NENS 230
Assignment #2
Data Import, Manipulation, and Basic Plotting

Compound Action Potential
Due: Tuesday, October 6th, 2015

Goals

- Become comfortable reading data into Matlab from several common formats
- Manipulate multi-dimensional data with matrices
- Create simple plots
- Use `fprintf` to display formatted output
- Calculate summary information such as the extrema, mean, and standard deviation from sets of data

Introduction

Cells in the nervous system use electrical impulses, known as action potentials, to transmit information. Some of the first electrophysiological recordings from action potentials were from nerves in the peripheral nervous system. In this assignment, you will be analyzing data recorded from the frog sciatic nerve. Since a single nerve bundle is made up of a bunch of different fibers (innervating different muscle pools), each of which is capable of transmitting an action potential, we call the aggregate voltage signal measured from the entire nerve the *compound action potential* (CAP).

The first piece of data is a voltage recording of the potential difference between two electrodes near the fiber, after it has been stimulated electrically at one end. We will use this data to plot a graph of the voltage trace recorded, and use Matlab to extract some information from the curve.

The stimulation used was a voltage pulse, which has two parameters: the strength (height of the pulse) and duration (width of the pulse). The second piece of data is a table containing the minimum strength (voltage) required to stimulate an action potential for different durations. We will use this to plot a strength-duration curve. In addition to these measurements, the delay of the CAP in milliseconds was measured for different electrode distances. This will allow us to estimate the conduction velocity of the signal.

Getting Started

Download the zip archive at nens230.stanford.edu/week2, uncompress it, and put it into some directory you'll be using for this assignment for example, `/Users/sstavisk/NENS230/Assignments/Week2/`. You'll see the

data files you'll need for this assignment: `actionpotential.mat, pulsedata.csv, recordings.mat`, as well as a starter script that you'll be modifying to do the assignment `assignment2.m`. All of your work for this assignment will consist of adding code to `assignment2.m`, and then turning in that plus a printout of what it does (this is done with the Publish feature of Matlab, described at the end). Also included in the .zip file is a helper function `smooth_ma.m` that you can use if you choose to do the optional 4th problem. Make sure all these files are on your path, or you won't be able to load them. If you navigate to your Week2 directory in Matlab, they should automatically be in your path.

The directions below explain each problem, but use the comments in `assignment2.m` to guide you step by step in implementing what is asked for. This template contains comments corresponding to the relevant analyses. Please put your code corresponding to a comment directly under that comment. Note that we provide an estimate of how many lines of code implementing each task requires. These are just there to help you not overthink/underthink the problem. They are *not* meant as specific constraints or requirements. There are many ways to program a solution to a given task, anything that gets the job done well is acceptable.

Problem 1

A) First, make sure to download the `actionpotential.mat` file from the zip archive at nens230.stanford.edu/week2. Load the data into Matlab, and use `whos` to check out what variables are included, and check out the dimensions of the data (via the `size` command). One thing that I find useful is visualizing the data early and often via the command line, before putting code in a script. For instance, you may find it useful to make a quick `plot` of the voltage vs. time to see what the signal looks like: `plot(time, voltage)`.

B) We are going to alter the data a bit, and then plot it. Notice how the signal only starts changing after a few milliseconds. We are going to artificially shift the signal so that the time t=0 is defined as the time when the voltage first deviates from 0. To do this, you will need to first use the `find` command to find the first index where the voltage is not zero. Then, find the time associated with this index. Finally, subtract that time from the entire time vector. Now if you make the plot of voltage vs. time, you should notice that the first deflection occurs at t = 0.

C) The time vector has units of seconds (s) and the voltage vector has units of milliVolts (mV). To make the graph easier to digest, we are going to switch the time axis to be in milliseconds (ms). Do this by multiplying the time vector by the appropriate scaling factor.

D) We are done adjusting our voltage trace. Fill in the section of code which asks you to plot the newly adjusted voltage vs. time, label it, and give it a title. Use the `grid on` command to place a dotted grid on the graph.

E) We are going to figure out the minimum and maximum voltages recorded from the CAP, and figure out when those extrema occured. You will want to use the `max` and `min` commands. Store the maximum and minimum voltage in variables named `vmax` and `vmin`, respectively, and the corresponding times in `tmax` and `tmin`. Mark the maximum and minimum with red circles on the existing plot. You can use the `hold on` command and use the syntax `plot(x,y,'ro')` to indicate that you want red circles instead of the default line.

F) We also want to print out the maximum/minimum times and voltages we found using `fprintf`. You should play around and figure out how you would like to display the information – the only requirement is that someone should be able to figure out what the maximum and minimum voltages are and at what times those voltages occured just from reading the output of `fprintf`.

Problem 2

A) First, load the strength-duration data from `pulsedata.csv` (part of the zip archive on the website). A CSV (comma separated values) file is a format for storing data that separates numbers using a "delimiter", such as a comma. Note that the first line of this CSV file contains header information, so we want to skip the first line when reading the data (we can do this via the `csvread` command). Again, you probably want to use `whos` to check out the size of the data once it is loaded to get oriented (do this for yourself, not for the assignment).

B) The first column contains the pulse voltage, in volts, while the second column contains the minimum pulse duration (in milliseconds) necessary in order to generate a compound action potential. Make a strength-duration curve by plotting the strength (voltage, y axis) against the duration (time, x axis). The plot should have circle markers and a solid line, see `doc plot` for details. Again, make sure that the code in the `assignment2.m` script has been updated underneath the appropriate comments.

Problem 3

A) We are now going to estimate the conduction velocity of the compound action potential. The *conduction velocity* is the speed at which the signal travels down the nerve fiber. We measured the CAP signal at different distances along the nerve fiber, and have stored the data in `recordings.mat`. The matrix `traces` contains the recorded signal from four different electrodes placed along the fiber (with the corresponding time of each voltage sample, in seconds after stimulation, saved in the `traceTime` vector). The distance of each recording electrode (in centimeters) from the stimulation site is stored in the `distances` vector. We can plot all four traces on the same plot easily by running the command `plot(traceTime,traces)`. If `traces` is a matrix, this command plots the *columns* of traces against the time vector.

B) First, we need to estimate the time at which the action potential reached each electrode. Since the action potential is so much bigger than the recording noise (as you can see if you made the plot), we can just find the time at which the maximum voltage occurs for each of the four recorded traces (for the purpose of this problem, you can ignore the few milliseconds of rise time it takes to reach this peak). For each column of the `traces` matrix, use the `max` command to find the index of the maximum value along that column. Then, use that index to find the time at which the maximum occurs (similar to Problem 1). Store these times in a vector called `maxTimes`.

C) We will compute the conduction velocity using the familiar equation, distance = rate $\times$ time. We have the times (in seconds) of the signals stored in `maxTimes`, and the distances (in centimeters) stored in the `distances` vector. Use this to compute the estimated conduction velocity in meters per second. Don't forget to use elementwise operations, and to multiply by the appropriate scaling factors so that the units work out.

You should end up with a vector of 4 different velocity estimates, which you should name `velocity`. These velocities should be on the order of 0.3 meters/second.

D) Compute the mean and standard deviation (using the `mean` and `std` commands) of the four measurements and store them into variables named `meanVelocity` and `stdVelocity`, respectively. Finally, use `fprintf` to print the mean and std. dev. of our measurements to the command line. Make sure to include units in your output (as in, write something like `m/s` into your string output — there's not an automatic way to display units, since Matlab just treats these data as unitless numbers). When finished, check to make sure your code is in the appropriate places in the `assignment2.m` script.

Problem 4 (optional)

This problem is optional. If you're new to programming, we understand the first 3 parts may have taken you a long time to get through. If so, feel free to not do this part (you'll still get full credit). But if you're interested in a quick lesson on smoothing data, proceed.

A) The following example uses Matlab to smooth a noisy signal. We are going to add noise to the voltage trace from Part 1 to simulate a noisier recording, and then using a moving average filter to smooth the signal and remove some of the noise. First, generate a noise vector that has the same dimensions as the voltage vector. Do this with the `randn` command, since we want the noise to come from a Gaussian, or normal, distribution. In addition, we want the noise to have a standard deviation of 25. We can do this by just multiplying the vector generated by randn by 25.

B) When you have your noise vector, add it to the voltage vector. Store this new variable as `noisyVoltage`. Make a quick plot (via the command line) of noisyVoltage vs. time to see what your new signal looks like. Compare it to the figure from Part 1.

C) We will now use the `smooth` command to filter the noisy signal. If your Matlab doesn't have the `smooth` command, which is part of the curve fitting toolbox, you can instead use the `smooth_ma` function that we've provided (just call `smooth_ma` anywhere we refer to `smooth`. By default, the `smooth` command uses a *moving average* filter to smooth the data. Generate a smoothed signal by running the `smooth` command on the noisyVoltage signal and store the result as `smoothVoltage`. Make a quick plot of smoothVoltage vs. time and compare it to noisyVoltage.

D) One of the *parameters* of the `smooth` function is the window size, which controls the strength of the smoothing (the default value is 5 elements). Call the `smooth` function again, and this time change the window by giving a second argument to the `smooth` function. Plot the data and compare it to the noisy and original data. Try a bunch of different values for the window size until you decide on one that works best. If the parameter is too small, there won't be enough smoothing and the signal will still have a lot of noise. If it is too large, however, you will start to smooth out the CAP signal that we are trying to see.

E) Once you have settled on a parameter for the `smooth` function, fill in the code for Part 4 of the `assignment2.m` script. The final plot asks you to plot both noisyVoltage and smoothVoltage on the same

axes but with different line styles. You may have check out the documentation by typing `help plot` to see how to do this.

Submission

When you are finished, publish your work by running the command: `publish('assignment2.m','pdf')` from the command line. This should generate all of the figures and along with the code and comments from the script put them in a PDF document in a folder named `html`. Email your script, `assignment2.m`, and the generated PDF document to `nens230@gmail.com` to submit your assignment.