Homework #7: Functions and GPS data **Due: 5:00 PM 11/04/16**

Please read the following questions carefully and make sure to answer the problems completely. In your MATLAB script(s), please include the problem numbers with your answers. Then use the *Publish* function in MATLAB to publish your script to a *pdf* document. For more on the *Publish* functionality within MATLAB see http://www.mathworks.com/help/matlab/matlab_prog/publishing-matlab-code.html. Upload your *pdf* file to Blackboard under Assignment #7. Your filename should be *GEOS397_HW7_Lastname.pdf*. Hint: You can achieve this automatically by calling your MATLAB script *GEOS397_HW7_Lastname.m*.

Intended Learning Outcomes

Students will be able to:

1. Create functions that that input and give output
2. Create functions that do not take any inputs
3. Create functions that test input values for validity and error out when necessary
4. Estimate volumetric properties of Earths' layered structure
5. Print formatted output to the screen
6. Plot Pie charts in MATLAB
7. Import *.neu GPS data into MATLAB and extract particular data
8. Plot multiple component GPS data using subplots for each component
9. Process large amounts of GPS from individual stations
10. Explain the term 'code snippet'

Part 1: Volume of Earth's layers *(55 pts.)*

Step 1: A function for the volume of a sphere

Create a MATLAB m-file called sphereVolume.m. In this file, write a function that accepts one argument – the radius. This function should return the volume of the sphere with the user-specified radius. *(5 pts.)*

This should be a very simple function to write. For example, other than comments and the end statement, my function has only two lines of code.

Step 2: Ensuring valid function inputs

Make sure that your new function checks for input errors. List the input tests you use to check the validity of your input radius. Make sure to use the **error** function to tell the user when an input is invalid. Make sure your error message is useful. *(5 pts.)*

Step 3: Another function for the thickness of Earth's layers

In the same directory, create another m-file called earthLayers.m. After your header info, copy in the following lines of code:

```
% thicknesses of Earth's layers [km]
crust  = 35;
mantle = 2850;
core   = 3486;
```

earthLayers() should be a function that does NOT take an input. You simply call this function and it returns the thickness of each layer. *(5 pts.)*

Step 4: Compute the volume of each layer

Use your two new functions **spehreVolume()** and **earthLayers()** to compute the volume of each layer of the Earth, as well as the total volume of the Earth. Assume the radius of the Earth is 6371 km (i.e. $35 + 2850 + 3486$). *Note: You will need to create a new script (e.g. GEOS397_HW7_Lastname.m) where you will call these functions and compute the volumes.* Use the following variable names to store your results:

- volCore *(2 pts.)*
- volMantle *(2 pts.)*
- volCrust *(2 pts.)*
- volEarth *(2 pts.)*

This script should be generalized, so that if you wanted to calculate the volumes of crust, mantle, and core on some other planet, all you would have to change are the values assigned to crust, mantle, and core. *(10 pts.)*

Step 5: Compute each layer's percentage of Earth's total volume

Once you have calculated the layer volumes and Earths total volume (in km^3), you can calculate what percent of Earths total volume each layer occupies. Store these values in variables named

- pctVolCore *(2 pts.)*
- pctVolMantle *(2 pts.)*
- pctVolCrust *(2 pts.)*

Step 6: Formatted output of computational results

Your script should end by printing out the following information:

```
Volume of Core:   x.xxe+xx [units]
Volume of Mantle: x.xxe+xx [units]
```

```
Volume of Crust:  x.xxe+xx [units]
Core:   xx.x [%]
Mantle: xx.x [%]
Crust:  xx.x [%]
```

Replace the xx.xx with your calculated values. The volumes should be printed in units of km^3 in scientific notation with two decimal places. The volume percent should be labeled with a percent sign and should be printed in floating point notation with one decimal place. Make sure that your values all line up nicely when printed to the screen (like shown above). This will make the output much easier to read. *(2 pts. for each of the 6 lines = 12 pts.)*

Step 7: Plotting a Pie chart

Have your script make a pie chart of the total % volumes of each layer. *NOTE: use the function* ***pie()***. Use the colormap called summer, and explode out each layer, so they are easy to see. Otherwise, your crust slice will be very small and hard to see. Pie will automatically label the %s of each layer, but add a text label for each slice (i.e. Crust, Mantle, Core) using the MATLAB function **text()**. For details on these new commands (text, pie), read the documentation, look online and experiment with different options. **pie** and **text** are relatively straightforward, so you should not have too much trouble with them. Make sure to look in the *doc* for help plotting both percentages and text in the pie chart. *(4 pts.)*

Part 2: Automated GPS Data Processing *(45 pts.)*

Your task is to write a function called **getGPSstats.m** that takes a single GPS *.neu filename as the input argument and returns nothing. The function will calculate several useful parameters from the data in the *.neu file and make plots of the north, east, and vertical positions over time.

Step 1: Download the data and import into MATLAB

Download and extract the zip file provided for this HW set. The files contain time series of locations (north, east, elevation) from seven continuous GPS sites in southern California. See Figure 1 for locations.

The data in each .neu file is consistently organized into columns. The column contents are

```
Col 1: Time [decimal year]
Col 2: Time [integer year]
Col 3: Time [integer day number]
Col 4: North Position [m]
Col 5: East Position [m]
Col 6: Vertical Position [m]
Col 7: North Position Error [m]
Col 8: East Position Error [m]
Col 9: Vertical Position Error [m]
```

Each file has its own unique number of header lines that appear before the data columns begin. You can see this by opening an *.neu file in a text editor or right-click in MATLAB and use 'Open as Text'. In your

function **getGPSstats.m** you will need to use **textscan()** to load the data from these columns. You can tell **textscan** to skip the header lines with something like

```
fid = fopen( filename, 'r' );
C = textscan( fid, format,'HeaderLines',2);
fclose(fid);
```

The piece of code above will skip the first two lines of any input file and then read in whatever *format* is set to.

Use the code snippet below in the beginning of your own function to determine how many header lines are in an *.neu file. *NOTE: If you are not familiar with the term 'code snippet' then Google it.*

```
% first need to find the number of headerlines
nHeaderLine = 0;
fid = fopen( fullfile(dirIn, filename), 'r' );
line = fgetl( fid ); % get the first line
nHeaderLine = nHeaderLine + 1;
line = fgetl( fid ); % get the second line

while strcmp(line(1:2),'% ')
    nHeaderLine = nHeaderLine + 1;
    line = fgetl( fid ); % get the second line
end
nHeaderLine = nHeaderLine + 1;
fclose(fid)
```

After adding this code to your function, you should then be able to read in the column data using **textscan**, setting *nHeaderLine* as the value you give **textscan** to skip the header lines. Write a data load *section* using **textscan** that stores the column data as *cell data type. (5 pts.)*

Step 2: Strip out the relevant data

To make analysis easier, parse out each cell element into a separate vector and give a each vector a variable name that is descriptive (i.e. short and mnemonic). You do not need to do anything with the errors, so you can ignore these data. Recall that this GPS data gives one data point per day, so each data point represents one day. *(5 pts.)*

Step 3: Process the GPS data

In your function **getGPSstats.m**, calculate the following parameters from the data:

- Station Name: (can be grabbed from the filename string since the name is always the first 4 characters)
- Total Time [decimal years]
- Number of Days in the Time Series (i.e. number of data points)
- Total East Displacement [cm] (i.e. final position - initial position)
- Total North Displacement [cm]
- Total Vertical Displacement [cm]
- Average East Velocity [cm/yr] (i.e. Total East Displacement/Total Time)

- Average North Velocity [cm/yr]
- Average Vertical Velocity [cm/yr]

This information should then be printed out to the screen in an organized format with all values shown in floating point format to only two decimal places, except for station name and number of data points which should be printed as a string and integer, respectively. *(20 pts.)*

For example:

```
Site name: xxxx
Time span: xx.xx [yrs]
Number of days with data: xxxx
Total north displacement:    xx.xx [cm]
Total east displacement:     xx.xx [cm]
Total vertical displacement: xx.xx [cm]
Avg north velocity:    xx.xx [cm/yr]
Avg east velocity:     xx.xx [cm/yr]
Avg vertical velocity: xx.xx [cm/yr]
```

Step 4: Plot the GPS data

Your function should also make a figure with three subplots (arranged vertically) for each *.neu file. Plot decimal time [yrs] on the x-axis and north position [cm] on the y-axis (top). Plot decimal time [yrs] on the x-axis and east position [cm] on the y-axis (middle). Plot decimal time [yrs] on the x-axis and vertical position [cm] on the y-axis (bottom). Make sure to label each axis. Plot the north position time series with a blue line and circles with white fill and blue edges in size 3. Do the same for the other two plots, but make east red and vertical green. *(10 pts.)*

Step 5: Process all of the data

Write a script called **allGPS.m** that calls your **getGPSstats.m** function seven times (i.e. one for each GPS file). Running this script should produce 7 plots and print the data statistics to the MATLAB command window once for each data file. *(5 pts.)*

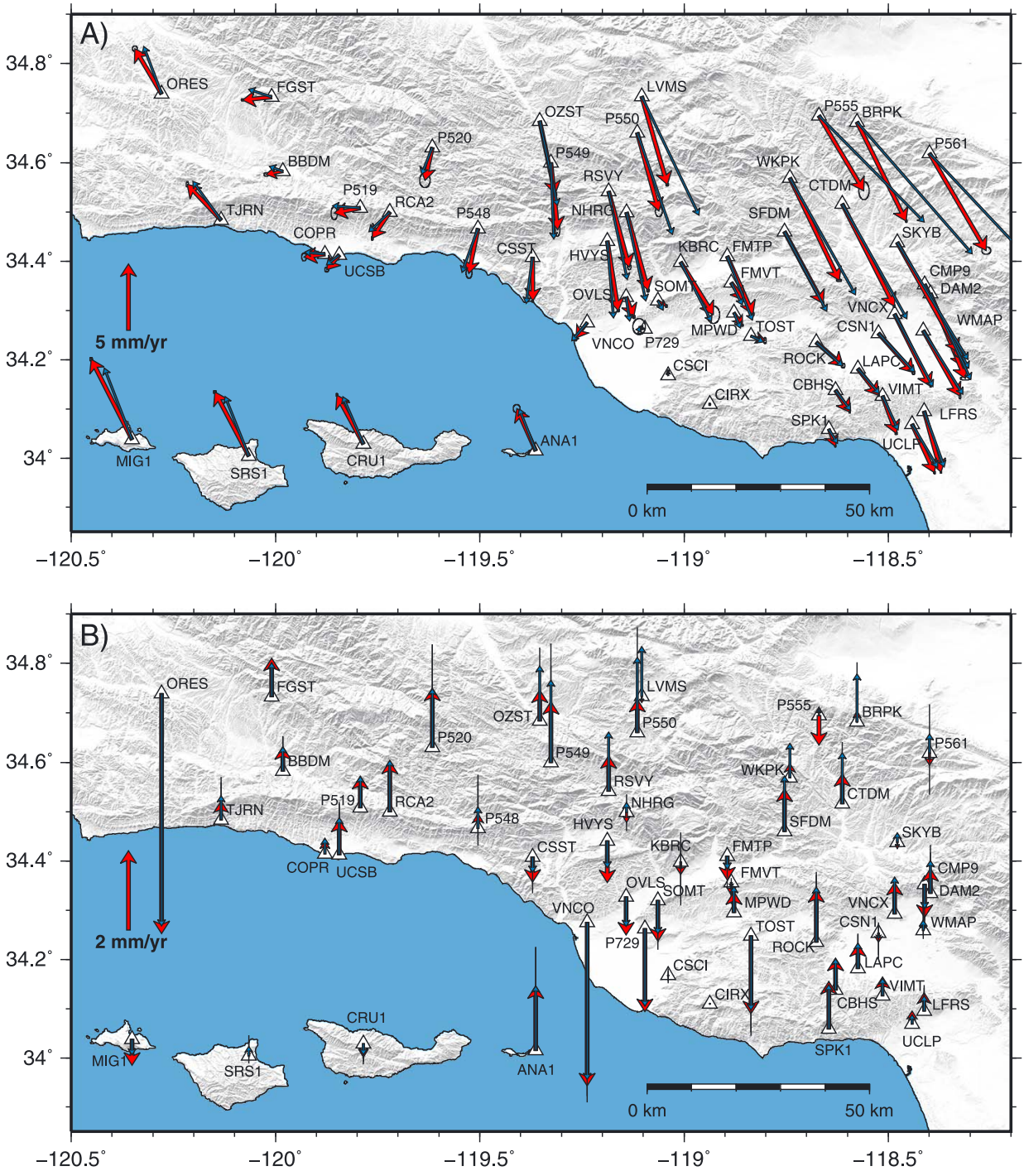

Figure 1: Map of horizontal (A) and vertical (B) GPS velocities in the Western Transverse Ranges, CA. From Marshall et al. [2013].