

MATLAB project 1: Spike detection and plotting

The goal of this project is to make a simple routine (a set of MATLAB commands) that will allow you to take voltage data recorded near a neuron or axon and determine the times that an action potential occurred in that axon or neuron. Because action potentials are the units of communication between neurons in different parts of the brain, the times of action potentials is the fundamental neuronal measure of most of neurophysiology, especially *in vivo*.

For now, some voltage data will be given to you, along with a list of action potential times, and this will allow you to develop and test your routine. However, later in the course, you will collect your own voltage data (where the times of the action potentials are not known) and run your routine on that data.

The basics of MATLAB functions

You should already know how to start MATLAB, make variables in the workspace, call functions (like “plot”), and make respectable plots. (*If you do not know this, go back and do MATLAB Project 0.*) To make your spike detection routine, you will need to learn how to put MATLAB commands together into a function (called an “m file” in MATLAB) so that you can later run that function on any set of voltage data.

From the MATLAB command window, let’s first run a function to detect spike times that we have already written for you. This function is not a very good spike detector (we want to leave that job to you!), but it will give you a basic structure for how your routine should look.

First, load the first test data into the workspace:

```
>>
>> clear all;
>> load('/mit/9.02/matlab/project1/data01');
>> whos
```

Name	Size	Bytes	Class
actualSpikeTimesMS	1x10	80	double array
timesMS	1x20001	160008	double array
voltagesUV	1x20001	160008	double array

Grand total is 40012 elements using 320096 bytes

Congratulations, you have just loaded some data into your workspace.

Variable

Contents

<code>actualSpikeTimesMS</code>	list of times that spikes occurred (units are msec)
<code>timesMS</code>	list of all the times (in msec) that a voltage was measured
<code>voltagesUV</code>	all of the measured voltages (units = microvolts = μV)

Now, run the poor routine provided to you to try to detect and report the times of spike from this data. Note, the routine does not use the actual spike time (that would be cheating!). To run this function, you simply type its name and provide the input data and a variable to hold the output data. Here is an example of what to type at the command line:

```
>>
>> detectedSpikeTimesMS = poorRoutineToFindSpikeTimes(timesMS, voltagesUV);
>> whos
>>
```

Type “`help poorRoutineToFindSpikeTimes`” in the Command window (without quotes) for more info on this particular function (just as you can get help on any other function).

At this point, you should inspect the code in function “`poorRoutineToFindSpikeTimes`” to understand what a function looks like.

To do this, open “`/mit/9.02/matlab/project1/poorRoutineToFindSpikeTimes.m`” in the editor of your choice. Using emacs under athena:

```
athena% emacs /mit/9.02/matlab/project1/poorRoutineToFindSpikeTimes.m &
```

(If you are not familiar with emacs, we recommend the brief emacs tutorial at <http://www.math.utah.edu/lab/unix/emacs.html>)

Spend some time going through the function line by line to see what it does (also see the information in the project assignment below before you look too closely at the file). As the function name indicates, this is not a good spike detector (as you should see when you figure out how the function works). But it is a fine example of a typical MATLAB function that has the same inputs and outputs as the function that you should create for your project 1 assignment.

Assignment for MATLAB Project 1

Goal 1 of Project 1 is to build a function that can do basic spike detection. Along the way, you will need to learn how to write your own MATLAB function. Believe me, this knowledge will be useful outside this course! Along with providing us an electronic copy of your routine, please provide a table showing the performance of your routine across the all of the test data sets (see below). **This is not simply an exercise, as you will need to use your routine in the FLY labs near the end of the course.**

Goal 2 of Project 1 is to plot one of the provided data files showing the voltage vs time plot for the data (similar to Project 0) along with tick marks showing the times that spikes were detected. It is even better if you can also plot the waveforms of all of the detected spikes. An example of what this plot might look like (for a different data file) is shown at the end of this document.

Writing your first MATLAB function

For MATLAB project 1, the skeleton form of your function should be something like:

```
function [spikeTimesMS]=myFindSpikeTimes(timesMS,voltagesUV) % line 1
    <<my command/call 1>>                                     % line 2
    <<my command/call 2>>                                     % line 3
    ...
    spikeTimesMS = ??;                                       % line 4
return                                                         % line 5
```

If you are already familiar with MATLAB (or related programming languages), the skeleton provided above should be clear. If not, do not worry, we will walk you through it. Remember, you can always go to the MATLAB help window to learn more about m files and functions in MATLAB. Let's go through it line by line:

There are 5 lines of code in this function skeleton, but that is just for illustration. Your function can have any number of lines. The only line that should be in the same place as this example is line 1.

Line 1. This is the most important line in the function. The word “function” tells MATLAB that this is the start of a function. The variable name in brackets is the name of the variable in the function whose data values should be returned by whoever called the function. For example, when you called the function from the Command window, with the line:

```
>> detectedSpikeTimesMS = poorRoutineToFindSpikeTimes(timesMS,voltagesUV);
```

you asked MATLAB to place the output of the function called “poorRoutineToFindSpikeTimes” into a variable called “detectedSpikeTimesMS”. Note that you did not have to create this variable before you ran the function – MATLAB created it for you. If you had created a variable with the same name before you ran the function, MATLAB would have overwritten any data in your variable with the data output from the function.

The next part of line 1 after the “=” symbol is the name of the function. The name can be almost anything you want. However, **any function you create should be saved as an “m file” with the same name.** For example, the function “poorRoutineToFindSpikeTimes” is contained in the file “poorRoutineToFindSpikeTimes.m”. **For now, your life will be a whole lot easier if you do not deviate from this convention!**

The last part of line 1 (in the parentheses) is the data that will be passed into the function. These are the names given by the function to the data that is provided by whoever called the function. In this case, the first data in the list is given the name “timesMS” and the second is given the name “voltagesUV”.

Note that a function can have any number of input arguments (or no arguments at all). In this case, the number of arguments is two. That does not mean that only two numbers can be passed in, because this function is meant to expect two vector arguments of any length (in the example above, the function accepted 20,000 numbers as input (10,000 numbers in each of the two vectors)).

Line 2 and line 3 are meant to illustrate that you can do operations on the data, make calls to other functions, etc.

Line 4 indicates that, at some point in the function, you should assign some data to the output variable(s) of the function (in this case, to the variable called “spikeTimesMS”).

Line 5 simply says to return control of the program to whoever called the function. This line is not needed, as MATLAB will automatically return when there are no more lines in the function, but it is nice to help you understand what is going on.

Getting started

- **Option 1:** You can use the file called “poorRoutineToFindSpikeTimes.m” as a starting point. To do this, make a copy of this file and place it in your ~/9.02 directory:

```
athena% cp /mit/9.02/matlab/project1/poorRoutineToFindSpikeTimes.m ~/9.02/
athena% cd ~/9.02
athena% emacs poorRoutineToFindSpikeTimes.m &
```

- **Option 2:** You can open a fresh function and just start typing.

```
athena% cd ~/9.02
athena% emacs mySpikeTimes.m &
```

Spike detection

Detecting spikes from voltage data is a non-trivial problem and research is still devoted to this problem today (for reviews on this topic, see the 9.02 Stellar site MATLAB supplemental section). However, some basic methods of spike detection are straightforward and can be written especially well by MIT undergraduate students! Part of the purpose of this project is to let you think on your own about different ways that you might detect spikes in a continuous voltage trace. At the end of the day, you do not need a perfect routine, but we have provided some test data for your routine and it should be able to do a reasonably good job* (see below) of detecting the main spike signal in each of those test data sets.

Hint: one MATLAB function that might be especially helpful to you is “find”. Try “help find”. “find” will return the indices of each element in a vector that meets a condition.

For example:

```
>>
>> x = [0 5 0 12 1 -4];
>> ind = find(x>2) % find all the element of x that are >2 and return the
                    element number

ind =          % here are the element numbers (the 2nd and 4th elements of x)

      2      4

>> x(ind)          % here is how you can easily see all those elements

ans =

      5      12

>> x([2 4])        % here is another way to see elements 2 and 4

ans =

      5      12
```

```
>> x([2:4])           % compare with above.  
                        Here you get all the elements from 2 to 4.
```

```
ans =
```

```
     5     0    12
```

```
>>
```

*What do we mean by a “reasonably good job?” Again, this is not a simple answer, but for the purpose of this course, we have quantified performance of a spike detector by two numbers:

- 1) the percent of spikes that are correctly detected (PCD) = the percentage of spikes correctly detected (A correctly detected spike is defined here as one with a reported time within ± 2 ms of the actual spike onset time.)
- 2) the false alarm rate (FAR) = the number of spikes reported per second that did not actually occur.

Clearly, the ideal detector has a PCD = 100% and a FAR = 0 spikes/sec. To help you gauge the performance of your routine, we have written a routine that computes these two numbers:

```
>> [PCD,FAR]=testSpikeDetector(detectedSpikeTimesMS, actualSpikeTimesMS);
```

For each of the test data files provided, you should be able to obtain the PCD > 90% and FAR < 2 spikes/sec (without changing the routine between runs of different data of course!!)

The data files that we have created for you are:

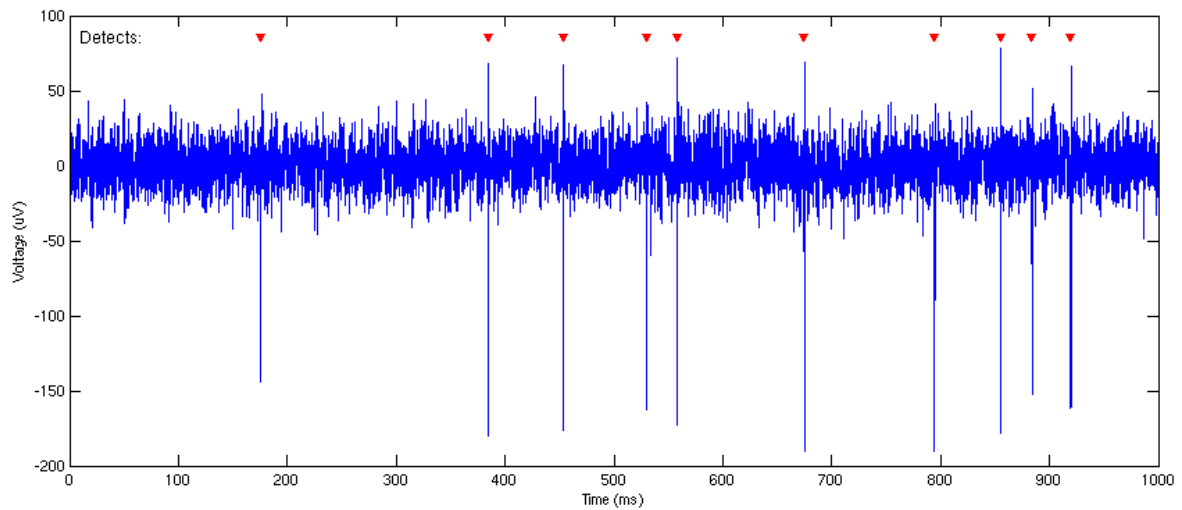
```
data01.mat  
data02.mat  
data03.mat  
data04.mat
```

The data in each file can be loaded and tested like this:

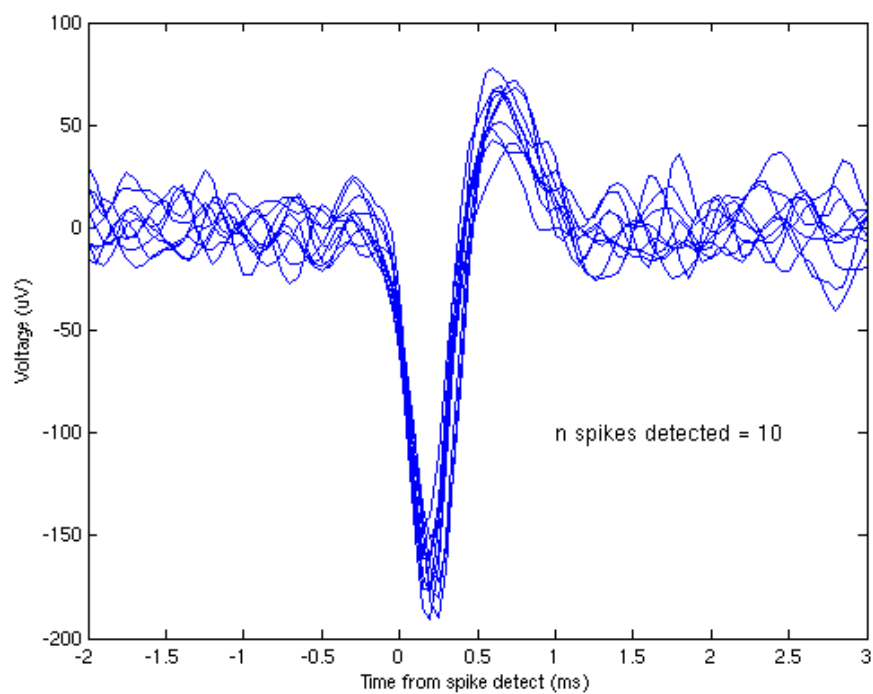
```
>> clear all;  
>> load('/mit/9.02/matlab/project1/data01'); % load the variables in the  
                                              file into the workspace  
>>  
>> [detectedSpikeTimesMS] = runMySpikeDetectRoutine (timesMS,voltagesUV);  
>>  
>> [PCD,FAR]=testSpikeDetector(detectedSpikeTimesMS, actualSpikeTimesMS)
```

Plotting detected spikes example (Goal 2 of project 1).

Data and detected spikes in file: 'project1data_xx'



Waveforms of all detected spikes in file: 'project1data_xx'



Summary performance of my spike detector

Data file	% correct detect	False alarm rate (spikes/sec)
project1data_01	100%	0
project1data_02	100%	0
project1data_03	100%	0
project1data_04	93%	0