basemap toolkit, one of many available in Matplotlib's mpl toolkits namespace, is the primary tool for this kind of display. Basemap isn't the most intuitive tool, and it may take a lot longer to display even very basic visualizations than you would expect. For more involved map renderings, more up-to-date options like leaflet or the Google Maps API may be preferable. Even so, Python programmers might benefit from familiarity with Basemap. Here we'll demonstrate some of the many kinds of map visualization that may be achieved with this set of tools.

Installation of Basemap is straightforward; if you're using conda you can type this and the package will be downloaded:

```
$ conda install basemap
```

We add just a single new import to our standard boilerplate:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
```

Once you have the Basemap toolkit installed and imported, geographic plots are just a few lines away (the graphics in the following also requires the `PIL` package in Python 2, or the `pillow` package in Python 3):

```
plt.figure(figsize=(8, 8))
m = Basemap(projection='ortho', resolution=None, lat_0=50, lon_0=-100)
m.bluemarble(scale=0.5);
```

The meaning of the arguments to `Basemap` will be discussed momentarily.

The useful thing is that the globe shown here is not a mere image; it is a fully-functioning Matplotlib axes that understands spherical coordinates and which allows us to easily overplot data on the map! For example, we can use a different map projection, zoom-in to North America and plot the location of Seattle. We'll use an etopo image (which shows topographical features both on land and under the ocean) as the map background:

```
fig = plt.figure(figsize=(8, 8))
m = Basemap(projection='lcc', resolution=None,
            width=8E6, height=8E6,
            lat_0=45, lon_0=-100,)
m.etopo(scale=0.5, alpha=0.5)

# Map (long, lat) to (x, y) for plotting
x, y = m(-122.3, 47.6)
plt.plot(x, y, 'ok', markersize=5)
plt.text(x, y, ' Seattle', fontsize=12);
```

This is just a taste of the kinds of geographical visualizations that can be created with relatively few lines of Python code. Following this, we'll go over Basemap's capabilities in greater detail and show you how to visualize map data in a variety of ways. You should be able to construct just about any kind of map visualization you can think of using the aforementioned simple examples as guides.

Map Projections

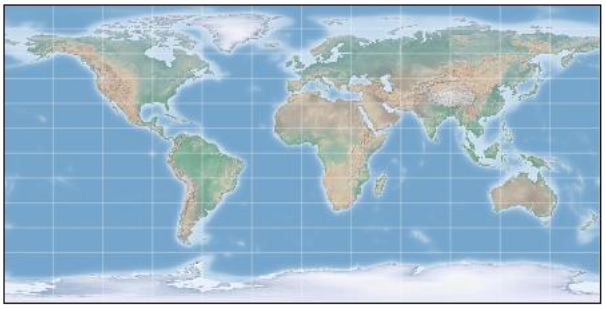
What projection to use is the first consideration when working with maps. You probably already know that attempting to project a spherical map, like Earth, onto a flat surface will inevitably result in distortion and/or a break in the map's continuity. Humanity has developed a wide variety of projections over time. Certain map characteristics (such as direction, area, distance, shape, and so on) are helpful to preserve, depending on the map's intended use.
Several dozen of these projections are implemented in the Basemap package and can be accessed via a short format code. Some of the more common ones will be briefly shown here.
To begin, we'll establish a shorthand procedure for sketching our world map with the longitude and latitude lines:

```
from itertools import chain

def draw_map(m, scale=0.2):
    # draw a shaded-relief image
    m.shadedrelief(scale=scale)

    # lats and longs are returned as a dictionary
    lats = m.drawparallels(np.linspace(-90, 90, 13))
    lons = m.drawmeridians(np.linspace(-180, 180, 13))

    # keys contain the plt.Line2D instances
    lat_lines = chain(*(tup[1][0] for tup in lats.items()))
    lon_lines = chain(*(tup[1][0] for tup in lons.items()))
    all_lines = chain(lat_lines, lon_lines)

    # cycle through these lines and set the desired style
    for line in all_lines:
        line.set(linestyle='-', alpha=0.3, color='w')
```

Cylindrical projections

Cylindrical projections are the simplest kind of map projection since they simply convert lines of constant latitude and longitude to horizontal and vertical lines. While the equatorial areas are accurately shown, the poles suffer from severe distortions on this style of map. Cylindrical projections differ in their distortion near the poles and in their conservation properties due to the variation in the spacing of latitude lines. Equidistant cylindrical projections, like the one seen in the accompanying picture, use a latitude scale that maintains distances along meridians. The Mercator (projection='merc') and Cylindrical Equal Area (projection='cea') Projection are two other examples of cylindrical projections..

```
fig = plt.figure(figsize=(8, 6), edgecolor='w')
m = Basemap(projection='cyl', resolution=None,
            llcrnrlat=-90, urcrnrlat=90,
            llcrnrlon=-180, urcrnrlon=180, )
draw_map(m)
```



The additional arguments to Basemap for this view specify the latitude (`lat`) and longitude (`lon`) of the lower-left corner (`llcrnr`) and upper-right corner (`urcrnr`) for the desired map, in units of degrees.

Pseudo-cylindrical projections

Pseudo-cylindrical projections relax the requirement that meridians (lines of constant longitude) remain vertical; this can give better properties near the poles of the projection. The Mollweide projection (`projection='moll'`) is one common example of this, in which all

meridians are elliptical arcs. It is constructed so as to preserve area across the map: though there are distortions near the poles, the area of small patches reflects the true area. Other pseudo-cylindrical projections are the sinusoidal (`projection='sinu'`) and Robinson (`projection='robin'`) projections.

```
fig = plt.figure(figsize=(8, 6), edgecolor='w')
m = Basemap(projection='moll', resolution=None,
            lat_0=0, lon_0=0)
draw_map(m)
```

The extra arguments to Basemap here refer to the central latitude (`lat_0`) and longitude (`lon_0`) for the desired map.

Perspective projections

Perspective projections are constructed using a particular choice of perspective point, similar to if you photographed the Earth from a particular point in space (a point which, for some projections, technically lies within the Earth!). One common example is the orthographic projection (`projection='ortho'`), which shows one side of the globe as seen from a viewer at a very long distance. As such, it can show only half the globe at a time. Other perspective-based projections include the gnomonic projection (`projection='gnom'`) and stereographic projection (`projection='stere'`). These are often the most useful for showing small portions of the map.

Here is an example of the orthographic projection:

```
fig = plt.figure(figsize=(8, 8))
m = Basemap(projection='ortho', resolution=None,
            lat_0=50, lon_0=0)
draw_map(m);
```

Conic projections

With a Conic projection, the map is projected onto a single cone, which is then unrolled. This can result in excellent neighborhood properties, but it can cause significant distortions in areas outside the cone's center of mass. The North American map we just looked at was projected using the Lambert Conformal Conic projection (projection='lcc'). Scale decreases between standard parallels (specified in Basemap by lat 1 and lat 2) and increases outside of them, and the map is projected onto a cone in such a way that this is accurately represented. The equidistant conic projection (projection='eqdc') and Albers equal-area projection (projection='aea') are two other helpful conic projections. Similar to perspective projections, conic projections work well when depicting relatively small to moderate regions of the Earth.

```
fig = plt.figure(figsize=(8, 8))
m = Basemap(projection='lcc', resolution=None,
            lon_0=0, lat_0=50, lat_1=45, lat_2=55,
            width=1.6E7, height=1.2E7)
draw_map(m)
```

Other projections

There are many different projections available, each with their own set of properties and benefits and drawbacks; if you plan to work extensively with map-based visualizations, you should familiarize yourself with these. The Basemap package probably includes them. There is a fantastic subculture of geo-viz geeks who will passionately defend their preferred projection for any given use case if you delve deeply enough into this topic.

Drawing a Map Background

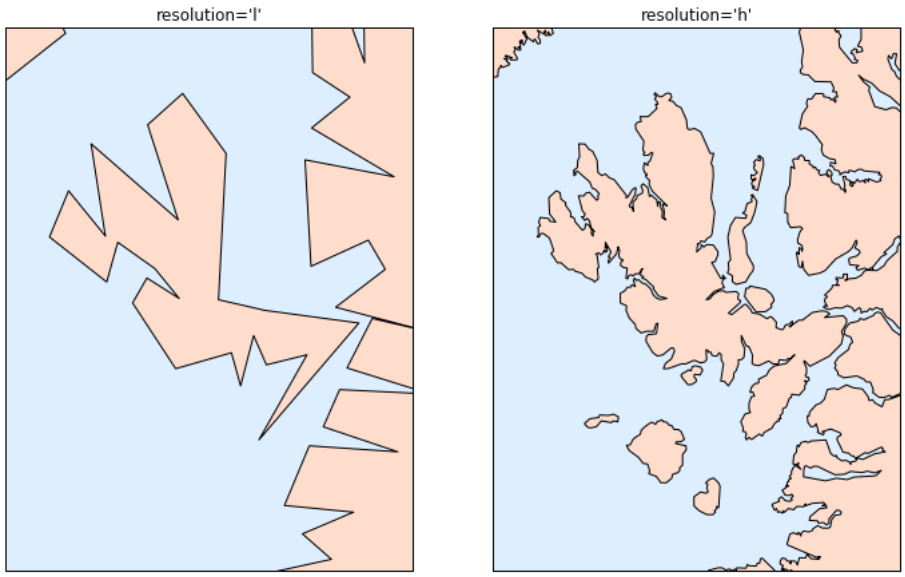
Earlier we saw the `bluemarble()` and `shadedrelief()` methods for projecting global images on the map, as well as the `drawparallels()` and `drawmeridians()` methods for drawing lines of constant latitude and longitude. The Basemap package contains a range of useful functions for drawing borders of physical features like continents, oceans, lakes, and rivers, as well as political boundaries such as countries and US states and counties. The following are some of the available drawing functions that you may wish to explore using IPython's help features:

- **Physical boundaries and bodies of water**
  - `drawcoastlines()`: Draw continental coast lines
  - `drawlsmask()`: Draw a mask between the land and sea, for use with projecting images on one or the other
  - `drawmapboundary()`: Draw the map boundary, including the fill color for oceans.
  - `drawrivers()`: Draw rivers on the map
  - `fillcontinents()`: Fill the continents with a given color; optionally fill lakes with another color
- **Political boundaries**
  - `drawcountries()`: Draw country boundaries
  - `drawstates()`: Draw US state boundaries
  - `drawcounties()`: Draw US county boundaries
- **Map features**
  - `drawgreatcircle()`: Draw a great circle between two points
  - `drawparallels()`: Draw lines of constant latitude
  - `drawmeridians()`: Draw lines of constant longitude
  - `drawmapscale()`: Draw a linear scale on the map
- **Whole-globe images**
  - `bluemarble()`: Project NASA's blue marble image onto the map
  - `shadedrelief()`: Project a shaded relief image onto the map
  - `etopo()`: Draw an etopo relief image onto the map
  - `warpimage()`: Project a user-provided image onto the map

For the boundary-based features, you must set the desired resolution when creating a Basemap image. The `resolution` argument of the `Basemap` class sets the level of detail in boundaries, either `'c'` (crude), `'l'` (low), `'i'` (intermediate), `'h'` (high), `'f'` (full), or `None` if no boundaries will be used. This choice is important: setting high-resolution boundaries on a global map, for example, can be *very* slow.

Here's an example of drawing land/sea boundaries, and the effect of the resolution parameter. We'll create both a low- and high-resolution map of Scotland's beautiful Isle of Skye. It's located at 57.3°N, 6.2°W, and a map of 90,000 × 120,000 kilometers shows it well:

```
fig, ax = plt.subplots(1, 2, figsize=(12, 8))

for i, res in enumerate(['l', 'h']):
    m = Basemap(projection='gnom', lat_0=57.3, lon_0=-6.2,
                width=90000, height=120000, resolution=res, ax=ax[i])
    m.fillcontinents(color="#FFDDCC", lake_color='#DDEEFF')
    m.drawmapboundary(fill_color="#DDEEFF")
    m.drawcoastlines()
    ax[i].set_title("resolution='{0}'".format(res));
```



Notice that the low-resolution coastlines are not suitable for this level of zoom, while high-resolution works just fine. The low level would work just fine for a global view, however, and would be *much* faster than loading the high-resolution border data for the entire globe! It might require some experimentation to find the correct resolution parameter for a given view: the best route is to start with a fast, low-resolution plot and increase the resolution as needed.

Plotting Data on Maps

Perhaps the most useful piece of the Basemap toolkit is the ability to over-plot a variety of data onto a map background. For simple plotting and text, any `plt` function works on the map; you can use the `Basemap` instance to project latitude and longitude coordinates to `(x, y)` coordinates for plotting with `plt`, as we saw earlier in the Seattle example.

In addition to this, there are many map-specific functions available as methods of the `Basemap` instance. These work very similarly to their standard Matplotlib counterparts, but have an additional Boolean argument `latlon`, which if set to `True` allows you to pass raw latitudes and longitudes to the method, rather than projected `(x, y)` coordinates.

Some of these map-specific methods are:

- `contour()`/`contourf()` : Draw contour lines or filled contours
- `imshow()`: Draw an image
- `pcolor()`/`pcolormesh()` : Draw a pseudocolor plot for irregular/regular meshes
- `plot()`: Draw lines and/or markers.
- `scatter()`: Draw points with markers.
- `quiver()`: Draw vectors.
- `barbs()`: Draw wind barbs.
- `drawgreatcircle()`: Draw a great circle.

We'll see some examples of a few of these as we continue. For more information on these functions, including several example plots,.