Data Analysis in Geophysics (CERI 7104/8104)
Homework 1 – Due 9/28/18

This homework involves programming in Python. There are two problems, each of which requires you to fill in some code. I have defined the functions to work with a test function, so that you and I can both check if your code is working correctly. I have provided some test cases for each problem; I will use these test cases, plus additional tests, when grading your homework. Points will also be awarded for code clarity, robustness, and documentation.

1. Write a Python function `catalog_hist(catalog, nbins, minmag)` to make a histogram of an earthquake catalog. To make a histogram, you will divide the catalog up into a specified number of time bins of a fixed size and count the number of earthquakes that occur in each bin. The catalog will be represented by a list, with each list entry being another list containing the event time (written in decimal years, so an event occurring at midnight on January 1, 2018, would be 2018.0) and the magnitude. The function should have three inputs: the list representation of the catalog (required), the number of bins `nbins` (integer, default value of 20), and the minimum magnitude `minmag` to use in the analysis (float, default is `None`, meaning that all events should be included).

   Your code needs to determine the start times of each bin, and the number of events with magnitudes above the threshhold in that time bin. The start of the first bin should be midnight on January 1 of the first year represented in the catalog and the end of the last bin should be midnight on January 1 following the latest year represented in the catalog. You should *not* assume the events in the catalog are sorted in time. The function should return two lists, one that is a list of floats of length `nbins` indicating the start time of each bin, and a list of integers of length `nbins` indicating the number of events in each bin.

   I have provided "hw1_catalog.py" containing skeleton code and some tests that will be run if you execute the file as a script. If you wish to do additional tests, I also have provided a function to read a catalog from a text file in one of two formats: (1) a file that is already in the desired format, and (2) a file that includes the full date and time in a `datetime` string, a format used in the ANSS catalog and available as a module in Python. You should make sure your program works on the catalogs that I have provided: "newmadridcatalog.txt" is a catalog for the New Madrid Seismic Zone, and "pagercatalog.txt" is a 100 year global earthquake catalog, in addition to the tests that are provided. Again, the provided tests will count towards your grade, and I will run additional tests not included here.

2. Write a Python script that calculates P-wave travel times for a series of source-receiver pairs. This code will be used in your final project to determine how to backproject seismograms recorded at a receiver to possible source locations. Your code will read the source and receiver latitude/longitude coordinates from two text files (these will be inputs to the script), calculate the distance between each pair of latitude/longitude values, and then get calculate the travel time in seconds from the external code that I have provided with the problem. The code will then print out the information for all possible source/receiver pairs.

   You will need to write several functions to accomplish this task:

   (a) The function `calc_distance` converts latitude/longitude to distance in degrees. The distance in degrees Θ can be found from latitude/longitude values (φ_1, λ_1) and (φ_2, λ_2) using the Haversine formula:

   $$\Theta = 2 \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right). \tag{1}$$

   The square root (`sqrt`), sine (`sin`), cosine (`cos`), and arcsine (`asin`) functions are available in the `math` module. Note that all trigonometric functions are calculated in radians; you can convert from degrees to radians using `math.radians(x)` and vice versa using `math.degrees(x)`.

   This function takes two inputs: source latitude/longitude (a list of length 2) and receiver latitude/longitude (also a list of length 2), and outputs the distance between those points in degrees as a float.

   (b) The function `read_text_file` is used to open a file and read a series of latitude/longitude points. Its input is a string representing the filename to be read, and its output is a list of lists, where each list entry is the latitude/longitude (a list of length 2) read from each line in the file. If your code encounters a line in the file that does not contain two numbers, you should skip that line entirely and not add anything to the output list.

   (c) The function `print_travel_times` takes a list of lists containing the travel time information for a pair of points, and prints it to the screen. Each list entry will be a list of five floats representing source latitude, source longitude, receiver latitude, receiver longitude, and travel time between those points in seconds. Your code should print out each source/receiver pair on a separate line, with a single space separating each of the five entries. For example, for the source/receiver pair $(35.5°, 270°)$ and $(55°, 10°)$, the code would output the following to the terminal on a single line:

   ```
   35.5 270.0 55.0 10.0 653.10247696669933
   ```

   Running your code on included files should produce the output above.

(d) The function `calc_travel_times` takes two lists of lists, each representing a series of latitude/longitude pairs, and returns a list of lists containing travel time information between all possible pairs of stations. Each entry in the output list will be a list of five floats representing source latitude, source longitude, receiver latitude, receiver longitude, and travel time between those points in seconds (i.e. the same format that is expected for the input to `calc_travel_times`). This function will need to loop over all possible pairs to calculate the travel time and combine the appropriate information into a list. You will find the provided function `get_p_travel_time` useful here, which takes a single float as input (the distance in degrees) and returns a single float representing the travel time in seconds.

(e) The function `main` takes the two files given to the code as inputs (strings), reads from the files to get the latitude/longitude values, calculates the travel times, and then prints out the results. `main` should not return a value. You will find the above functions useful for writing this function.

Note: Not all pairs of sources and receiver will have a direct P-wave arrival. If you try to pick a source-receiver pair that does not have a direct P-wave arrival, the code will give an error. The code I have provided will give this error message, and you do not have to worry about this possibility in your solution.

The code is provided in "calc_travel_times.py" with functions that you need to fill in. Additional code needed to perform the travel time calculations is included in the file "travel_time.py" and the associated libraries in the "lib" folder. As with the first problem, I have provided some code with which you can test your functions. However, I will use more test cases not included here when grading your work, so you should think about how I might test your code when writing it to ensure that it works correctly for any number of situations.

To run your code, you should call it from the command line with the source and receiver files as arguments. For example, using the provided files, I would run them from the shell as follows (with output included):

```
$ python calc_travel_times.py source_points.txt receiver_points.txt
35.5 270.0 55.0 10.0 653.10247696669933
```

If you omit the source/receiver files, the code will run the provided test functions. I have also provided some longer source/receiver files that are longer that you can use to test your code.

Remember that you will be graded on coding style in addition to code correctness. This means providing sufficient documentation by writing a docstring, commenting your code, and choosing clear variable names and code structure. You should also include assertions in your code to check input values (this is important – think hard about what might trip up your code and what you can check to avoid these problems!).