Maps with Cartopy (./maps-with-cartopy.html)

Date Thu 15 November 2018 Tags python (./tag/python.html) / matplotlib (./tag/matplotlib.html) / cartopy (./tag/cartopy.html) / xarray (./tag/xarray.html)

Maps in Scientific Python

Making maps is a fundamental part of geoscience research. Maps differ from regular figures in the following principle ways:

- Maps require a *projection* of geographic coordinates on the 3D Earth to the 2D space of your figure.
- Maps often include extra decorations besides just our data (e.g. continents, country borders, etc.)

Mapping is a notoriously hard and complicated problem, mostly due to the complexities of projection. In this lecture, we will learn about Cartopy (https://scitools.org.uk/cartopy/docs/latest/), one of the most common packages for making maps within python. Another popular and powerful library is Basemap (https://matplotlib.org/basemap/); however, Basemap is going away (https://matplotlib.org/basemap/users/intro.html#cartopy-new-management-and-eol-announcement) and being replaced with Cartopy in the near future. For this reason, new python learners are recommended to learn Cartopy.

Credit: Phil Elson

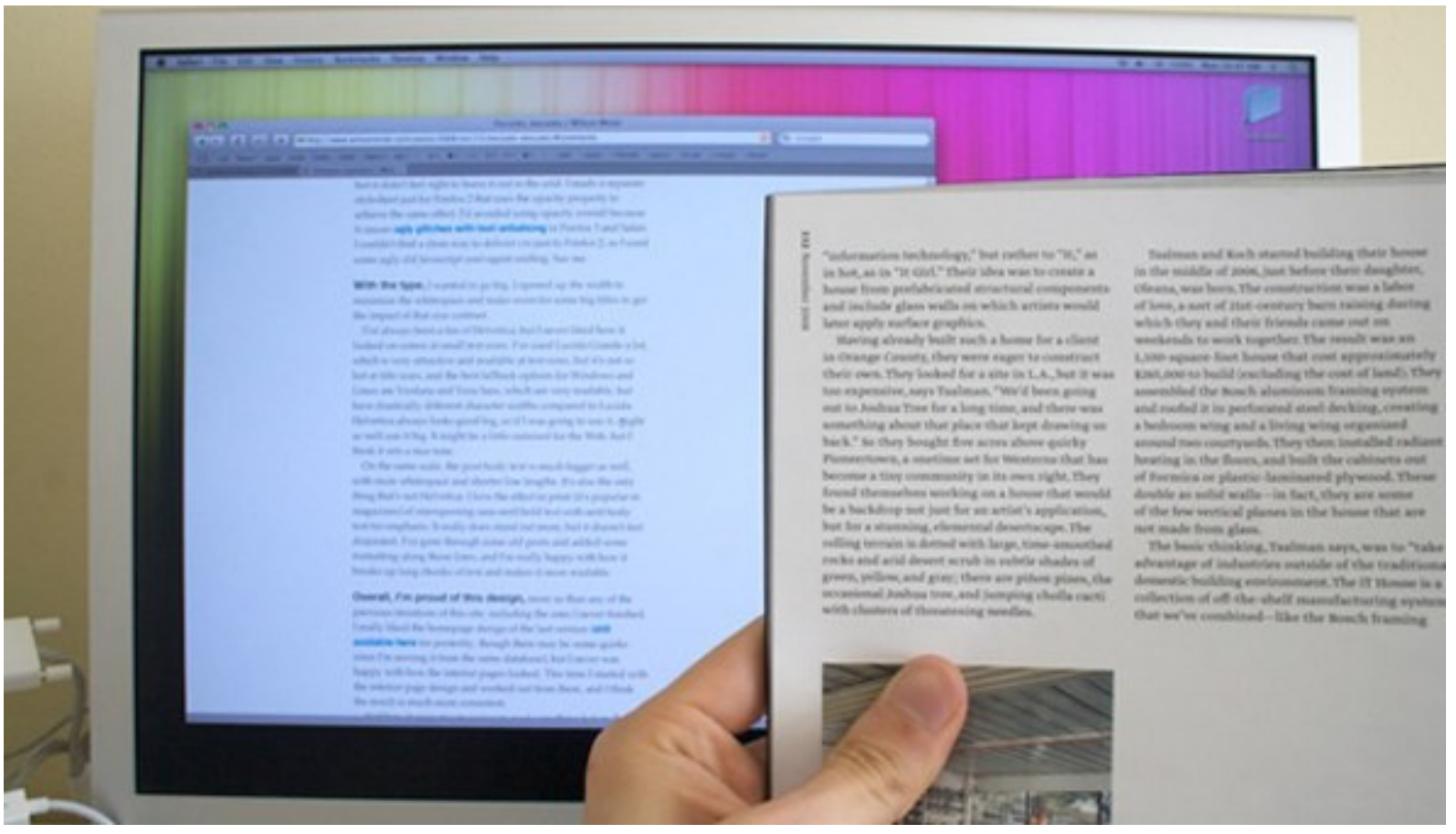
Lots of the material in this lesson was adopted from Phil Elson (https://pelson.github.io/)'s excellent Cartopy Tutorial (https://github.com/SciTools/cartopy-tutorial). Phil is the creator of Cartopy and published his tutorial under an open license (http://www.nationalarchives.gov.uk/doc/open-government-licence/version/3/), meaning that we can copy, adapt, and redistribute it as long as we give proper attribution. **THANKS PHIL!** 👏👏👏

Background: Projections

Most of our media for visualization *are* flat

Our two most common media are flat:

- Paper
- Screen

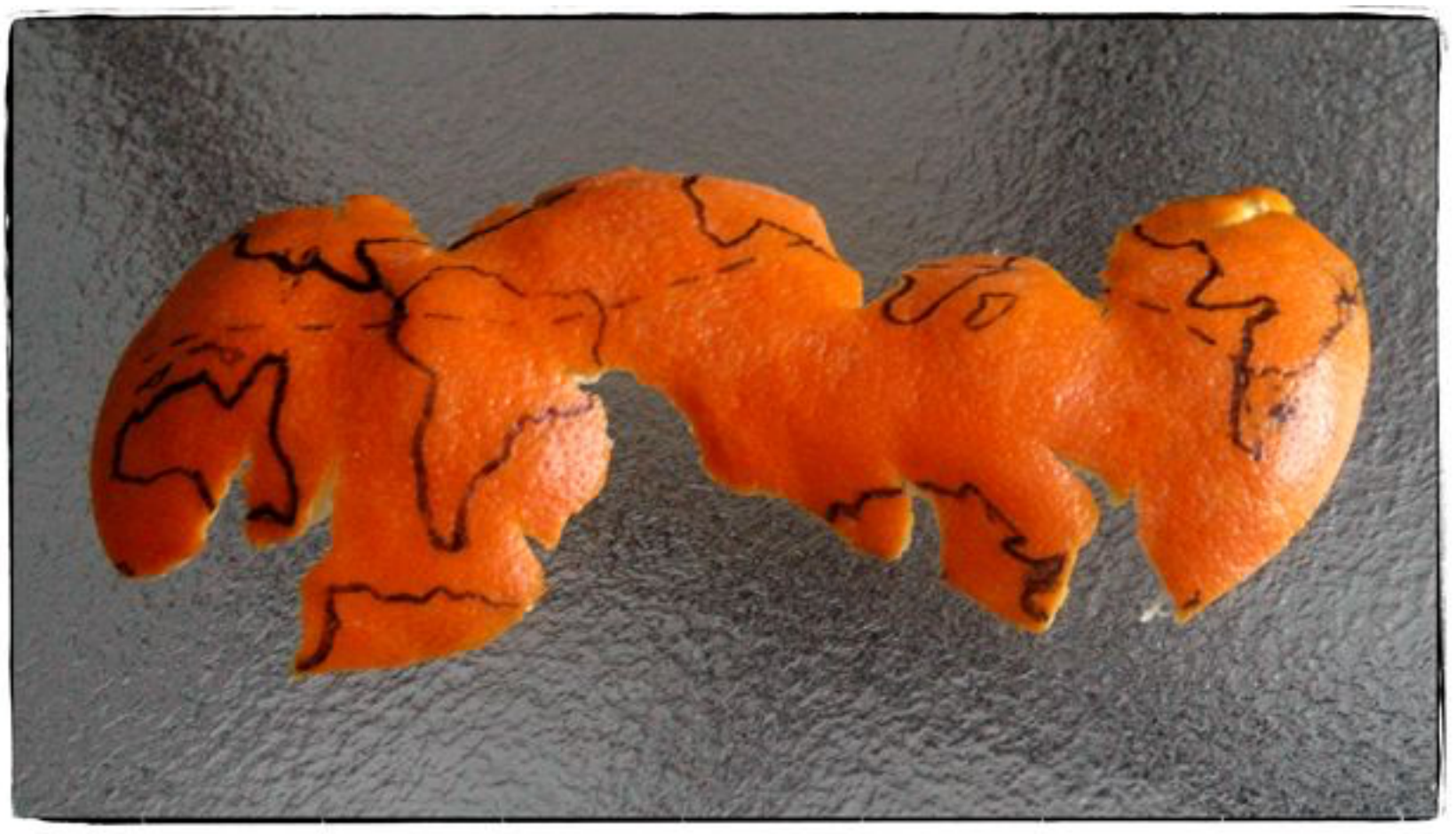


[Map] Projections: Taking us from spherical to flat

A map projection (or more commonly refered to as just "projection") is:

> a systematic transformation of the latitudes and longitudes of locations from the surface of a sphere or an ellipsoid into locations on a plane. [Wikipedia: Map projection (https://en.wikipedia.org/wiki/Map_projection)].

The major problem with map projections



- The surface of a sphere is topologically different to a 2D surface, therefore we *have* to cut the sphere *somewhere*
- A sphere's surface cannot be represented on a plane without distortion.

There are many different ways to make a projection, and we will not attempt to explain all of the choices and tradeoffs here. Instead, you can read Phil's original tutorial (https://github.com/SciTools/cartopy-tutorial/blob/master/tutorial/projections_crs_and_terms.ipynb) for a great overview of this topic. Instead, we will dive into the more practical sides of Caropy usage.

Introducing Cartopy

https://scitools.org.uk/cartopy/docs/latest/ (https://scitools.org.uk/cartopy/docs/latest/)

Cartopy makes use of the powerful PROJ.4 (https://proj4.org/), numpy and shapely libraries and includes a programatic interface built on top of Matplotlib for the creation of publication quality maps.

Key features of cartopy are its object oriented projection definitions, and its ability to transform points, lines, vectors, polygons and images between those projections.

Cartopy Projections and other reference systems

In Cartopy, each projection is a class. Most classes of projection can be configured in projection-specific ways, although Cartopy takes an opinionated stance on sensible defaults.

Let's create a Plate Carree projection instance.

To do so, we need cartopy's crs module. This is typically imported as `ccrs` (Cartopy Coordinate Reference Systems).

In [1]:

```
import cartopy.crs as ccrs
import cartopy
```

Cartopy's projection list tells us that the Plate Carree projection is available with the `ccrs.PlateCarree` class:

https://scitools.org.uk/cartopy/docs/latest/crs/projections.html (https://scitools.org.uk/cartopy/docs/latest/crs/projections.html)

Note: we need to *instantiate* the class in order to do anything projection-y with it!

In [2]:

```
ccrs.PlateCarree()
```

Out[2]:

Drawing a map

Cartopy optionally depends upon matplotlib, and each projection knows how to create a matplotlib Axes (or AxesSubplot) that can represent itself.

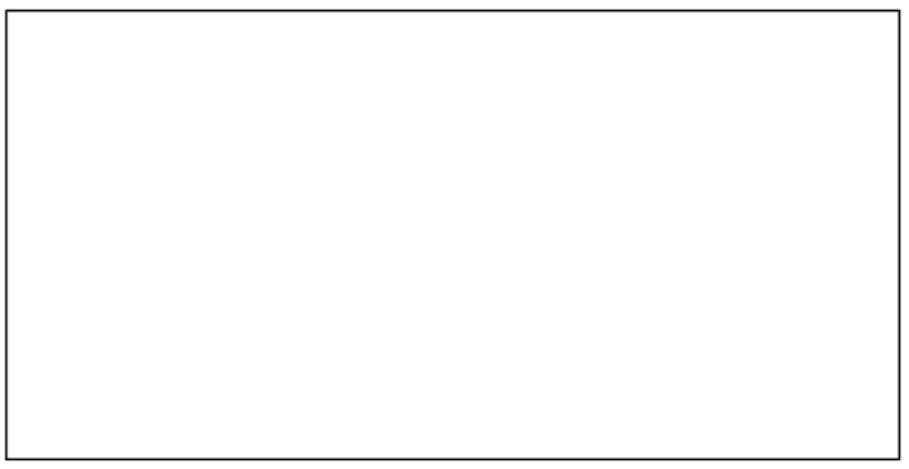
The Axes that the projection creates is a cartopy.mpl.geoaxes.GeoAxes (https://scitools.org.uk/cartopy/docs/latest/matplotlib/geoaxes.html#cartopy.mpl.geoaxes.GeoAxes). This Axes subclass overrides some of matplotlib's existing methods, and adds a number of extremely useful ones for drawing maps.

We'll go back and look at those methods shortly, but first, let's actually see the cartopy+matplotlib dance in action:

In [3]:

```
%matplotlib inline
import matplotlib.pyplot as plt

plt.axes(projection=ccrs.PlateCarree())
```

Out[3]:



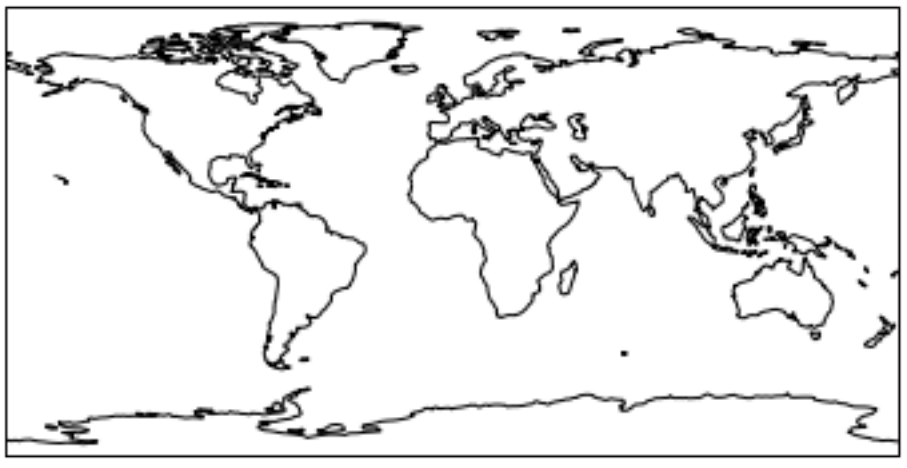
That was a little underwhelming, but we can see that the Axes created is indeed one of those GeoAxes[Subplot] instances.

One of the most useful methods that this class adds on top of the standard matplotlib Axes class is the `coastlines` method. With no arguments, it will add the Natural Earth `1:110,000,000` scale coastline data to the map.

In [4]:

```
plt.figure()
ax = plt.axes(projection=ccrs.PlateCarree())
ax.coastlines()
```

Out[4]:

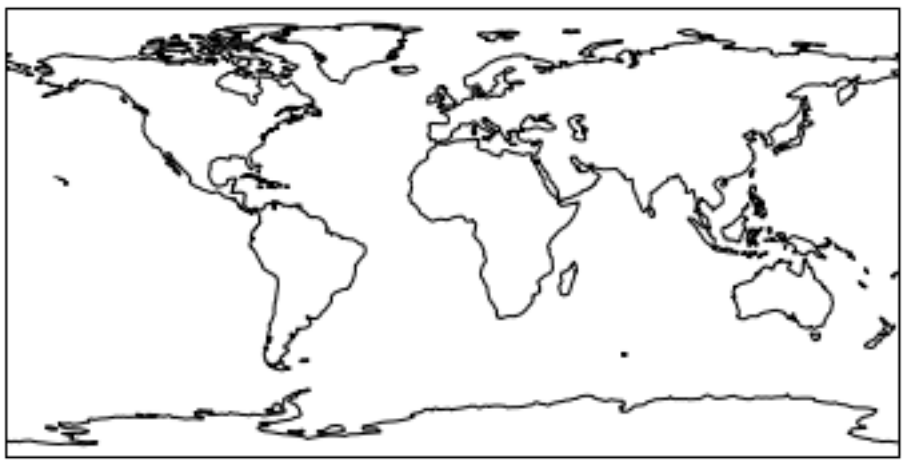


We could just as equally created a matplotlib subplot with one of the many approaches that exist. For example, the `plt.subplots` function could be used:

In [5]:

```
fig, ax = plt.subplots(subplot_kw={'projection': ccrs.PlateCarree()})
ax.coastlines()
```

Out[5]:



Projection classes have options we can use to customize the map

In [6]:

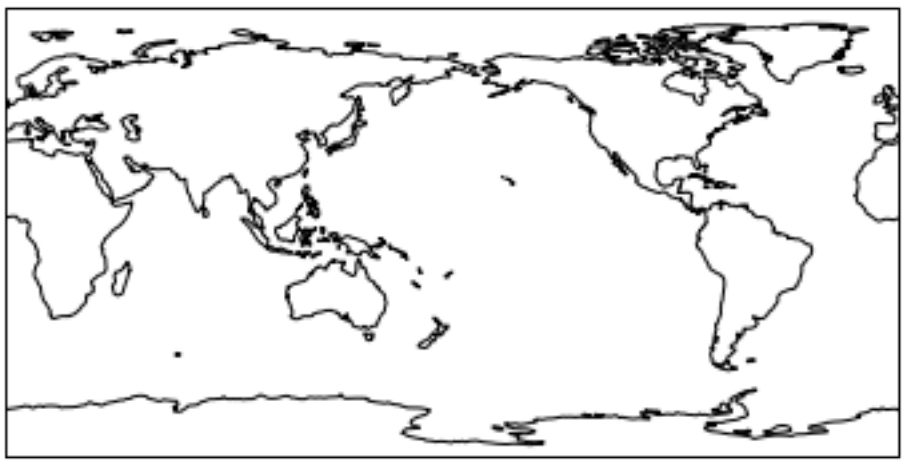
```
ccrs.PlateCarree?
```

```
Init signature: ccrs.PlateCarree(central_longitude=0.0, globe=None)
Docstring:
The abstract class which denotes cylindrical projections where we
want to allow x values to wrap around.
Init docstring:
Parameters
----------
proj4_params: iterable of key-value pairs
    The proj4 parameters required to define the
    desired CRS.  The parameters should not describe
    the desired elliptic model, instead create an
    appropriate Globe instance. The ``proj4_params``
    parameters will override any parameters that the
    Globe defines.
globe: :class:`~cartopy.crs.Globe` instance, optional
    If omitted, the default Globe instance will be created.
    See :class:`~cartopy.crs.Globe` for details.
File:           /opt/conda/lib/python3.6/site-packages/cartopy/crs.py
Type:           ABCMeta
```

In [7]:

```
ax = plt.axes(projection=ccrs.PlateCarree(central_longitude=180))
ax.coastlines()
```

Out[7]:



Useful methods of a GeoAxes

The cartopy.mpl.geoaxes.GeoAxes (https://scitools.org.uk/cartopy/docs/latest/matplotlib/geoaxes.html#cartopy.mpl.geoaxes.GeoAxes) class adds a number of useful methods.

Let's take a look at:

- set_global (https://scitools.org.uk/cartopy/docs/latest/matplotlib/geoaxes.html#cartopy.mpl.geoaxes.GeoAxes.set_global) - zoom the map out as much as possible
- set_extent (https://scitools.org.uk/cartopy/docs/latest/matplotlib/geoaxes.html#cartopy.mpl.geoaxes.GeoAxes.set_extent) - zoom the map to the given bounding box

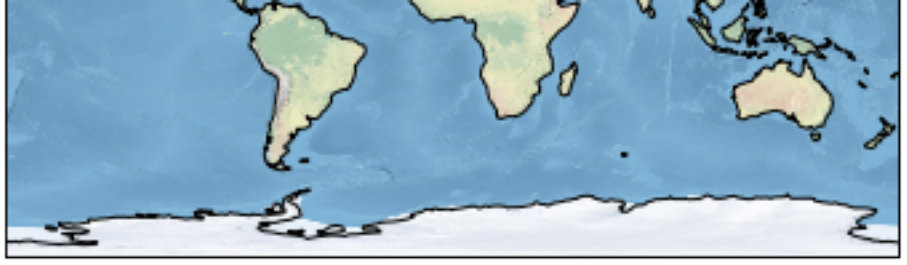
- gridlines (https://scitools.org.uk/cartopy/docs/latest/matplotlib/geoaxes.html#cartopy.mpl.geoaxes.GeoAxes.graticule) - add a graticule (and optionally labels) to the axes
- coastlines (https://scitools.org.uk/cartopy/docs/latest/matplotlib/geoaxes.html#cartopy.mpl.geoaxes.GeoAxes.coastlines) - add Natural Earth coastlines to the axes
- stock_img (https://scitools.org.uk/cartopy/docs/latest/matplotlib/geoaxes.html#cartopy.mpl.geoaxes.GeoAxes.stock_img) - add a low-resolution Natural Earth background image to the axes

- imshow (https://matplotlib.org/api/_as_gen/matplotlib.axes.Axes.imshow.html#matplotlib.axes.Axes.imshow) - add an image (numpy array) to the axes
- add_geometries (https://scitools.org.uk/cartopy/docs/latest/matplotlib/geoaxes.html#cartopy.mpl.geoaxes.GeoAxes.add_geometries) - add a collection of geometries (Shapely) to the axes

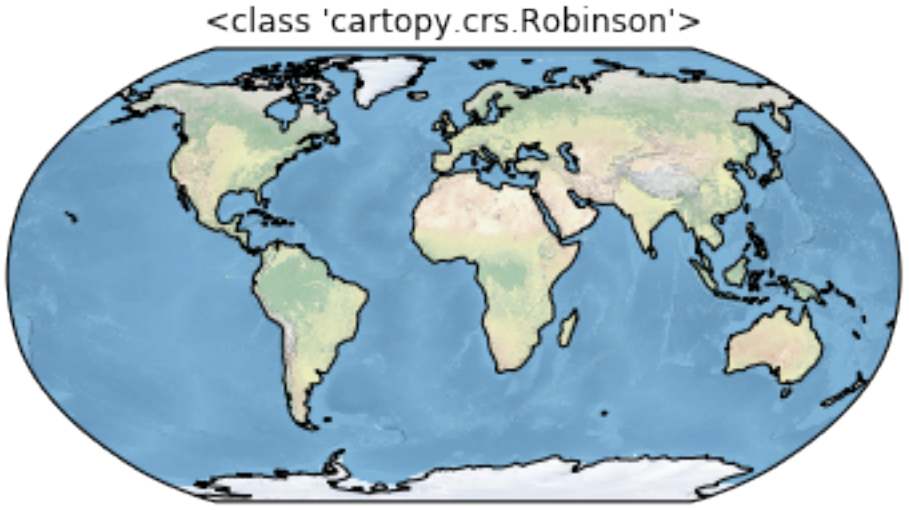
Some More Examples of Different Global Projections

In [8]:

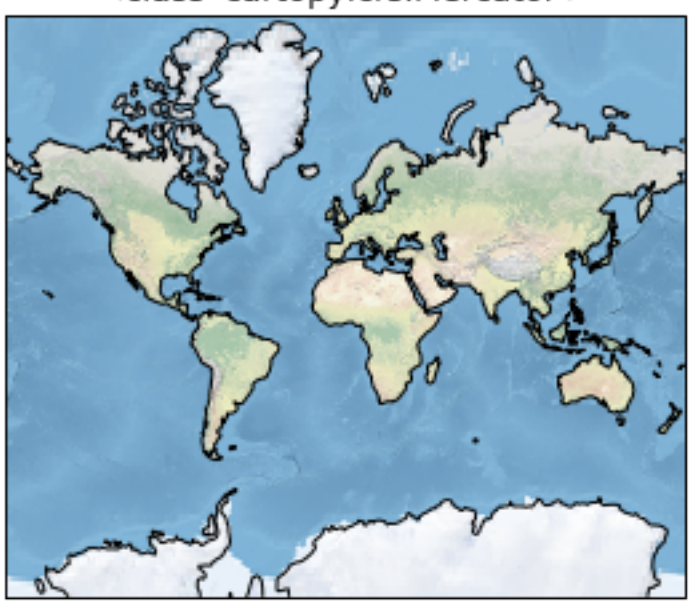
```
projections = [ccrs.PlateCarree(),
               ccrs.Robinson(),
               ccrs.Mercator(),
               ccrs.Orthographic(),
               ccrs.InterruptedGoodeHomolosine()
              ]


for proj in projections:
    plt.figure()
    ax = plt.axes(projection=proj)
    ax.stock_img()
    ax.coastlines()
    ax.set_title(f'{type(proj)}')
```



<class 'cartopy.crs.Robinson'>

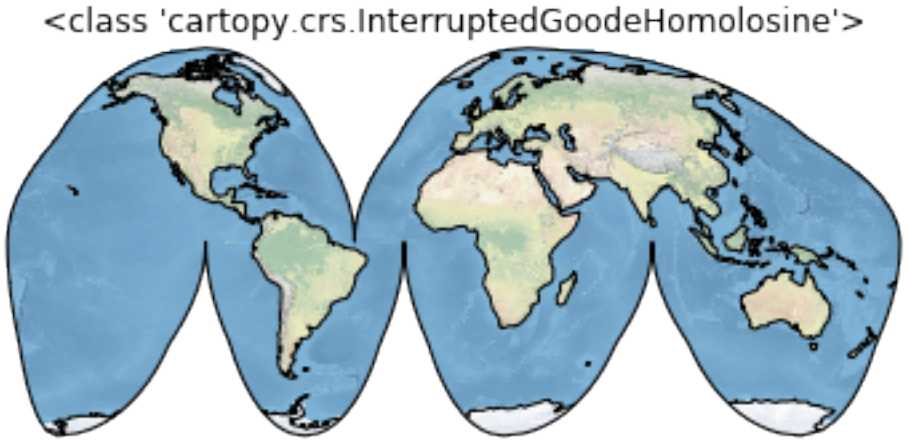


<class 'cartopy.crs.Mercator'>



<class 'cartopy.crs.Orthographic'>

<class 'cartopy.crs.InterruptedGoodeHomolosine'>



Regional Maps

To create a regional map, we use the `set_extent` method of GeoAxis to limit the size of the region.

In [9]:

```
ax.set_extent?
```

```
Signature: ax.set_extent(extents, crs=None)
Docstring:
Set the extent (x0, x1, y0, y1) of the map in the given
coordinate system.

If no crs is given, the extents' coordinate system will be assumed
to be the Geodetic version of this axes' projection.

Parameters
----------
extent
    Tuple of floats representing the required extent (x0, x1, y0, y1).
File:      /opt/conda/lib/python3.6/site-packages/cartopy/mpl/geoaxes.py
Type:      method
```

In [10]:

```
central_lon, central_lat = -10, 45
extent = [-40, 20, 30, 60]
ax = plt.axes(projection=ccrs.Orthographic(central_lon, central_lat))
ax.set_extent(extent)
ax.gridlines()
ax.coastlines(resolution='50m')
```

Out[10]:

Adding Features to the Map

To give our map more styles and details, we add `cartopy.feature` objects. Many useful features are built in. These "default features" are at coarse (110m) resolution.

`cartopy.feature.BORDERS`	Country boundaries
`cartopy.feature.COASTLINE`	Coastline, including major islands
`cartopy.feature.LAKES`	Natural and artificial lakes
`cartopy.feature.LAND`	Land polygons, including major islands
`cartopy.feature.OCEAN`	Ocean polygons
`cartopy.feature.RIVERS`	Single-line drainages, including lake centerlines
`cartopy.feature.STATES`	(limited to the United States at this scale)

Below we illustrate these features in a customized map of North America.

In [11]:

```
import cartopy.feature as cfeature
import numpy as np

central_lat = 37.5
central_lon = -96
extent = [-120, -70, 24, 50.5]
central_lon = np.mean(extent[:2])
central_lat = np.mean(extent[2:])

plt.figure(figsize=(12, 6))
ax = plt.axes(projection=ccrs.AlbersEqualArea(central_lon, central_lat))
ax.set_extent(extent)

ax.add_feature(cartopy.feature.OCEAN)
ax.add_feature(cartopy.feature.LAND, edgecolor='black')
ax.add_feature(cartopy.feature.LAKES, edgecolor='black')
ax.add_feature(cartopy.feature.RIVERS)
ax.gridlines()
```

Out[11]:

If we want higher-resolution features, Cartopy can automatically download and create them from the Natural Earth Data (http://www.naturalearthdata.com/) database or the GSHHS dataset (https://www.ngdc.noaa.gov/mgg/shorelines/gshhs.html) database.

In [12]:

```
rivers_50m = cfeature.NaturalEarthFeature('physical', 'rivers_lake_centerlines', '50m')

plt.figure(figsize=(12, 6))
ax = plt.axes(projection=ccrs.AlbersEqualArea(central_lon, central_lat))
ax.set_extent(extent)

ax.add_feature(cartopy.feature.OCEAN)
ax.add_feature(cartopy.feature.LAND, edgecolor='black')
ax.add_feature(rivers_50m, facecolor='None', edgecolor='b')
ax.gridlines()
```

Out[12]:

Adding Data to the Map

Now that we know how to create a map, let's add our data to it! That's the whole point.

Because our map is a matplotlib axis, we can use all the familiar maptplotlib commands to make plots. By default, the map extent will be adjusted to match the data. We can override this with the `.set_global` or `.set_extent` commands.

In [13]:

```
# create some test data
new_york = dict(lon=-74.0060, lat=40.7128)
honolulu = dict(lon=-157.8583, lat=21.3069)
lons = [new_york['lon'], honolulu['lon']]
lats = [new_york['lat'], honolulu['lat']]
```

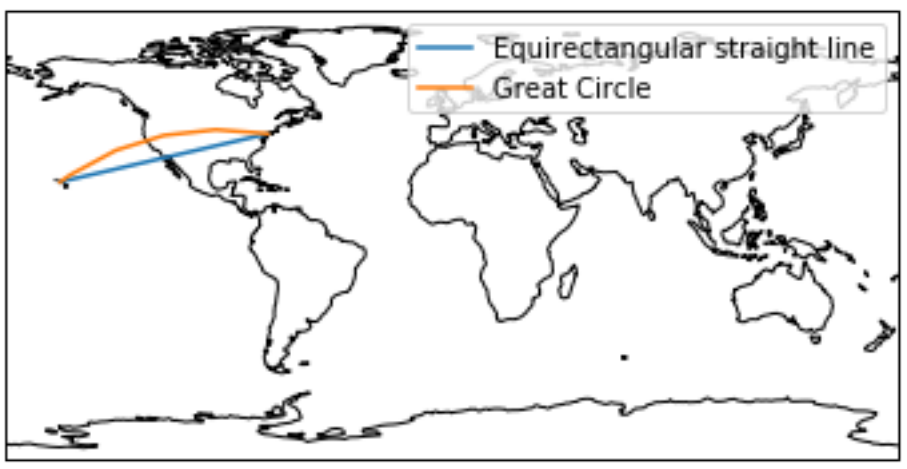
Key point: **the data also have to be transformed to the projection space**. This is done via the `transform=` keyword in the plotting method. The argument is another `cartopy.crs` object. If you don't specify a transform, Cartopy assume that the data is using the same projection as the underlying GeoAxis.

From the Cartopy Documentation (https://scitools.org.uk/cartopy/docs/latest/tutorials/understanding_transform.html)

> The core concept is that the projection of your axes is independent of the coordinate system your data is defined in. The `projection` argument is used when creating plots and determines the projection of the resulting plot (i.e. what the plot looks like). The `transform` argument to plotting functions tells Cartopy what coordinate system your data are defined in.

In [14]:

```
ax = plt.axes(projection=ccrs.PlateCarree())
ax.plot(lons, lats, label='Equirectangular straight line')
ax.plot(lons, lats, label='Great Circle', transform=ccrs.Geodetic())
ax.coastlines()
ax.legend()
ax.set_global()
```



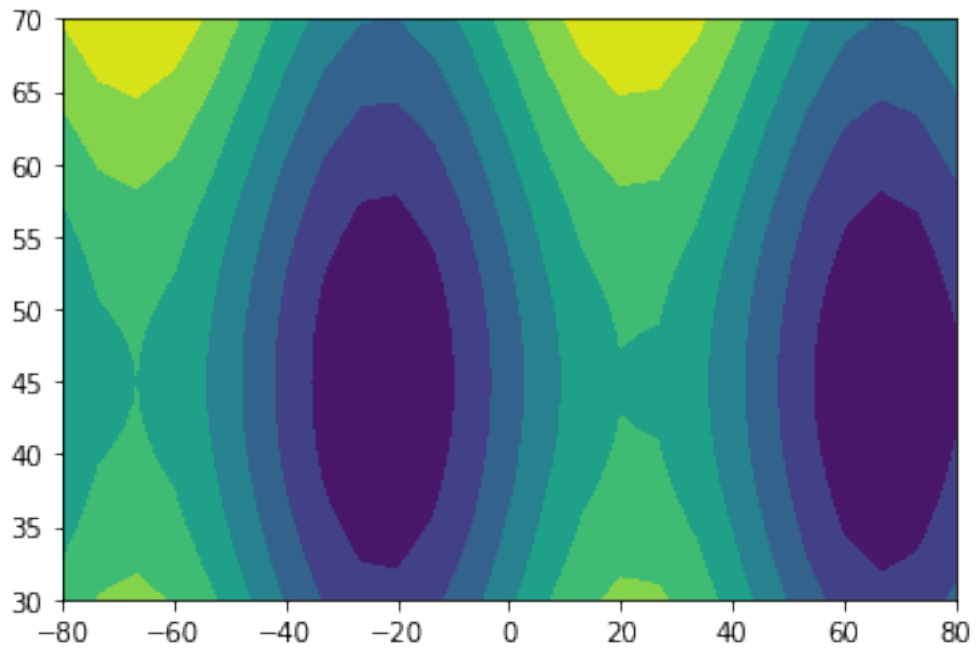
Plotting 2D (Raster) Data

The same principles apply to 2D data. Below we create some example data defined in regular lat / lon coordinates.

In [15]:

```
import numpy as np
lon = np.linspace(-80, 80, 25)
lat = np.linspace(30, 70, 25)
lon2d, lat2d = np.meshgrid(lon, lat)
data = np.cos(np.deg2rad(lat2d) * 4) + np.sin(np.deg2rad(lon2d) * 4)
plt.contourf(lon2d, lat2d, data)
```

Out[15]:

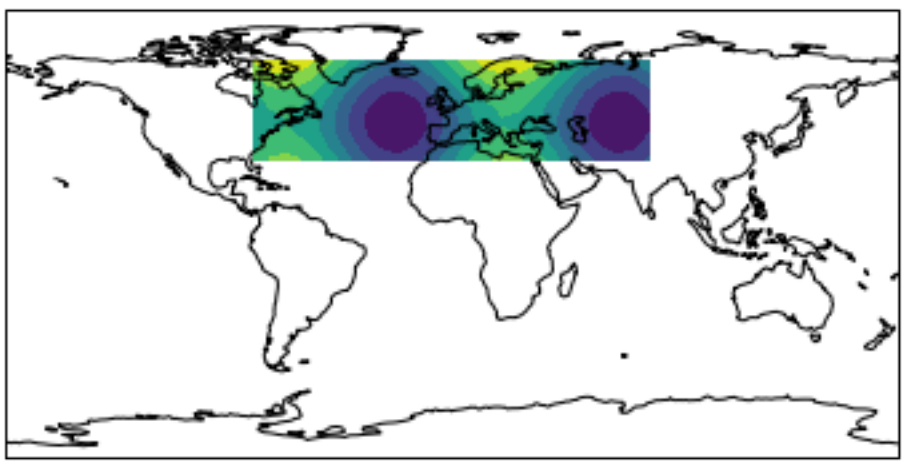


Now we create a `PlateCarree` projection and plot the data on it without any `transform` keyword. This happens to work because `PlateCarree` is the simplest projection of lat / lon data.

In [16]:

```
ax = plt.axes(projection=ccrs.PlateCarree())
ax.set_global()
ax.coastlines()
ax.contourf(lon, lat, data)
```

Out[16]:

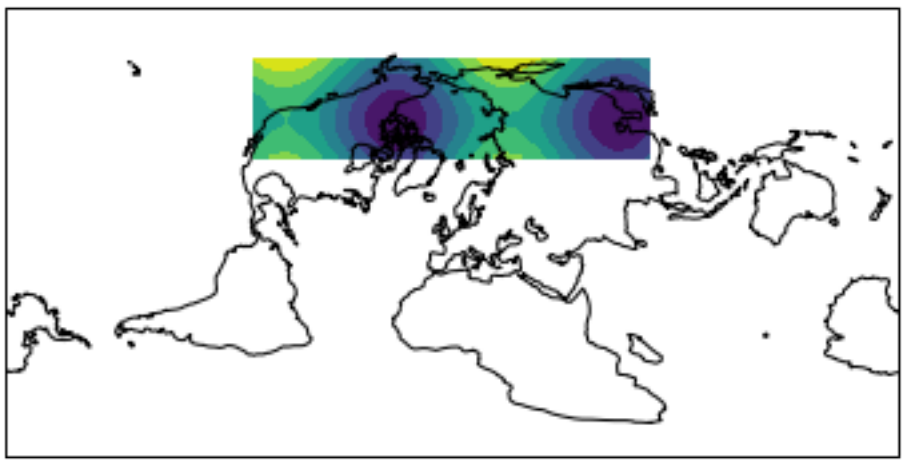


However, if we try the same thing with a different projection, we get the wrong result.

In [17]:

```
projection = ccrs.RotatedPole(pole_longitude=-177.5, pole_latitude=37.5)
ax = plt.axes(projection=projection)
ax.set_global()
ax.coastlines()
ax.contourf(lon, lat, data)
```

Out[17]:

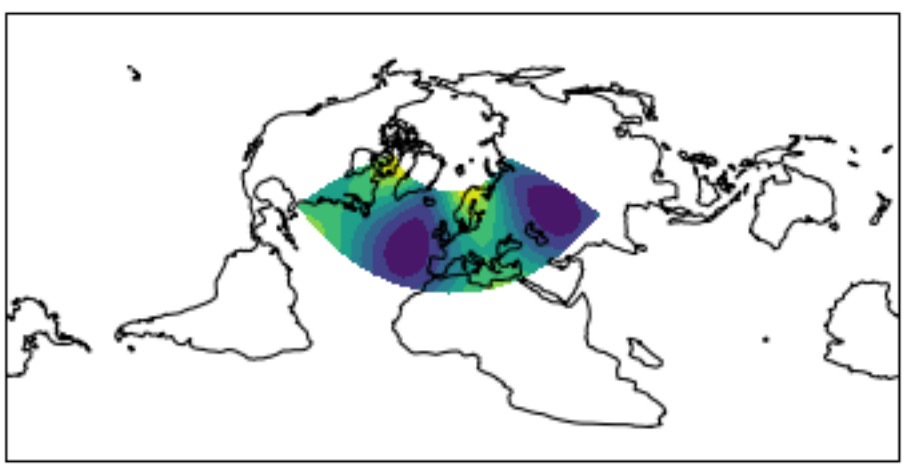


To fix this, we need to pass the correct transform argument to `contourf`:

In [18]:

```
projection = ccrs.RotatedPole(pole_longitude=-177.5, pole_latitude=37.5)
ax = plt.axes(projection=projection)
ax.set_global()
ax.coastlines()
ax.contourf(lon, lat, data, transform=ccrs.PlateCarree())
```

Out[18]:



Showing Images

We can plot a satellite image easily on a map if we know its extent

In [19]:

```
! wget https://lance-modis.eosdis.nasa.gov/imagery/gallery/2012270-0926/Miriam.A2012270
.2050.2km.jpg
```

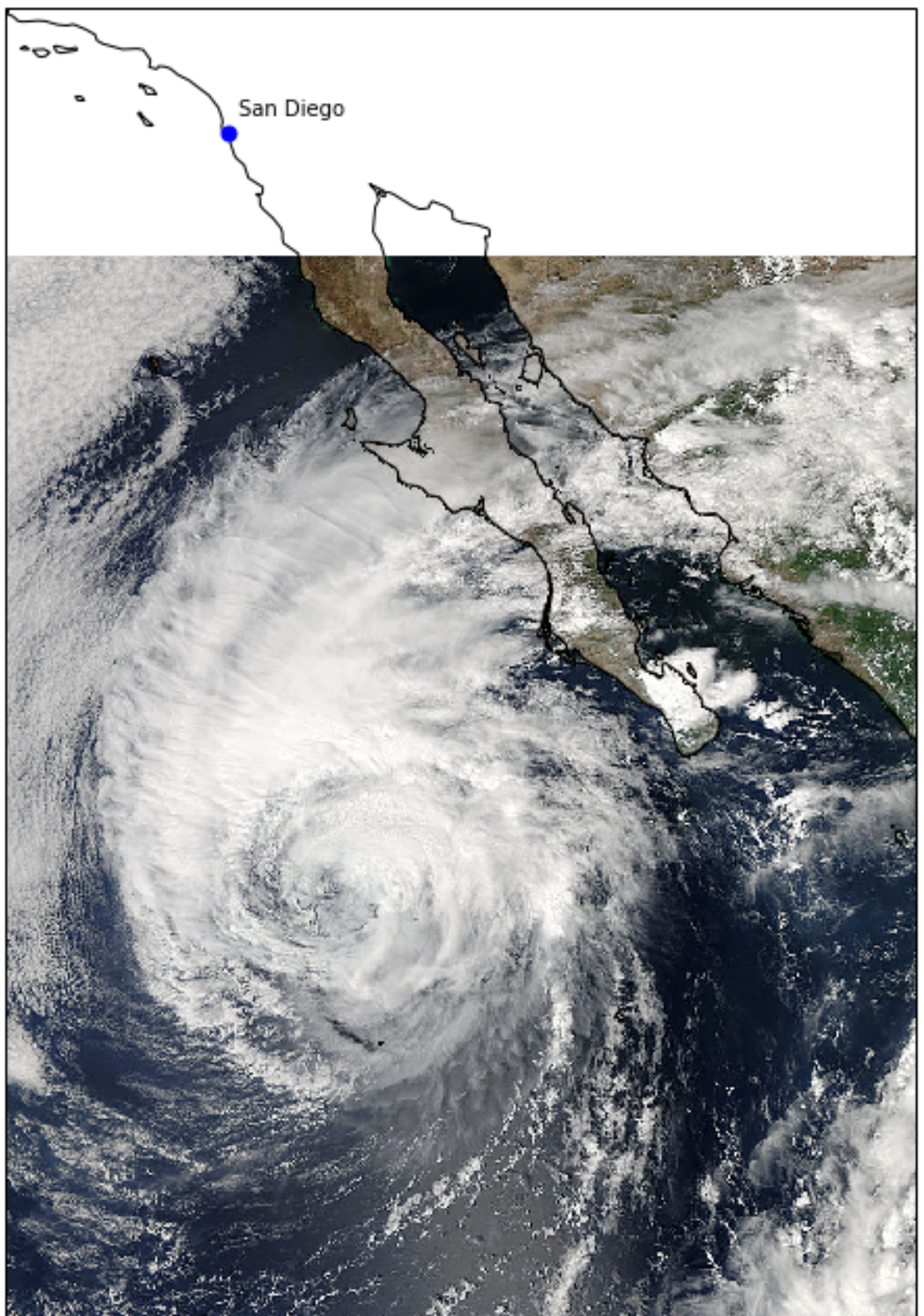
In [20]:

```
fig = plt.figure(figsize=(8, 12))
```

```
# this is from the cartopy docs
fname = 'Miriam.A2012270.2050.2km.jpg'
img_extent = (-120.67660000000001, -106.32104523100001, 13.2301484511245, 30.7668999999
99502)
img = plt.imread(fname)

ax = plt.axes(projection=ccrs.PlateCarree())

# set a margin around the data
ax.set_xmargin(0.05)
ax.set_ymargin(0.10)

# add the image. Because this image was a tif, the "origin" of the image is in the
# upper left corner
ax.imshow(img, origin='upper', extent=img_extent, transform=ccrs.PlateCarree())
ax.coastlines(resolution='50m', color='black', linewidth=1)

# mark a known place to help us geo-locate ourselves
ax.plot(-117.1625, 32.715, 'bo', markersize=7, transform=ccrs.Geodetic())
ax.text(-117, 33, 'San Diego', transform=ccrs.Geodetic())
```

Out[20]:

```
Text(-117, 33, 'San Diego')
```



Xarray Integration

Cartopy transforms can be passed to xarray! This creates a very quick path for creating professional looking maps from netCDF data.

In [21]:

```
import xarray as xr
ds = xr.open_dataset('NOAA_NCDC_ERSST_v3b_SST.nc')
ds
```

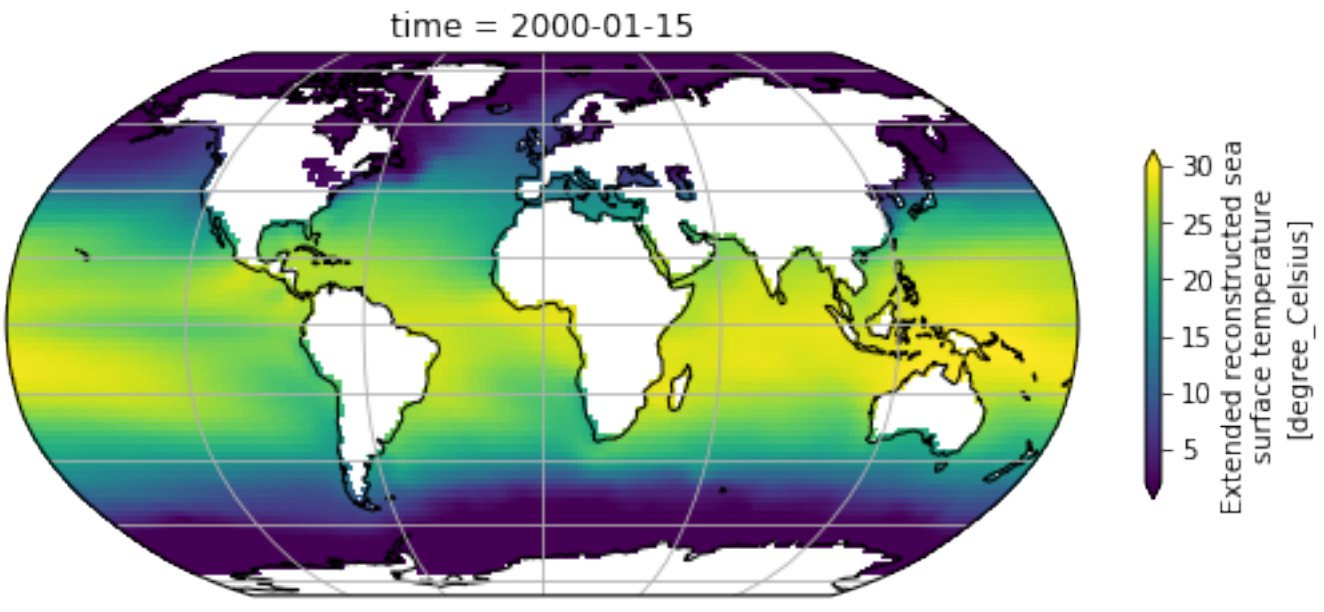
Out[21]:

```
Dimensions:  (lat: 89, lon: 180, time: 684)
Coordinates:
  * lat      (lat) float32 -88.0 -86.0 -84.0 -82.0 -80.0 -78.0 -76.0 -74.0 .
  * lon      (lon) float32 0.0 2.0 4.0 6.0 8.0 10.0 12.0 14.0 16.0 18.0 20.0
  * time     (time) datetime64[ns] 1960-01-15 1960-02-15 1960-03-15 ...
Data variables:
    sst      (time, lat, lon) float32 ...
Attributes:
    Conventions:  IRIDL
    source:       https://iridl.ldeo.columbia.edu/SOURCES/.NOAA/.NCDC/.ERSST
    history:      extracted and cleaned by Ryan Abernathey for Research Comp
```

In [22]:

```
sst = ds.sst.sel(time='2000-01-01', method='nearest')
fig = plt.figure(figsize=(9,6))
ax = plt.axes(projection=ccrs.Robinson())
ax.coastlines()
ax.gridlines()
sst.plot(ax=ax, transform=ccrs.PlateCarree(),
         vmin=2, vmax=30, cbar_kwargs={'shrink': 0.4})
```

Out[22]:



Doing More

Browse the Cartopy Gallery (https://scitools.org.uk/cartopy/docs/latest/gallery/index.html) to learn about all the different types of data and plotting methods available!

In [ ]:

Recent Posts

Tags

(./)

classes (./tag/classes.html)

functions (./tag/functions.html)

cartopy (./tag/cartopy.html)

git (./tag/git.html)

bash (./tag/bash.html)

environments (./tag/environments.html)

shell (./tag/shell.html)

binder (./tag/binder.html)

assignment (./tag/assignment.html)

file system (./tag/file-system.html)

Jupyter (./tag/jupyter.html)

github (./tag/github.html)

conda (./tag/conda.html)

version control (./tag/version-control.html)

modules (./tag/modules.html)

↑ Back to top