CS 6014
Homework 6, Fortran and Python

Turn in to Collab all source files, paste printed output into the textbox, and upload the plots. Read the language-specific instructions for your language carefully.

Download the file wm1deg.grid from the Collab site for this assignment (Resources/Data/FP_HW). This is a topographic dataset for the entire Earth at a resolution of 1 deg by 1 deg. Each row is a line of latitude; the entries in the row are the elevations at each longitude along that value of latitude. Elevations are represented as deviations from sea level (plus or minus) in meters, to the nearest meter (i.e. no fractional parts). The starting latitude is 89.5 N (positive) and the starting longitude is 0.5 E. Remember that latitudes run from -90 to 90 while longitudes run either -180 to 180 or 0 to 360.

Your program should read the name of the topographic dataset from the command line. Do not hardcode it or ask for user input.

1. Read in the data and make a contour plot of it. The standard for maps is that north is at the top of the page and east is to the right. Does your picture conform to this? If it does not, correct the data ordering. You will need to make one-dimensional arrays for lat and long that are of the appropriate size and whose values you compute from the starting points and the resolution. You can use these as the x and y arrays for your contour plots.

Find the locations of the maximum and minimum elevations. Plot these points on your final contour plot. If you can, have the plotting software indicate the points; otherwise you may draw them by hand (neatly and carefully, please). Are they where you think they should be?

You should put your subprograms that are related to geographical manipulations or computations into a module called geo.[f90,py]. Functions related to reading data may go into your main program or you may create a module for that.

Language-specific instructions:

Fortran: Read the name of the topography file from the command line. Read in the following using a namelist: starting latitude, starting longitude, resolution in latitude, resolution in longitude. Make your arrays allocatable. Using the resolution values you just read, compute the sizes of the arrays you will need and allocate them to the appropriate size. Then read the dataset. You may use max/minloc and max/minval as appropriate. For writing the data for some plotting software, you may use an implied DO. You may also use an array slice. Remember that nearly all plotting software wants to read in data by rows. NB: Excel can make simple contour and surface plots, so if you don't know another package like Matlab or R you can still use Excel. You may use the files.f90 module to open the file, get the number of lines, and so forth. You may use a linspace function that I post.

Python: Read the name of the file from the command line. Read parameters starting latitude, starting longitude, resolution in latitude, and resolution in longitude from the command line. Use the optparse (built-in) module to accomplish this. Your final command line will look something like
python geostats.py --file wm1deg.grid –-lat 89.5 –-latres 1.0 --lon 0.5 --lonres 1.0
You can find examples of optparse online; a pretty good example is at
http://www.saltycrane.com/blog/2009/09/python-optparse-example/
Once you have read in your parameters, compute the dimensions of the arrays you will need. Read in the data using NumPy functions such as loadtxt or fromfile and reshape. Use linspace from numpy to get arrays of lat and lon.
In your main module include an appropriate main() function.

2. Using the code from Problem 1 as a base, write a function/subroutine to compute the fraction of the surface area of the Earth that is ocean. Write another function/subroutine to compute the total volume of ocean in cubic meters. How to do this: The element of area of the surface of a sphere is given by $dA=R^2\cos(\theta)d\theta d\phi$. (Cosine rather than sine because we measure latitudes from the equator and not the pole.) The radius of the Earth is 6378.1 km. Theta is latitude and phi is longitude. To compute the area, we sum up all the (approximate) area elements. The $d\theta$ is given by π/ilat and the $d\phi$ by 2π/ilon, where ilat and ilon are the number of latitudes and longitudes. Remember that latitude must be in radians when you compute the cosine. You may find that your fraction won't be all that accurate due to the coarse resolution but it shouldn't be wacky, so be sure to sanity-check your result. (It's surprisingly close considering the coarse resolution.) You can also check your code by ignoring topographic information and making sure you get 4π (for R=1). For volume, take into account the length of the column (positive, please). Print your results with some explanation of what each number is. Watch your units.

Fortran: you should write these as subroutines and pass arrays you need as arguments. They are allocatable in the calling routine so be sure to pass them in the correct manner so that you don't segfault. Usually we handle this by passing the size of the array as well.

3. The boundaries of North America are, very roughly, 12 degrees N to 66.5 degrees N and 130 degrees W to 60 degrees W. Don't forget to convert the longitude to whatever numerical system you are using. Pull out the data for this rectangle and make a contour map of it. Fortran: output the data in a form that your plotting software can read (e.g. csv or space-separated).

Hints:
Fortran: Fortran does not have a `find` so we have to fake it. You may need to do a loop to count the number of longitudes in your range, then use an allocatable array

to hold the index values. Please note that it's not uncommon when doing something like this to run a loop over the same data twice, once to count and once to assign values. This will be faster and easier than trying to come up with some kind of Python-like list type. You will need to figure out a way to write out rows of latitude lines that have only the required longitudes; there are several ways to do this.

Python: you can use the nonzero function of numpy. Usage: nonzero(<conditional>) returns the indices where <conditional> is True.