Computational Project 1

due: 09/07/2017, 2 pm

Last updated August 31, 2017. Remember to submit both electronic and hard copies of your code. Your final code should consist of three functions (one primary function, two subfunctions) all in a single .m file that can be directly executed without any user intervention to complete all steps of the project.

This exercise is designed to help you practice plotting, manipulating data, and using functions.

1. Load the database *CO_topo.txt* of vertical altitudes [m] for Colorado in a latitude × longitude grid into a matrix using the `load('CO_topo.txt')` command (make sure this file is in your current directory). Use the `size` command to figure out how the data is arranged in your matrix in terms of the number of rows and columns.

2. Convert all values in the topo map from units of m to units of ft.

3. Topo maps that you see often de-emphasize rugged vertical variations by plotting the square root of altitude. Make a plot of the square root of the topo data using the `surf` command. The plot may look black at first, so figure out how to set the `EdgeColor` property to `none` so that the grid-box edges are not displayed. Make sure you haven't plotted the state upside-down, backwards, etc. Label the axis, and add a figure title. Experiment with setting the `view` to a pair of angles such that the map is easily recognizable and the axis labels legible. Also, the plot will by default end up square. Adjust the x aspect ratio using the `pbaspect` command so that Colorado is accurately proportioned (it is about 280 miles tall and 380 miles wide).

4. We have provided a separate function, `get_coords.m`, that takes as input arguments a longitude and latitude and returns as output the corresponding row and column indices of the topography data matrix. Add to this function some code to make it print out an error message and return an index pair of -999,-999 if the requested latitude and longitude lie outside of the domain of the topo matrix.

5. Use the `get_coords.m` function and the `plot3` command to draw a filled black circle on your map at the correct altitude, latitude and longitude for Denver (39.7392° N, -104.9903° W). Hint: Use the `hold` command to allow for plotting on top of the current map.

6. Encapsulate all of these previous steps into a function that can be directly executed without any inputs from the command line. Place the `get_coords.m` function in the same file, so that it is a subfunction.

7. In the next step, you are going to make a new subfunction that evaluates and plots several aspects of potential bike rides originating from Denver. For simplicity, let's only consider bike rides that are straight lines heading diagonally from Denver. These lines will be specified in terms of the number of steps in the longitudinal direction (n_{lon}), the number of steps in the latitudinal direction (n_{lat}), and the step length. Consider each step to be of fixed length $\delta = 0.01/\sqrt{2}$ degrees for either direction. The function inputs should be n_{lat} and n_{lon}, as well as the latitude and longitude of Denver, and the topo matrix. The function should address the following requirements:

   (a) Draw a filled black circle on your topo map at the correct altitude, latitude, and longitude of the final destination. Hint: you can get MATLAB to add points to a previous figure, say Figure 1, by adding `figure(1)` directly before the plotting command.

   (b) Calculate the latitude x_i and longitude y_i at each point along the ride, each a distance of δ apart, including the endpoints. Hint: use the colon operator (`start:increment:stop`), but keep in mind that you need to consider the sign of the increment.

   (c) Calculate the total horizontal distance traveled [km] at each point i along the way from Denver to the destination as

   $$d_i = \beta(i-1)\delta\sqrt{2}$$

   including the endpoints at Denver and the destination, where β is a unit conversion factor from degree to km that we will assume for simplicity is equal to 100 km per degree. The will give you a vector, $\mathbf{d}$, to be used for plotting below. Display the calculated value of the final distance traveled in the command window.

   Self-check: what value should be displayed if $n_{lon} = n_{lat} = 100$?

   (d) Find a vector **z** of altitudes values at each point along the ride. Hint: first use `get_coords` and the latitude and longitude coordinate vectors defined in step (b) to determine the corresponding row and column of the topo matrix for each point, and then access this row and column of the topo matrix to get the correct altitude.

   (e) Calculate the average altitude along the ride:

   $$\bar{z} = \frac{1}{n}\sum_{i=1}^{n} z_i \tag{1}$$

   Display the calculated value in the command window.

(f) Calculate the total altitude gained (just the uphill portions) along the ride:

$$dz = \sum_{i=2}^{n} z_i - z_{i-1} \text{where } z_i > z_{i-1} \tag{2}$$

Display the calculated value in the command window.

(g) Calculate the running average altitude at each point i along the ride using the following iterative formula

$$\bar{a}_{i+1} = \bar{a}_i + \frac{1}{i+1}(z_{i+1} - \bar{a}_i) \tag{3}$$

(h) Create a new plotting window with the `figure()` command. Subdivide this plotting window into two panels using the `subplot` command. Plot the altitude vector $\mathbf{z}$ as a function of distance $\mathbf{d}$ using a red line. Set the axis bounds to tightly capture the plot range. Label the axis.

(i) In the second panel, plot the running average elevation $\bar{\mathbf{a}}$ as a function of distance $\mathbf{d}$ as a dotted blue line.

8. Add code to the end of your function defined in part 6 that calls your subfunction from part 7 for the following bike rides:

   (a) 100 steps south and 100 steps west

   (b) 50 steps north and 50 steps east

   (c) 150 steps north and 150 steps west