Computational Assignment 1: Introduction to programming and scientific computation in Matlab

Computational assignments contain questions for all background levels of programming and mathematical experience (Levels 1, 2 and 3). You will be graded only on those questions that correspond to your agreed-upon background level (if you don't remember your level, ask me). However, it may be necessary to answer Level 1 questions (or at least be *able* to answer them) in order to answer Level 2 questions. Similarly, Level 2 capabilities may be needed to answer Level 3 questions.

In general, there is one place you should always check first before asking questions of the instructor or even other students, which is the **Help** documentation that comes with Matlab. This documentation is online, along with tutorial videos and other helpful things, at: https://www.mathworks.com/help/matlab/index.html.

If you have access to the Matlab program itself, you can, for example, get help using Matlab's **plot** command, by typing "**help plot**" at the command prompt (the command prompt looks like this: **>>**), or type "**doc**", then hit Return, then search for **plot** in the documentation. Try copying the little snippets of code examples that are included in the Help articles. Paste them into the Command Window and hit Return, and you'll see what the code does. Then change it a little bit, and see what that does. That seems to be the best, most practical way to learn how to do what you need to do.

To get oriented to Matlab for the first time, it would be a good idea to begin with this video: https://www.mathworks.com/videos/getting-started-with-matlab-101684.html

The first thing to do after the video is to explore the hierarchical structure of the Matlab documentation. This documentation structure lets you see the broad categories of things that Matlab can do.

For a first example, go to the documentation (https://www.mathworks.com/help/matlab/index.html). Then click on "**Getting Started**", to find the very first things a new Matlab user should learn. Since it will help to have a standard way to describe doc pages in the future, here is how I would describe the "location" of this **Getting Started** page within the documentation:

Matlab / Getting Started

The Getting-Started-with-Matlab tutorials will teach you most of what you need to know for the Level 1 aspect of Assignment 1.

Collaboration Policy: I expect students to try to do these problems individually, and to submit their answers in their own writing (or their own code). ***Do not copy and paste from any other student. If your code is identical with someone else's, that will be grounds for declaring an Honor Code violation. Code should never be identical if you write it yourself.***

However, you should feel free to discuss problems with other students, and you should feel free to modify code from the Help documentation articles as you see fit, as long as you do that yourself (i.e., don't copy code from the Help documentation and then send it to another student for their use – they should look that information up themselves).

How to submit your work: Just copy and paste your code, your figures, or your written answers into the spaces provided in this document. Then email to me by the deadline at:
psimen@oberlin.edu
Please name the file "*YourLastName_YourFirstName*_NSCI360_HW1". Files can be in Word, text, Pages (Apple), or PDF format.

PROBLEMS

LEVEL 1 ONLY

1. Intro to Matlab. Go to this website and watch a short video on Matlab: http://www.mathworks.com/videos/working-in-the-development-environment-69021.html?s_cid=learn_vid

From watching the video, explain how can you get a sort of spreadsheet view for examining the values of variables:

2. Create variables that store vectors of numbers. Create a variable **A** consisting of a row vector of the following values: 1, 2, 8, 52. Paste the code you used to create it below. After that, do the same for a column vector called **B** of the following values:
8
7
6
24

*Hint: read through **Matlab / Getting Started / Matrices and Arrays** in the Help documentation to find out how to do this.*

Paste your code here:

```
    B = [8; 7; 6; 24]
```

3. Create a *script*. Create a *script* that saves commands in a separate file (later, you can automatically run all those commands by typing the name of the script at the command prompt >>). In your script, create two matrices, each of which is four rows by four columns in size, consisting of whatever numbers you want, and then add the two matrices together. Note that you can write comments for yourself in a script by typing the percent character: "%". After the % sign, any text is ignored. At the top of your script, write a comment that contains your name, and the name of your script. At the end of your script, use the **save** command to save the variables you created into a file named "MyData.mat". After the script runs, type "clear all" to get rid of all variables in the workspace. Then type "load" to get the variables you saved back into the workspace.

*Hint: read through **Matlab / Getting Started / Programming and Scripts** for some useful info, and also type "help save" at the command prompt to see what **save** does.*

Paste the contents of your script file here:

LEVEL 1 and LEVEL 2

4. Concatenation. Write a script that creates the variables a = [1 2 3], b = 4, c = 5, d = [6; 9], e = [7 8]. Create the matrix m = [1 2 3; 4 5 6; 7 8 9] in a single line of code through horizontal & vertical concatenation of a, b, c, d & e. Now change the script into a function. What is difference between these two programs?

Attach the separate files for the script and function to your email submission (you should thus have two files with ".m" as a suffix.

```
m = vertcat( a, horzcat (b,c, d(1,1)), horzcat(e,d(2,1)))
```

5. Indexing vectors. Write a *function* (not a script) whose input is a vector, and which returns the indices of elements that are greater than 5. For example, the vector [5 6 3 2 7] has elements greater than 5 at positions 2 and 5, but nowhere else.

*Hint: you will need to use the **find** function along with a logical comparison, so search for **find** in the Help documentation's search bar, or type "help find". Also, you should consult **Matlab / Programming Scripts and Functions / Functions / Function Basics / Create Functions in Files.***

Paste your code here:

```
function f = find(X > 5, n)
   f =  find(X>5,n);
end
```

6. Number Classification. Write a function that will indicate (using the **disp** command) whether an input is negative, positive, or zero, as well as even or odd. No output arguments are necessary.

*Hint: you will need to look up how to write "conditional" statements in **Matlab / Getting Started / Programming and Scripts / Loops and Conditional Statements.***

```
function y = defnum(x)
   if x >0
      if rem(x,2) == 1
         disp('positive, odd');
```

```
        elseif rem(x,2) == 0
            disp('positive, even');
        end
    elseif x < 0
        if rem(x,2) == -1
            disp('negative, odd');
        elseif rem(x,2) == 0
            disp('negative, even');
        end
    else x = 0
        disp('zero');

    end
end
```

LEVEL 2 and LEVEL 3

7. *For* loops. Use a **for** loop to print out the even numbers from 2 to 10. Here's the tricky part: Inside the loop, the only command should be the name of the variable you use in the **for** command.
Hint: look up the ***:*** *(the colon) operator in Matlab. This is really useful in Matlab for creating sequences of numbers, which is what Matlab is all about.*

Paste your code here:

```
for x = 2:2:10
        disp(x);
    end
```

8. Plot out a signal. First, create a vector of numbers called **t** representing different moments in time, equally spaced from time 0 to time 10 in steps of size 0.1 (you could think of these numbers as representing time in seconds, or minutes, or whatever). *Hint: use a* ***for*** *loop, or better yet, Matlab's ":" notation for creating arrays.* Create a variable of the same size as **t** called **Sig1**, which is equal to the sum of two sine waves with length equal to the length of **t**, *each with different frequencies*. Now do the whole thing over again, and divide time into much smaller time steps of size 0.001.
Recommendation: use the ***sin*** *function and look it up for help on its use; look up* ***plot*** *and the* ***:*** *operator.*

Paste your code here, and try to place a copy of your output plot figure after the code (save the figure as a pdf file and insert it):

```
t = 0:0.1:10
Sig1 = sin(t) + sin(2*t)

plot(t,Sig1)
title('Sig 1')
xlabel('t')
ylabel('x(t)')
axis tight
figure

t = 0:0.001:10
Sig1a = sin(t) + sin(2*t)
plot(Sig1a)
title('Sig 1a')
xlabel('t')
ylabel('x(t)')
axis tight
```

9. Plot out a more complex signal. As in Problem 8, create **Sig2** to be the sum of three sine waves, each with a different frequency and phase, and with a small amount of random noise added to it.
Hint: Frequency is determined by one of the arguments you provide to the ***sin*** *command; you can change the phase by adding something to the inputs. "Frequency" is how many times per second a sine wave goes from maximum amplitude to minimum and back to maximum; "phase" specifies how far a signal is from its maximum value at some given time (such as time = 0 seconds). If you still don't understand, look up what frequency and phase are by searching in the Help documentation or wikipedia. Ask me about them if all of that fails. Finally, use* ***normrnd*** *or* ***randn*** *to create the random noise to add to the signal (one noise sample per time step).*

Paste your code here, and try to place a copy of your output plot figure after the code (save the figure as a pdf file and insert it):

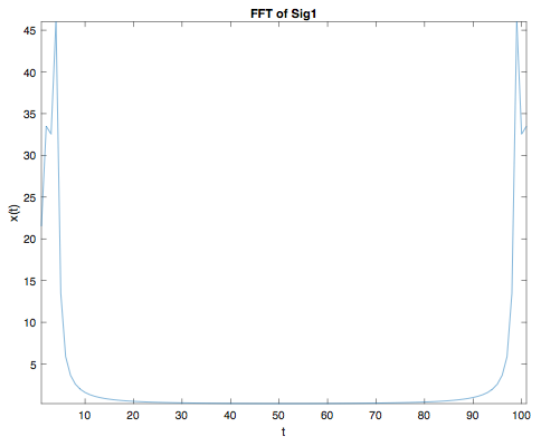
```
t = 0:0.1:10;
mu=0;sigma=1;

Sig1 = sin(2.*(t+randn(1,1))+4);
Sig2 = sin((t+randn(1,1)/2)-6);
```

```
Sig3 = sin(((t+randn(1,1)).*(2/3))-1);

Signal = Sig1 + Sig2 + Sig3;

%SignalNoise = Sig1.*randn(1,length(t)) + Sig2.*randn(1,length(t)) +
Sig3.*randn(1,length(t));

SignalNoise = Signal.*randn(1,length(t));
plot(SignalNoise)
```

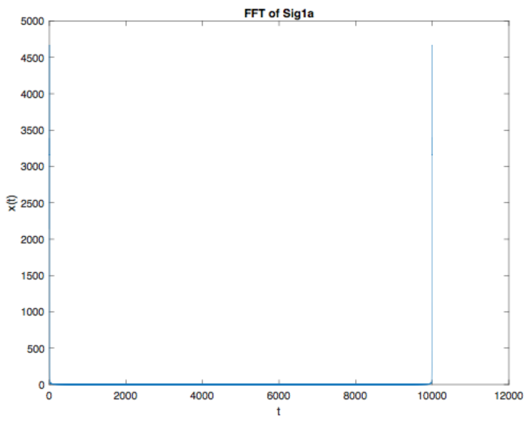
LEVEL 3 ONLY
10. Power spectra. Find the power spectrum for each signal in problems 8 and 9. Plot this power spectrum out in a figure. Save the figure as a pdf file. Paste the pdf file below. This is a useful thing to be able to do when writing a paper. It may also prove that some aspects of Matlab are kind of screwed up. You may need to crop the figure somehow to get it to fit into this document.
*Hint: look up **fft**, which is the Help documentation topic I looked up in class and used to show a power spectrum example.*

Paste your code and its graphical output here:

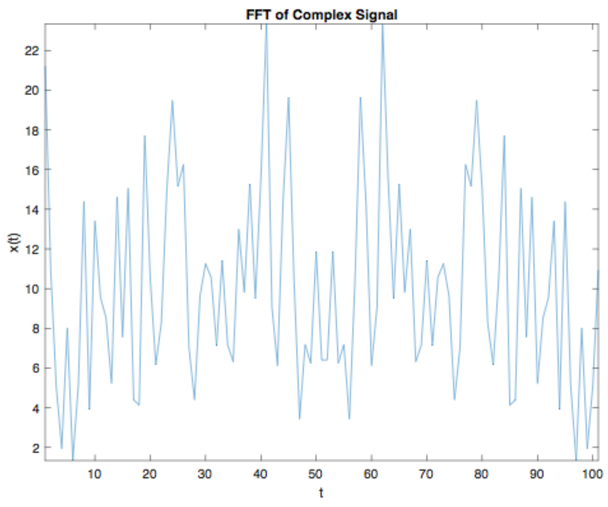
```
plot(abs(fft(Sig1)));

title('FFT of Sig1')
xlabel('t')
ylabel('x(t)')
axis tight
figure;
```



```
plot(abs(fft(Sig1a)));
title('FFT of Sig1a')
xlabel('t')
ylabel('x(t)')
figure;
```



```
plot(abs(fft(SignalNoise)));
title('FFT of Complex Signal')
xlabel('t')
ylabel('x(t)')
axis tight
```

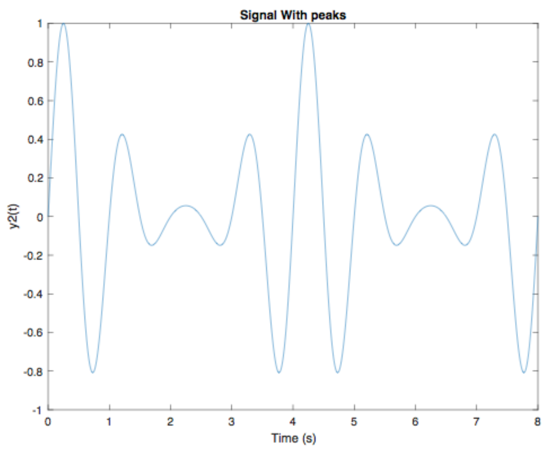


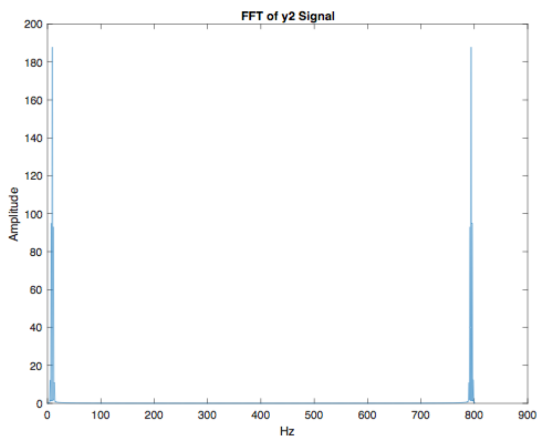
```
figure;
```

11. Signal construction. Create a signal that has strong peaks in its power spectrum at X Hz and Y Hz. But make sure there is a little bit of power at all other frequencies as well.

Paste your code and your output plot here: peaks at t = 0.25, t = 4.25

```
t=0:0.01:8;   %Time vector
w = 1; %pulse width
d= w/4:4*w:10; %delay vector
y2=(pulstran(t,d,'gauspuls',w));

plot(t,y2);
xlabel('Time (s)')
ylabel('y2(t)')
title('Signal With peaks')
figure;

plot(abs(fft(y2)));
title('FFT of y2 Signal')
xlabel('Hz')
ylabel('Amplitude')
```

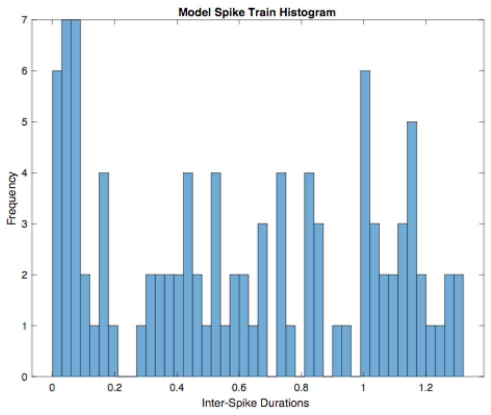


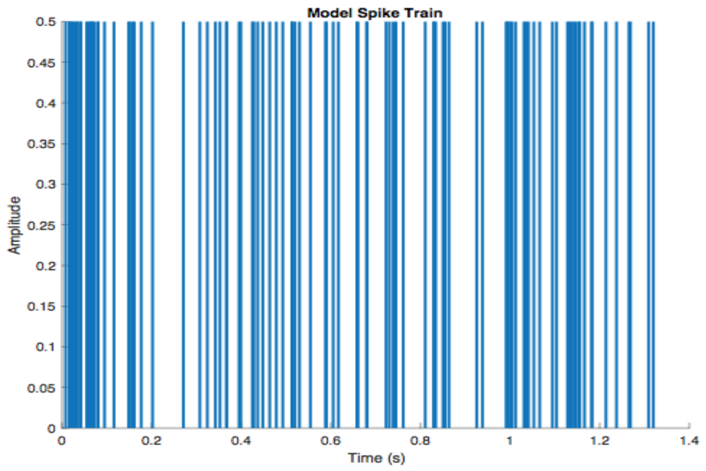


12. Create model spike trains and plot them as spike raster plots. Use the **exprnd** command in Matlab's Statistics Toolbox to obtain an inter-spike time: the time between when a neuron fires one action potential and then fires the next one. In fact, create a whole sequence of inter-spike intervals. Make a time vector **t** that encodes the time of each spike (you may want to use the **cumsum** command to find these times, by applying it to the output of **exprnd**). Now, use the **line** command to plot a tiny vertical line at the time of each spike. Also, create a separate figure in which you plot a histogram of the inter-spike durations (use the **hist** command).

Paste your code and your output plot here:

```
for i = 1:20
   n(i) = exprnd(0.005);
   t = cumsum(n)
end
for j = 1:length(t)
    line([t(j) t(j)],[0 0.5], 'LineWidth', 2 );
end

xlabel('Time (s)')
ylabel('Amplitude')
title('Model Spike Train')
figure;



histogram(t, 'BinWidth',0.03)
xlabel('Inter-Spike Durations ')
ylabel('Frequency')
title('Model Spike Train Histogram')
```


Model Spike Train Histogram
Frequency
Inter-Spike Durations


Model Spike Train
Amplitude
Time (s)