CS/NEUR125 Brains, Minds, and Machines

Lab 6: Inferring Location from Hippocampal Place Cells

Due: Wednesday, March 8

This lab explores how place cells in the hippocampus encode the location of an animal in its environment. You will work with real recordings of neural activity from the place cells of a rat moving along a linear track, provided by the laboratory of Prof. Matt Wilson at MIT. Following some MATLAB preliminaries, you will write a MATLAB script to visualize the data and then analyze the data to infer the animal's location from the neural activity.

To begin, create a copy of this Google document, modify the title of the copy to include your partner names, and share the copy between partners, as you did in previous labs. A description of the code submission and questions for you to answer are shown in blue.

Start MATLAB on your lab Mac and use **Fetch** to download the folder named **placeLab** from the CS file server to the Desktop on your Mac. You will find this folder inside the cs125/download folder in your individual account on the CS server. For this lab, set the **Current Folder** in MATLAB to the **placeLab** folder.

I. MATLAB Preliminaries

(1) More on plotting: figure position and size, subplot, and bar plots

When a new figure window is opened with the **figure** command, MATLAB uses a default window size. For the plots in this lab, it will help to set a desired size for the window. This can be done by setting the **Position** property when opening a new figure window, as shown below:

figure('Position', [left bottom width height])

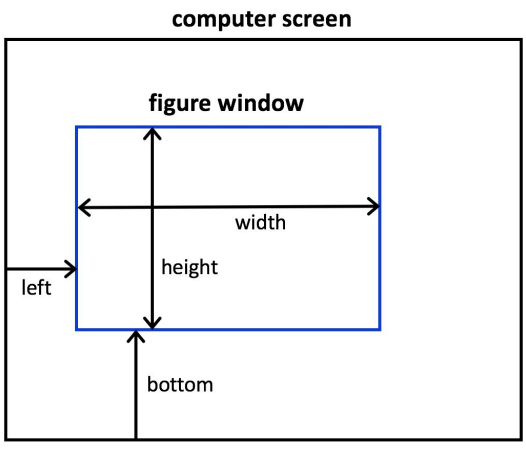
left: distance in pixels, from left edge of computer screen to left edge of figure window

bottom: distance from bottom edge of computer screen to bottom edge of figure window

width: width of figure window in pixels

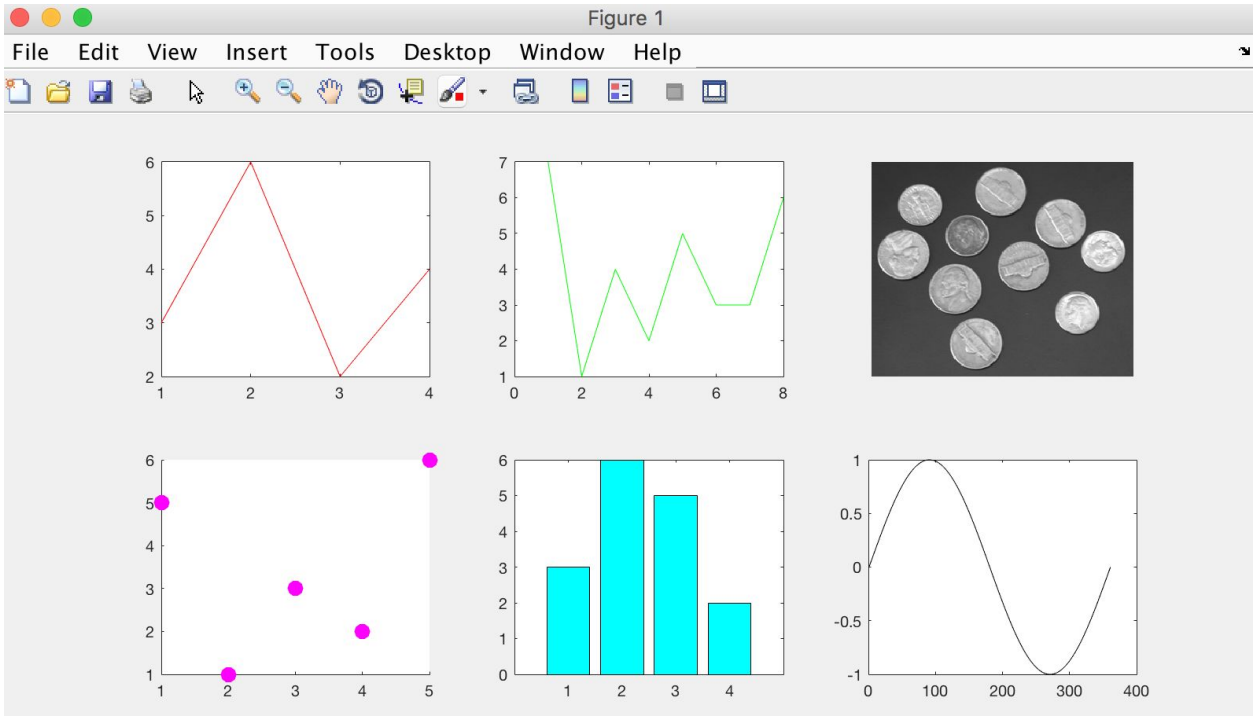
height: height of figure window in pixels

example: figure('Position', [50 100 800 500])



Multiple display areas within a figure window can be specified using the **subplot** command. The general form of this command, code sample, and resulting figure window are shown below. Note the creation of a **bar plot** in the second row, using the **bar** command:

subplot(rows, cols, place) **rows:** number of rows within the figure window **cols:** number of columns within the figure window **place:** indicates which subarea to use for the next display command - the number of the subplot area increases from left to right within each row, and continues to increase with each new row, as shown in the example on the right	figure('Position', [100 100 800 400]) subplot(2, 3, 1) plot(1:4, [3 6 2 4], 'r') subplot(2, 3, 2) plot(1:8, [7 1 4 2 5 3 3 6], 'g') subplot(2, 3, 3) coins = imread('coins.png'); imshow(coins) subplot(2, 3, 4) scatter(1:5, [5 1 3 2 6], 100, 'm', 'filled') subplot(2, 3, 5) bar(1:4, [3 6 5 2], 'c') subplot(2, 3, 6) plot(sind(0:360), 'k')



(2) Applying max and min to matrices, and using colon notation for indices

In the last lab, you learned that when applying the **mean** function to a matrix, MATLAB returns a row vector containing the **average value for each column** within the matrix. The mean function can also compute the **average value for each row** within the matrix. This function has a second input that is ***optional***, which specifies which direction to compute the average. Study the following example to understand how the mean function works.

```
>> mx = [2  5  1  4;  4  2  0  2;  3  5  2  6]      % create a sample matrix
mx =
    2    5    1    4
    4    2    0    2
    3    5    2    6
>> mean(mx)              % call the mean function with no second input - by default,
ans =                          % MATLAB computes the average for each column
    3    4    1    4
>> mean(mx, 1)           % specify a second input of 1 to compute average for each column
ans =
    3    4    1    4
>> mean(mx, 2)           % specify a second input of 2 to compute average for each row
ans =
    3
    2
    4
```

The **max** and **min** functions operate in a similar way, but when providing an optional input to specify whether to compute the minimum or maximum of the rows or columns of the matrix, an additional input of **[ ]** (empty square brackets) needs to be inserted as a second input. In the examples below, two output variables are also specified, to store both the maximum values and the indices of each row or column where these values were found.

```
>> mx                                  % show contents of sample matrix
mx =
    2    5    1    4
    4    2    0    2
    3    5    2    6
>> [maxVal, maxInd] = max(mx)       % call max function with no additional inputs - by default,
maxVal =                                % MATLAB computes maximum value for each column
    4    5    2    6
maxInd =                                % and row numbers where max values were found
    2    1    3    3
>> [maxVal, maxInd] = max(mx, [ ], 1)     % specify a third input of 1 to compute maximum
value
maxVal =                                          % for each column
    4    5    2    6
maxInd =
    2    1    3    3
>> [maxVal, maxInd] = max(mx, [ ], 2)     % specify a third input of 2 to compute maximum
value
maxVal =                                          % for each row
    5
    4
    6
maxInd =
    2
    1
```

```
    4
```

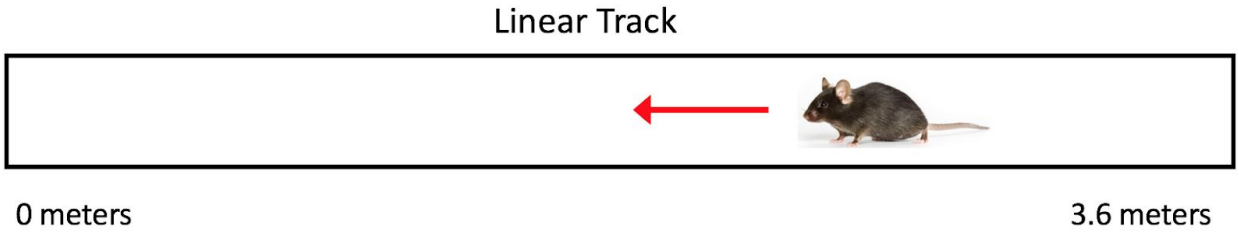
The use of the max function in both directions will come in handy in this lab!

Finally, recall that colon notation (**:**) can be used to access all or part of a row or column in a matrix, as shown in the following examples:

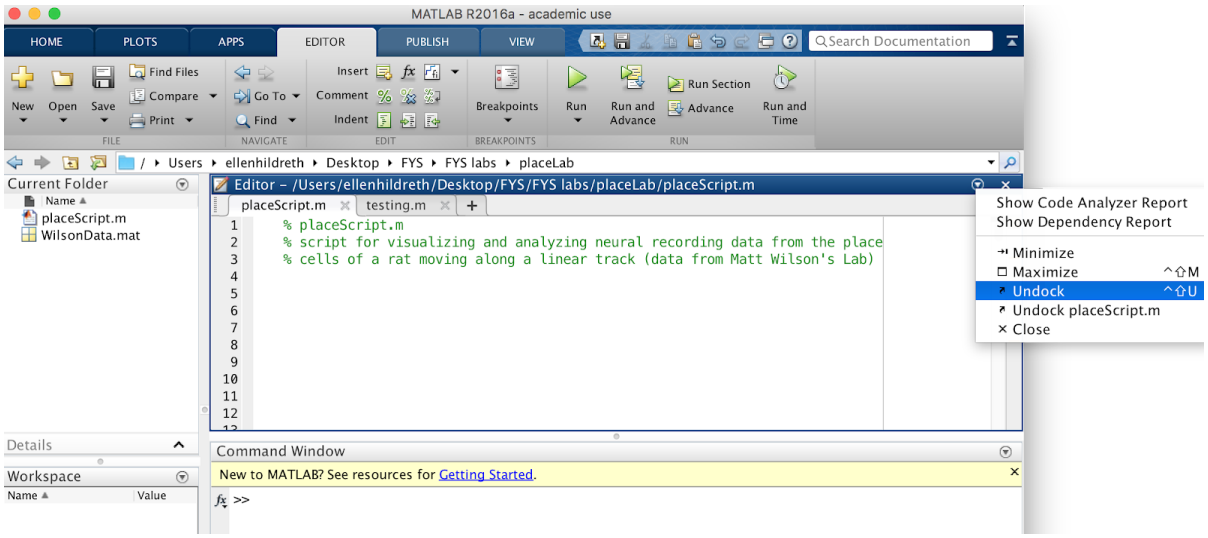
```
>> data = [2  5  1  4  3  0  4;  4  2  0  2  7  1  5;  3  5  2  6  1  0  4]      % create sample matrix
data =
    2    5    1    4    3    0    4
    4    2    0    2    7    1    5
    3    5    2    6    1    0    4
>> data(2, :)                                  % entire second row of the matrix
ans =
    4    2    0    2    7    1    5
>> data(:, 4)                                  % entire fourth column of the matrix
ans =
    4
    2
    6
>> data(2, 2:5)                                % columns 2 through 5 in the second row of the matrix
ans =
    2    0    2    7
```

II. Visualizing the Place Cell Data

Neural activity was recorded from place cells in the hippocampus of a rat while it was moving back and forth along a linear track that was 3.6 meters long, as illustrated below. The location of the animal on the track was recorded at a frequency of 30 Hz (i.e. one sample every 0.033 sec). We will first view the position of the animal over a long time period of about 14 minutes, spanning multiple traversals of the track, and then focus on the neural activity measured during one traversal of the track. Our goal will be to see if we can infer, or "decode," the rat's position at each moment from the spiking activity of the simultaneously recorded place cells.



All of the code for this part of the lab should be placed in a MATLAB script that you create from scratch, which you should name **placeScript.m**. Start the new script by clicking on the **New Script** icon in the upper left corner of the MATLAB window. We recommend that you **undock the Editor subwindow** from the rest of the MATLAB window, as shown in the picture at the top of the next page.



Add brief comments to the top of your script file as shown above, and then add the following commands to clear any variables in the workspace, close any open windows, and load the data:

```
clear all
close all
load WilsonData.mat
```

If you run your script and then enter **whos** in the Command Window, you'll see the following variables:

```
>> whos
  Name                 Size                Bytes  Class

  animalPos            1x25465            203720  double
  animalPosTrial1      1x571                4568  double
  animalPosTrial2      1x391                3128  double
  placeFields          30x71               17040  double
  posBins              1x71                  568  double
  spikesTrial1         30x571             137040  double
  spikesTrial2         30x391              93840  double
  timeStamp            1x25465            203720  double
  timesTrial1          1x571                4568  double
  timesTrial2          1x391                3128  double
```

General note: For every plot that you create for this lab, we expect you to add labels on the x and y axes using xlabel and ylabel. You do not need to add a title to each plot.

(1) View the animal's extended trajectory

The vector **animalPos** stores the position of the animal at each moment in time and **timeStamp** stores the actual time of each sample. Use the **figure** command to open a new figure window with a width of 800 and height of 300, and plot the animal's position as a function of time. As noted above, use **xlabel** and **ylabel** to add labels to your plot, including units.

> **Q1. Drag a copy of the figure of the animal's trajectory into this document.** Observing this plot, are there places along the track where the animal spends more time?

(2) View the animal's trajectory for a single trial

The vectors **animalPosTrial1** and **timesTrial1** contain the positions and times recorded for one traversal of the animal along the track, which we will refer to as **Trial 1**. Open a new figure window and plot the animal's position as a function of time for this one trial. *(You do not need to add this plot to this document.)*

> **Q2.** Observing the plot, about how long did it take the animal to traverse the track? Referring to the earlier picture of the mouse on the linear track, is the mouse moving from left to right, or from right to left, in this particular trial?

(3) View the place fields for the first 12 place cells

Place cells in the hippocampus are active when the animal is in a particular place in its environment. The **place field** of a place cell is the region of space over which the cell is active. The neural firing rate of the cell varies within its place field. This is similar to the *receptive field* of cells in the visual system. Given that the animal is moving along a linear track in this experiment, the place field is described in one dimension, corresponding to location along the track. **placeFields** is a 30 x 71 matrix that contains the place fields of 30 place cells. Each row of the matrix stores the place field for a different cell. The place field is defined as the average firing rate of the cell as a function of the animal's position on the track. The length of the track is divided into 71 discrete locations (every 0.025 meters over a range from 0.025 to 3.525 meters). These 71 locations are stored in the **posBins** vector. Each row of the **placeFields** matrix stores the average firing rate in the vicinity of each of these 71 locations, for a single cell.

To visualize some sample place fields, first open a new figure window of width 1000 and height 600. Construct a **for** loop to plot the place fields of the first 12 cells, which are stored in the first 12 rows of the **placeFields** matrix. Use **subplot** to create an arrangement of display areas within the figure window, with 3 rows and 4 columns. Use the **axis** command to specify the following range of values on the x and y axes for every place cell:

```
axis([0  4  0  30])
```

and label the axes of each of the plots. *Note an important advantage of the **for** loop - the subplot, plot, axis, xlabel, and ylabel commands only need to be written once, in order to display all 12 place fields!*

> **Q3. Drag a copy of the figure of 12 place fields into this document.** What is the overall shape of the place fields? Roughly how wide is a typical place field, in meters? Is the maximum firing rate the same for all 12 place fields?

(4) View sample spike trains from Trial 1

The sequence of action potentials recorded from a neuron in response to a stimulus is sometimes referred to as a **neural spike train**. In this part, you will view the neural spike train for four (carefully selected) place cells during the single trial that you viewed in **part (2)**. These four cells correspond to cell numbers 25, 23, 7, and 28, among the set of 30 place cells. View the spike trains by adding a statement to your script that executes the following script that has been provided for you: **showSpikeTrains**

Q4. Drag a copy of the figure of spike trains into this document. Do these place cells fire action potentials throughout the animal's traversal of the track, or are they concentrated in certain time intervals? Roughly speaking, where do you think the place fields for these four cells are located along the track?

(5) Calculate the preferred position for each place field -- that is, for each place cell

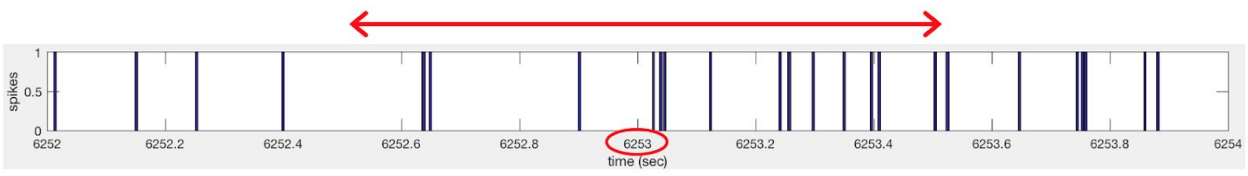
We have so far seen how place cells can **encode** the position of the animal in the environment through increased spike activity when the animal enters the place fields of these neurons. The aim of the rest of this lab is learn how we could **decode** neural activity to recover the position of the animal. If this can be done, it would suggest that place cells adequately encode the position of an animal in space. The basic idea is to ***determine which place cell has the highest firing rate at each moment, and use the preferred location for this place cell to infer the current location of the animal***. The first step in this analysis is to determine the preferred location for each place cell from its place field. Remember that the place fields for each of the 30 cells are stored in the rows of the **placeFields** matrix. Use the **max** function to create two vectors, **maxFR** and **maxPlace**, that store the **maximum firing rate** and the **index of this maximum rate**, for each place field stored in the **placeFields** matrix. ***Hint:*** *this can be done with a single MATLAB statement!* Then add the following code statement to convert the *indices* stored in **maxPlace** to the corresponding *positions* along the track, stored in the **posBins** vector *(we will illustrate how this statement works in lab):*

```
maxPlace = posBins(maxPlace);  % maxPlace originally stored INDICES of preferred places;
                               % now it stores the POSITIONS of preferred places in meters
```

In the Command Window, view the contents of the **maxFR** and **maxPlace** vectors to be sure they "make sense" *(you don't need to include this in your script).*

(6) Calculate average firing rate over an extended time window

To determine which place cell has the highest firing rate at each moment, we need to compute the firing rate of each of the 30 place cells, at each point in time. Firing rate is typically expressed as the number of spikes per second (Hz), so one strategy to compute firing rate is to count the number of spikes that occurred within a one-second time window around each time step. The figure below shows a snippet of the spike train for cell #7, from 6252 to 6254 seconds. The red arrow spans a one-second time window around 6253 seconds. Within this one-second window, we can count the number of spikes to determine the firing rate in Hz (note that in this figure, some spikes occur so close together in time, they are not separated in the plot).



spikesTrial1 is a 30 x 571 matrix that stores information about the number of spikes recorded within fixed time intervals, for each of the 30 cells. Within a single row of the **spikesTrial1** matrix, each location stores the number of spikes recorded from a single cell, within a fixed 33.3 msec window around each moment in time. A one-second time window would span 30 locations

within a row of the **spikesTrial1** matrix (30 * 33.3 ms = 1000 ms = 1 sec). Neural activity was recorded during Trial 1 over a total time frame that spanned 571 of these 33 ms time steps (the number of columns of the **spikesTrial1** matrix), which corresponds to about 19 seconds. The following values show a sample sequence of spike counts over 30 locations of one row of the **spikesTrial1** matrix. The total number of spikes in this one-second time window is 13, yielding an estimated firing rate of 13 Hz.

0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 1 2 0 1 0 0 0 2 1 0 1 1 1

To see how you can write MATLAB code to compute the firing rates at every time point and for every place cell, consider the code example shown below, which creates a 3 x 10 matrix of integers that we can think of as containing spike counts in each of 10 time bins (the columns) for each of 3 different neurons (the rows). From that matrix of spike count data, the code below then constructs a 3 x 10 matrix named **meanVals** to store the average of the 5 values around each location of the data matrix -- the average of a "moving window" within the same row of each matrix location. In order to avoid accessing elements outside the bounds of the matrix, average values are only computed for columns 3 through 8. Execution of this code snippet yields the results shown below the code. This code illustrates ***nested for loops***, in which an "inner" **for** loop is placed inside an "outer" **for** loop. For each value of i (the control variable for the outer **for** loop), the entire inner **for** loop is executed.

```
data = [1 3 2 4 5 0 2 1 0 3; 3 1 5 4 0 2 3 0 1 2; 4 0 2 1 5 2 1 3 2 4]     % create a sample matrix
meanVals = zeros(3, 10);
for i = 1:3                    % for each row of the matrix--each recorded neuron
    for j = 3:8                % for each column (time bin) from 3 to 8 (avoids going out of bounds)
        meanVals(i, j) = mean(data(i, j-2:j+2));    % store the average of 5 locations centered on j
    end
end
meanVals
```

```
data =
    1    3    2    4    5    0    2    1    0    3
    3    1    5    4    0    2    3    0    1    2
    4    0    2    1    5    2    1    3    2    4

meanVals =
    0    0    3.0    2.8    2.6    2.4    1.6    1.2    0    0
    0    0    2.6    2.4    2.8    1.8    1.2    1.6    0    0
    0    0    2.4    2.0    2.2    2.4    2.6    2.4    0    0
```

Think about how the same code structure can be used to calculate the firing rates at each moment, for each of the 30 place cells. A few tips:

1) create a matrix named **ratesTrial1** of size 30 x 571 to store the firing rates
2) use a window that includes 15 samples on either side of each time step (i.e. replace j-2:j+2 with j-15:j+15 in the above code), and use the **sum** function to count the spikes *(this actually gives 31 samples around each location, but that's ok for this problem)*
3) compute the firing rates over columns from 16 to 556, to avoid going out of bounds

To check whether your calculation of the firing rates is reasonable, plot the firing rates for the same four cells whose spike trains were viewed in **part (4)**. To do this, first open a new figure window that has a width of 800 and height of 600. Use **subplot** to define an arrangement of four display areas, with four rows and one column. Create a vector to store the cell numbers of the four cells, [25 23 7 28], which indicate the four rows of your **ratesTrial1** matrix to be plotted. Construct a **for** loop to display the four plots of firing rate as a function of time (use the time values stored in the **timesTrial1** vector).

> **Q5. Drag a copy of the figure of firing rates into this document.** Examine the spike trains (from **part (4)**) and the firing rate plots that you just created in this part. What is the relationship between the plots in the two figures?

(7) The finale! Decoding the neural activity to recover the animal's position

Our strategy for determining the animal's position from the neural activity is to ***determine which place cell has the highest firing rate at each moment, and use the preferred location for this place cell to infer the current location of the animal***. To do this, first compute the maximum values and their indices, ***within each column (time bin)*** of the **ratesTrial1** matrix:

```
[maxVal  maxInd] = max(ratesTrial1);
```

Recall that the **maxPlace** vector stores the preferred locations for each of the 30 place cells. Given the indices of the place cells with the highest firing rates stored in **maxInd**, you can now determine the location of the animal at each moment with a single MATLAB statement:

```
posTrial1 = maxPlace(maxInd);
```

(We will illustrate how this statement works in lab.)

Your final task is to plot the computed and actual trajectories of the animal, superimposed in one graph. Open a new figure window (the default size is ok) and plot the computed positions (**posTrial1**) and measured positions (**animalPosTrial1**) as a function of time (**timesTrial1**). Use **hold on** to show both plots together, and use a different color for the two plots. Given that the computation of firing rates was only done for columns 16 through 556 of the **spikesTrial1** matrix, we recommend that you just display this region of the two position plots. The example below also shows how you can draw a thicker line for a plot using the **LineWidth** property:

```
plot(timesTrial1(16:556), posTrial1(16:556), 'r', 'LineWidth', 2)
```

> **Q6. Drag a copy of the figure of computed and measured positions into this document.** How well was the position of the animal recovered? Were there any obvious errors? Why might errors occur in this decoding analysis?

We can infer from the analysis in this lab, that the ***neural activity of place cells in the hippocampus can, in principle, adequately encode the location of an animal in its environment.***

Final code and document submission

While you are still working on your code, store a copy of your current **placeLab** folder inside the **cs125** folder on the accounts of both partners on the CS file server. Before submitting your final code, add comments throughout your code, describing what is done in each section of the code, and be sure to label the axes of all your figures, including units.

> **Q7.** When you have finished this lab, share this Google document with Mike and Ellen, and also **drag your placeLab folder, with the final placeScript.m code file, into the drop subfolder** of at least one of the partners, and indicate here, where the final folder is located.