Assignment 1 - Numpy and Matplotlib

1 Plotting and analyzing ARGO float data

1.1 Import numpy

1.2 Use the shell command curl to download an example ARGO float profile from the North Atlantic.

The data file's url is <http://www.ldeo.columbia.edu/~rpa/argo_float_4901412.npz>

1.3 Load the data file

1.4 Extract the temperature, pressure and salinity arrays to arrays T, S, P and mask out invalid data (the nan values from missing points).

1.5 Extract the date, lat, lon, and level arrays.

1.5 Note the shapes of T, S and P compared to these arrays. How do they line up?

1.6 Load the necessary package for plotting using pyplot from matplotlib.

1.7 Make a 1 x 3 array of plots for each column of data in T, S and P.

The vertical scale should be the levels data. Flip the veritcal axis direction so that levels increase downward on the plot. Each plot should have a line for each column of data. It will look messy. Make sure you label the axes and put a title on each subplot.

1.8 Compute the mean and standard deviation of each of T, S and P at each depth in level.

1.9 Now make a similar plot, but show only the mean T, S and P at each depth. Show error bars on each plot using the standard deviations.

Again, make sure you label the axes and put a title on each subplot.

In [ ]:

1.10 Compute the mean and standard deviation of each of T, S and P for each time in date.

1.11 Plot the mean T, S and P for each entry in *time*, now on a *3 x 1* subplot grid with time on the horizontal axis. Show error bars on each plot using the standard deviations.

1.12 Create a scatter plot of the positions of the ARGO float data. Color the positions by the date. Add a grid overlay.

Don't forget to label the axes!

2 Matrix multiplication revisited

2.1 Create a function called myMatrixMultiply that takes input matrices X and Y and computes their matrix product.

Use the same three loop formulation from Assignment 5. If you want, you can replace the innermost loop with the sum operation or a matrix dot product since that may speed things up a bit.

2.2 Create ones() square matrices for A and B with n = 100. Use the %timeit function to compute the matrix product AB using your function myMatrixMultiply.

2.3 Now let's see how much faster Numpy's built in matrix multiplication routine is.

In Numpy, matrix multiplication is done using the dot() function. Use the %timeit function to compute the matrix product AB for n = 100 using dot() and time it using the %timeit function.

Now time how long it takes for n = 1000