Computer Lab 6: Modeled Precipitation Data

Climate Data Analysis, ATS 301, Fall 2018

As before, we start by specifing that we want plots to be displayed inside the Jupyter Notebook, and load the modules we'll need.

In [ ]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

In [ ]:
```
# xarray is a module that will allow us to store and use the data.
import xarray as xr
```

Read in and examine the contents of the data file

Read in the data as an `xarray DataSet`, called `ds` (short for "data set"). You could call this anything you wanted to (within reason) though.

In [ ]:
```
ds = xr.open_dataset('/data/ATS_301/Data/pr_Amon_GISS-E2-H_rcp85_r1i1p1_200601-205012.nc')
```

Take a look at what's in this dataset.

In [ ]:
```
ds
```

Some of the **attributes** provide useful information about this model run.

To look at a **variable** or **coordinate** within this dataset, we add a dot ('.') and the name of the variable after the dataset name.

There are latitude, longitude, and time **dimensions**. Let's look at the values in them to get an idea of the model's **resolution**. How far apart are the latitudes and longitudes? What time-averaging has already been done on the data?

In [ ]:
```
ds.lat
```

In [ ]:
```
ds.lon
```

In [ ]:
```
ds.time
```

The variable we're interested in is precipitation:

In [ ]:
```
ds.pr
```

Depending on the varible, using `ds.` followed by the variable name will output all, some, or none of the data (array). To force it to output (at least some) of the values, add `.values` to the end. This will show at least the first 3 and last 3 of each array index.

```
In [ ]: ds.pr.values
```

Note that this variable it too large to write it all out.

Plotting 2-D data using Cartopy

We'd like to plot some of the data to visualize it. The `.plot()` function is a special function from `xarray`. It uses the **metadata** within the dataset to label axes, add a colorbar and title, and make a guess at what colors to use.

In general, we've been using **functions** that come before the variables we're passing to them. (For example, `plt.plot(var_name)`.) However, sometimes the function needs to follow the variable. (This is technically called a **method**, but I'll use the word **function** for simplicity.) The xarray functions are ones that need to follow the variable. The syntax is a little different:

`dataset.variable.plot()`

So try:

```
In [ ]: ds.pr.plot()
```

Hm. We'll that's interesting, but what I really wanted was to look at a **map (lat-lon) plot**. Since this data is 3-D, the xarray.plot function automatically created a **histogram** plot.

```
In [ ]: help(xr.plot)
```

```
In [ ]: help(xr.plot.plot)
```

To do a **lat-lon plot**, we need to use 2-D data (which will result in a `pcolormesh` plot). So let's just pick the first time (month) to examine. We use array subscripting to do this:

```
In [ ]: ds.pr[0,:,:].plot()
```

What does the "0" indicate? What do the two ":"'s indicate?

We can do a little better using the `cartopy` module.

```
In [ ]: import cartopy.crs as ccrs
```

```
In [ ]: ax = plt.axes(projection=ccrs.PlateCarree())
        ds.pr[0,:,:].plot(ax=ax, transform=ccrs.PlateCarree())
        ax.coastlines() # Cartopy can add coastlines to a plot.
```

In [ ]:
```
# Use a different projection.

ax = plt.axes(projection=ccrs.Mollweide())
ds.pr[0,:,:].plot(ax=ax, transform=ccrs.PlateCarree())
ax.coastlines()
```

In [ ]:
```
# Make the figure bigger
plt.figure(figsize=[12,8])

ax = plt.axes(projection=ccrs.Mollweide())
ds.pr[0,:,:].plot(ax=ax, transform=ccrs.PlateCarree())
ax.coastlines()
```

The `cmap='Blues'` option specifies the blue-white colormap. `cbar_kwargs={'shrink':0.4}` shrinks the colorbar by 0.4.

In [ ]:
```
# Make the figure bigger
plt.figure(figsize=[12,8])

ax = plt.axes(projection=ccrs.Mollweide())
ds.pr[0,:,:].plot(ax=ax, transform=ccrs.PlateCarree(),\
                  cmap='Blues',cbar_kwargs={'shrink':0.4})
ax.coastlines()

# Make the title more useful
plt.title('Jan 2006 Precipitation (units here!)')
```

See this page for a list of different colormaps you can use:

http://matplotlib.org/examples/color/colormaps_reference.html (http://matplotlib.org/examples/color/colormaps_reference.html)

For a list of the different projections:

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

Means in Xarray

We've been working with precip data from one particular time (month). Often, we want to work with the average (more precisely, the mean) over 1, 2, or even 3 dimensions.

If we are averaging over 2 (or more dimensions), does it matter which dimension we average first?

Let's start by averaging over the time dimension. This will give us the **time-mean**.

What happens when we just use the `mean` function w/o specifying any options?

In [ ]:
```
np.mean(ds.pr)
```

Use the `help` function to figure out what's going on and how to get the mean we want.

In [ ]:
```
help(np.mean)
```

```
In [ ]: time_average_pr=np.mean(ds.pr,axis=0)
```

```
In [ ]: time_average_pr
```

How many dimensions does the new array have?

Xarray allows us to do this in an easier way. Rather than having to remember which **axis** corresponds to which dimension, we can use the **names** of the **dimensions** themselves.

To average over a specific dimension, use the `mean(dim='   ')` xarray function. (Note that this function is added after the xarray variable name: `ds.ps` .)

```
In [ ]: new_time_average_pr=ds.pr.mean(dim='time')
```

Now make a lat-lon (map) plot for the time-average precipitation.

```
In [ ]: # Your code here.↔
```

Zonal and Global Means

By averaging over longitude (rather than time), we can calculate the time-series of the zonal-mean precip.

```
In [ ]: zonal_ave=ds.pr.mean(dim='lon')
```

```
In [ ]: zonal_ave
```

We can take a quick at data from one time to see if it looks as expected, by plotting the zonal average for the first time step.

```
In [ ]: zonal_ave[0,:].plot()

        # Alternate way of plotting this:

        #plt.plot(ds.lat,zonal_ave[0,:])
```

For the global mean, we have to be more careful. The area of a latitude band changes with latitude. So we have to take this in to account when calculating the global mean.

We start by creating an array of **weights**. Areas for the different latitudes are approximately proportional to the cos of latitude. The `np.cos` function takes input in radians, so we have to convert the latitude in degrees to the latitude in radians first, using the `np.radians` function.

```
In [ ]: lat_weight=np.cos(np.radians(ds.lat))
```

```
In [ ]: lat_weight
```

Now we use these weights to calculate the global averages.

The `sum` xarray function sums all of the values along the given dimension (similar to how the `mean` function works.

```
In [ ]: global_ave=((lat_weight*zonal_ave).sum(dim='lat'))/lat_weight.sum()
```

```
In [ ]: global_ave
```

The `np.average` function (below) can calculate a weighted average, but it's good to know how to do it yourself if you have to (above). Also, there is not a similar function for `xarray` , so the result is just a regular numpy array, and you've lost the time metadata.

```
In [ ]: np.average(zonal_ave,axis=1,weights=lat_weight)
```

Make a quick plot of the global averages.

```
In [ ]: global_ave.plot()
```

The above plot show the global average for each month. The seasonal cycle is dominating the variability, so let's look at some averages in time to better examine the data.

Working with time in xarray

One handy thing we can do with the time variable in xarray is look at values for given years or months.

Take a quick look at the time dimension again:

```
In [ ]: global_ave.time
```

Using xarray, you can output a list of the months for each time (not that the list of months should be very surprising to you, but this functionality will be more useful later).

You can also look at the year, day of the year (the middle of the month for this dataset, since they are monthly averages), or season for each time.

Note the use of "[]" rather than "()" below. You can think of this as taking a **subset** of the `global_ave` xarray **metadata** that is just the month values of the `time` coordinate.

```
In [ ]: global_ave['time.month']
```

```
In [ ]: global_ave['time.year']
```

```
In [ ]: global_ave['time.dayofyear']
```

```
In [ ]: global_ave['time.season']
```

This functionality becomes really useful when we want to average over some set of points in the time series. For example, to average all the data from each year together (i.e. construct the **annual average**, we use `groupby('time.year')` . This "groups" all of the data according to the specified variable (in this case, the years), and then calculates the mean for each group separately.

```
In [ ]: # This just averages all of the values together.
        global_ave.mean()
```

```
In [ ]: # This averages the values together for each year separately.
        annual_global_ave=global_ave.groupby('time.year').mean()

        annual_global_ave
```

Without xarray, you would actually have to create a loop that cycles through each year, and then calculate the mean for each set of twelve months. It would look something like this:

```
In [ ]: annual_glob_ave_2=np.zeros(45)
        for i in range(45):
            annual_glob_ave_2[i]=np.mean(global_ave[i*12:i*12+12])

        annual_glob_ave_2
```

This is more complicated, and we've lost the time coordinate data associated with the variable. I almost had us start with this technique first, so you could really understand how much easier it is to use xarray. But just take my word on this one.

Let's plot the **time series** (for the first annual, global mean we calculated) and see how it looks now that we've averaged together the months in each year (thus removing the seasonal cycle).

```
In [ ]: annual_global_ave.plot()
```

Alternately, we can go back to the global average time series data and calculate the **mean monthly** precipitation for each month (i.e., the **seasonal cycle**).

Note that we have to use the `global_ave` variable, not the `annual_global_ave` , which has already lost the information from the different months of the year.

```
In [ ]: month_averages = global_ave.groupby('time.month').mean()
```

```
In [ ]: month_averages
```

```
In [ ]: month_averages.plot()
```

The above plot shows the **seasonal cycle**.

Using data from two files

We'd like to use the model simulation for the whole 21st Century (minus the first 5 years, which are not included in this simulation). Read in the data and concatenate the two datasets together. Because they have all the same variables and dimensions, this is easy to do. xarray will use the `time` coordinate to determine which dataset comes first. If you have overlapping values of the `time` coordinate, you'd get an error.

In [ ]:
```
# You already read in this first dataset above, so you don't need to do it again.
#ds = xr.open_dataset('/data/ATS_301/Data/pr_Amon_GISS-E2-H_rcp85_r1i1p1_200601-20
5012.nc')

ds2 = xr.open_dataset('/data/ATS_301/Data/pr_Amon_GISS-E2-H_rcp85_r1i1p1_205101-21
0012.nc')

big_ds=xr.concat([ds,ds2],dim='time')
```

In [ ]:
```
big_ds
```

NOTE: `time_average_pr` is used earlier in the lab. The variable below uses the values from the new, combined data set. Either is fine to use here. You'll want to use this combined data set for your homework

In [ ]:
```
# Average over all times, for the combined data set.
time_ave_pr=??????
```

In [ ]:
```
#↔
```

In [ ]:
```
time_ave_pr
```

Plot the time-mean precip for the combined data set (lat-lon plot).

In [ ]:
```
#↔
```

To make the numbers easier to understand, we convert from kg m^{-2} s^{-1} to mm/day.

The density of water is approx 1000 kg m^{-3}. So one kg m^{-2} is 1/1000 m high, or 1 mm. The number of seconds in a day is 60X60X24.

We then multiply this by the 2-D array before plotting. (Note the `()` around this multiplication before plotting.)

In [ ]:
```
num_sec_in_day=60*60*24
```

In [ ]:
```
plt.figure(figsize=[12,8])
ax = plt.axes(projection=ccrs.Mollweide())
(num_sec_in_day*time_ave_pr).plot(ax=ax, \
                                   transform=ccrs.PlateCarree(),\
                                   cmap='Blues',\
                                   cbar_kwargs={'shrink':0.4})
ax.coastlines()
plt.title('Time-mean Precipitation (mm/day)')
```

Subplots

We'd like to compare this figure to other plots. However, if we plot everything individually, `plot` will pick a different range of values for the colors, which might not be the same for every plot. To make everything consistent, we specify the `vmin` and `vmax` in the xarray `plot` function. We'll use a minimum of 0 and a maximum of 35 (mm/day), since we expect some values larger than the time-mean in the previous plot.

```
In [ ]: help(xr.plot.pcolormesh)
```

```
In [ ]: plt.figure(figsize=[12,8])
        ax = plt.axes(projection=ccrs.Mollweide())
        (num_sec_in_day*time_ave_pr).plot(ax=ax, vmin=0, vmax=35,\
                                          transform=ccrs.PlateCarree(),\
                                          cmap='Blues',cbar_kwargs={'shrink':0.4})
        ax.coastlines()
        plt.title('Time-mean Precipitation (mm/day)')
```

Place plots next to each other for easier comparison using `subplot`. The first number in the `plt.subplot` function indicates the number of rows, the second the number of columns. The third number indicates which location this particular subplot corresponds to. Note that the first plot corresponds to ax1, the left side, and the second to ax2, on the right.

```
In [ ]: plt.figure(figsize=[14,6])

        # Make the subplots a little closer horizontally.
        plt.subplots_adjust(wspace=0)

        # Create the first plot

        ax1 = plt.subplot(1,2,1, projection=ccrs.Mollweide())
        (num_sec_in_day*time_ave_pr).plot(ax=ax1, vmin=0,vmax=35, \
                                            transform=ccrs.PlateCarree(), \
                                            cmap='Blues',\
                                            cbar_kwargs={'shrink':0.4})
        ax1.coastlines()
        plt.title('Time-mean Precipitation (mm/day)')

        # Create the second plot

        ax2 = plt.subplot(1,2,2, projection=ccrs.Mollweide())
        (num_sec_in_day*ds.pr[0,:,:]).plot(ax=ax2, vmin=0,vmax=35, \
                                           transform=ccrs.PlateCarree(),cmap='Blues',\
                                           cbar_kwargs={'shrink':0.4})
        ax2.coastlines()
        plt.title('Jan 2006 Precipitation (mm/day)')

        plt.show()
```

Move the colorbars to the bottom so we have more room horizontally.

In [ ]:
```
plt.figure(figsize=[14,5])

# Make the subplots a little closer horizontally.
plt.subplots_adjust(wspace=0)

# Create the first plot

ax1 = plt.subplot(1,2,1, projection=ccrs.Mollweide())
(num_sec_in_day*time_ave_pr).plot(ax=ax1, vmin=0,vmax=35, \
                                    transform=ccrs.PlateCarree(),\
                                    cmap='Blues',\
                                    cbar_kwargs={'orientation':'horizontal','shrin
k':0.8})
ax1.coastlines()
plt.title('Time-mean Precipitation (mm/day)')

# Create the second plot


ax2 = plt.subplot(1,2,2, projection=ccrs.Mollweide())
(num_sec_in_day*ds.pr[0,:,:]).plot(ax=ax2, vmin=0,vmax=35, \
                                   transform=ccrs.PlateCarree(),cmap='Blues',\
                                   cbar_kwargs={'orientation':'horizontal','shrink
':0.8})
ax2.coastlines()
plt.title('Jan 2006 Precipitation (mm/day)')


plt.show()
```

Extracting data for a particular month

xarray also allows us to easily extract the data that corresponds to a particular month, year, season, etc. Here, I'm extracting all the pr data that corresponds to a month of 1 (i.e., Jan), and then averaging over all of the Januarys.

This is simpler than what you would need to do if you didn't have xarray.

(Alternately, you can calculate the mean for each month using the `groupby` function, and then use subscripting to extract the result for the month you want.)

In [ ]:
```
big_ds.pr['time.month']
```

In [ ]:
```
big_ds.pr['time.month'] == 1
```

In [ ]:
```
big_ds.pr[big_ds.pr['time.month'] == 1]
```

In [ ]:
```
# Note the use of double equal sign here.

Jan_ave_pr=big_ds.pr[big_ds.pr['time.month'] == 1].mean(dim='time')


plt.figure(figsize=[14,5])

# Make the subplots a little closer horizontally.
plt.subplots_adjust(wspace=0)

ax1 = plt.subplot(1,2,1, projection=ccrs.Mollweide())
(num_sec_in_day*time_ave_pr).plot(ax=ax1, vmin=0, vmax=35, \
                                    transform=ccrs.PlateCarree(),cmap='Blues',\
                                    cbar_kwargs={'orientation':'horizontal','shrin
k':0.8})
ax1.coastlines()
plt.title('Time-mean Precipitation (mm/day)')


ax2 = plt.subplot(1,2,2, projection=ccrs.Mollweide())
(num_sec_in_day*Jan_ave_pr).plot(ax=ax2, vmin=0,vmax=35, \
                                 transform=ccrs.PlateCarree(),cmap='Blues',\
                                 cbar_kwargs={'orientation':'horizontal','shrink':
0.8})
ax2.coastlines()
plt.title('Mean Jan Precipitation (mm/day)')


plt.show()
```

Plot the difference between two lat-lon plots, and the difference as the fractional change. This is just an example. In your homework, you will need to compare the differences between averages over different sets of years.

In [ ]:
```
plt.figure(figsize=[10,8])
ax = plt.axes(projection=ccrs.Mollweide())
(num_sec_in_day*(Jan_ave_pr-time_ave_pr)).plot(ax=ax, \
                            transform=ccrs.PlateCarree(),\
                            cmap='bwr_r',cbar_kwargs={'shrink':0.4})
ax.coastlines()
plt.title('Jan-Annual Time-mean Precipitation (mm/day)')
plt.show()
```

In [ ]:
```
frac_diff=(Jan_ave_pr-time_ave_pr)/time_ave_pr

plt.figure(figsize=[10,8])
ax = plt.axes(projection=ccrs.Mollweide())
frac_diff.plot(ax=ax, transform=ccrs.PlateCarree(),cmap='bwr_r',\
               cbar_kwargs={'shrink':0.4})
ax.coastlines()
plt.title('Jan-Annual Time-mean Precipitation (fraction)')
```

In [ ]:
```
np.min(frac_diff)
```

In [ ]:
```
np.max(frac_diff)
```

In [ ]:
```

```