NENS 230: Analysis Techniques for Neuroscience using MATLAB
27 September 2011

Assignment One: Voltage Trace MATLAB Warm-Up
DUE DATE: 9am, Monday, October 3rd

Please submit this assignment to nens230@gmail.com, with the subject line "hw1". Detailed assignment description is below, but in short, you will submit two scripts, *hw1.m* and *VoltageTracePlot.m*, and a figure called *VoltageTrace* which should be in the .png format. You should send these files in a single (zipped) folder attached to your email called hw1_SUID where SUID is your SuNET ID (e.g. sstavisk).

This first assignment is meant to be a straightforward review of the MATLAB basics we covered in the first lecture, and will be similar to the in-class example. You will find a directory called Assignment1 on CourseWorks, which will contain code to generate current-clamp electrophysiology data and some useful scripts that you will add to in order to process and visualize this data. Your task is to load and clean up the current measurement data, combine the recording segments into a single vector, convert it to voltage, and make a figure showing the voltage over time data. You will also calculate and display the approximate resting membrane potential from the data.

The goal of this assignment is get everyone comfortable navigating MATLAB, loading and manipulating data, creating/editing scripts, and creating and saving a figure.

Brief neuroscience background

A critical property of a neuron is its membrane voltage, which is the potential between the exterior and interior of the cell. Typical resting membrane voltage value is -65mV, but this can briefly rise to +40mV in a 1-3ms event called an action potential or "spike". Much of our understanding of action potentials come from electrophysiology experiments where a fine-tipped electrode is used to record the membrane voltage or current of a single neuronal cell. In this assignment you will be looking at simulated data from what's called a "voltage-clamp" recording. Recall that voltage and current are related to each other using Ohm's Law: $V = IR$.

Detailed Instructions

1. Download and unzip the folder "Assignment1" from CourseWorks. Copy this directory structure and the files it contains, exactly as they are, to wherever you want your working MATLAB directory to be.
2. Make sure that this directory and its subfolders are on your MATLAB path.
3. You will notice that there are no .mat data files in the CurrentData directory, but there is an m-file called genData. Call this function by entering
   ```
   genData
   ```
   in the command window; it will generate five .mat data files in the

CurrentData directory. Make sure that you're calling the genData.m that you want to be calling!

4. These five files contain current measurements, taken every 0.1ms, from a 50ms epoch of a hypothetical patch-clamp electrophysiology experiment. The data are from a continuous recording, but the hypothetical recording equipment used saved data in 10ms segments, hence there are five separate files. Note that these data are randomly generated, which is why your classmates' data will look different from yours.
5. Create a script called hw.m. You will end up putting most of the commands you want to use to complete the assignment into this script, but it's often helpful to test commands in the Command Window first. Note that some parts of the assignment (such as looking through the data to find corrupted data) you will have to do "by hand", i.e. not programmatically in this script. As the course proceeds you will learn to do more and more in scripts and functions.
6. Add commands do your script to load the five data files, using the
   `load('filename.mat')`
   syntax.
7. You will see variables in your workspace called I_x and t_x, where x is 1 through 5, denoting the five segments of the data. The I_x vectors contain the current measurements (in nA), while corresponding elements of the t_x vectors contain the time (in milliseconds) of the measurement. Look through the current measurements in the variable editor or by plotting these vectors with `plot( I_x )` . You will see that the first few measurements in each file appear corrupted. Note how many measurements will need to be removed from each data segment (it is the same for all).
8. Add commands to your hw1.m script to modify I_1, … I_5, and t_1, …, t_5 to keep only those elements that are not corrupted (i.e. `I_1 = I_1(N:end)` where N is the index of the first good element.) Notice that you have to shrink the timestamp vectors in the same was as the current measurement vectors so that the timestamps and measurements keep their 1:1 correspondence. For example, if you remove the first N elements of I_3, you should also remove the first N elements of t_3. Being careful to maintain the implicit alignment of related data is a common theme of data analysis.
9. You will notice that t_1 is a bit bizarre, and needs some special attention. The orientation of this vector is different from all of the other vectors; use the transpose apostrophe ' command to fix it. Add this command to your script.
10. There is also one extra element inserted into t_1 vector.
    Find which element this is; it should be obvious from visual inspection of `plot(t_1)`.
    You may find it helpful to turn on the Data Cursor, which you do by the clicking the button at the top of the figure window whose icon consists of a plus drawn over a blue curve with a text bubble coming out. Once you've enabled Data Cursor, you can click on the plot and then use the arrow keys to move point to point on the plot while it displays this point's X and Y values.

11. Excise this erroneous element using the `t_1( badElement ) = [ ]` syntax. Add this command to your script. Remember that if you call this command multiple times (to test it, for example) you will end up excising multiple elements; you can always reload the data and run your script up to this point to undo this.
12. To be thorough, check that the length of t_1 is the same as the length of the I_1 using the `length( yourVector )` command at the command line.
13. You will want to now create variables **t** and **I_m** which will have the concatenated (combined) timestamps and current. Add lines to your script to define t as the concatenation of t_1, … t_5 and I_m as the concatenation of I_1, … , I_5. Make sure you are *vertically* concatenating these *column* vectors.
14. The current data you now have is in nA. Add a command to your script to convert it to Amperes.
15. You will now want to convert from current to voltage. Use Ohm's Law, and assume that your recording electrode had a resistance of 8 MOhm. Add a command to your script to create a variable **V_m** which is the vector of membrane voltage, as computed by element-wise multiplication
    `.*`
    of I_m and the electrode resistance.
16. Neuroscientists often work in units of millivolts rather than volts. Thus, add a line to your script to convert your V_m from volts to millivolts.
17. Let's calculate an estimate of the resting membrane voltage of this neuron. Make a quick and dirty plot of V_m by calling `plot(V_m)` and look at it to choose a segment of measurements where there is not an action potential happening. Then, add a command to your script to define a variable called **restV** which is the mean value of the spikeless segment of V_m. For example, if there are no spikes in the first 3ms, I might estimate the resting membrane potential by calling:
    `restV = mean( V_m(1:30) )`
18. Now it's time to make a nice plot of this voltage trace. To help you, we've provided a nearly-complete script called VoltageTracePlot.m. Open this script up.
19. Read through this script to learn about how it works. You'll notice that this script requires you to have successfully defined variables t, V_m, and restV in the workspace; if these aren't there, then VoltageTracePlot.m will not successfully generate the figure.
20. On line 42 (note the line numbers at the left of the Editor) you'll see that in addition to a line plot of the membrane voltage, we also want to create a scatter plot of the data to see exactly what our measurements were (line plots can be deceiving, remember?) You can look at the sample figure in the Figures_Voltage directory to see what we mean. You'll need to open this .png from outside of MATLAB.
    Use the MATLAB help browser to figure out how to use the scatter command to make a scatter plot of the x-values specified by variable t and y-values specified by variale V_m. Add this command to line 42 of VoltageTracePlot.m
    .

21. On line 53 of VoltageTracePlot.m, you will want to add a line to add an appropriate y axis label to the figure.
22. Uncomment lines 62 through 70 of VoltageTracePlot.m to add a dotted horizontal line showing the resting membrane voltage to your plot. Note how this is done; you can look up the line command to understand the syntax we've used.
23. Add a call to the VoltageTracePlot script to the end of hw1.m. Recall that to call a script, you just enter the name of that script, e.g.
    ```
    VoltageTracePlot
    ```
    to its own line of code. This completes your hw1.m
24. At this point you'll probably want to `clear` your workspace and run hw1.m from the beginning to load, clean up, convert, and plot the data. You should see a nice figure appear, except that the scatterplot color is green while the line color is blue. Some might call this ugly. Open up Figure Tools (the far-right icon at the top of the figure window) and use this editing tool to change the color of the markers to blue. Feel free to further edit the figure using this tool to make it more pretty.
25. When you're satisfied, save the figure as a .png named VoltageTrace. You can do this either through File→Save in the Figure window, or at the command line with the command
    ```
    saveas( figh, 'VoltageTrace ', 'png')
    ```
26. That's it! Zip up hw1.m, VoltageTracePlot.m, and VoltageTrace.png and email it in.