BIOE 198MI Biomedical Data Analysis. Spring Semester 2018.

Lab 1: Introduction.

Introduction to MATLAB: Matlab is a numerical computing environment featuring a programming language based on matrix mathematics. It includes a collection of toolboxes containing functions for numerical analysis, statistics, signal and image processing, data acquisition, symbolic math, machine learning, and other modeling and data analysis tools. We use versions R16A or B.

Your objective in this course is to become familiar with writing scripts in Matlab to solve general engineering problems and those in bioengineering. The method we adopt is to tell you a story that expresses a problem to be solved. That story can be expressed as a set of equations that you interpret into code that generates answers. Our goal is to lecture as little as possible while providing example problems that each of you work out in class. Coding skill is self-taught and a result of practice. We expect you to write lab reports that convincingly explain solutions from clearly visualized results. OK, let's go.

Help: Type `help fft` at the prompt. We will discuss options available there.
Also type `lookfor 'fourier'`. We will discuss the differences and advantages of each.

A. Scalars, vectors, matrices, and the associated syntax

```
>> a=2 %define scalar (a) at prompt (>>).  Percent (%) indicates comment
a =
     2
>> a = 2; %semicolon (;) suppresses the monitor output
>> v = [1;2;3;4;5;6] %6x1 column vector. Semicolon in data string is CR
v =
     1
     2
     3
     4
     5
     6
>> u = v' %u is the (conjugate) transpose of vector v.
u =
     1     2     3     4     5     6
>> vp = [1 2 3 4 5 6] %This shows vp = v', the same as vp = 1:6
vp =
     1     2     3     4     5     6
>> b=a*v; %scalar-vector product.  Note that b' = (a*v)' = a*v'
>> b=v'*v %b is redefined as a scalar given by inner product v'*v
b =
    91
>> B = v*v' %define B as 6x6 matrix given by the outer product v*v'
B =
     1     2     3     4     5     6
     2     4     6     8    10    12
     3     6     9    12    15    18
     4     8    12    16    20    24
     5    10    15    20    25    30
     6    12    18    24    30    36
```

```
>> % note that vector products do not commute
>> % type in 3x2 matrix C and 2x3 matrix D
>> C = [1 2;3 4;5 6] %spaces & semicolons indicated different actions
C =
     1     2
     3     4
     5     6

>> D = [1 3 5;2 4 6]
D =
     1     3     5
     2     4     6

>> E = C-D'
E =
     0     0
     0     0
     0     0

>> F = C'-D
F =
     0     0     0
     0     0     0

>> % Note that C = D' and C' = D but C-D' ≠ C'-D.  Also,
>> D = [1:3;4:6]  %another method for generating matrices
D =
     1     2     3
     4     5     6

>> E = C*D % matrix product exist only if C is NxM, D is MxP, and E is NxP
E =
     9    12    15
    19    26    33
    29    40    51

>> F = E/D %right divide D into E: F = E/D = C*D/D = C.  Note since
% C and D are rectangular matrices, inv(C) and inv(D) do not exist.
F =
    1.0000    2.0000
    3.0000    4.0000
    5.0000    6.0000

>> G = C\E % left divide of C into E: G = C\E = C\C*D = D
G =
    1.0000    2.0000    3.0000
    4.0000    5.0000    6.0000
```

Default number values in MATLAB are double precision float point where 5 significant figures are displayed. For example, the decimal value for 5/4 is 1.2500. Try it!

Class Assignment: Do the following both analytically and numerically.
type `clear all`

1. Let `A = [1 2;3 4];` Find `A^2, A.^2, sqrt(A), A*A', A'*A` and explain differences.

2. Find the inner and outer products of `a = [4;3;2;1];`

3. Let `b = [4 3 2 1];` Try to compute `a-b, a'-b` and `a-b'` Explain the differences.

4. Complex numbers (vectors in the complex plane), e.g., $z = x + iy$ and its conjugate $z^* = x - iy$.

Let `a=1+i; b=2-2*i;` Also let `c = (2*a').';` and `d = conj(b/2);` Explain why `a = d` and `b = c`

5. Take the square root of the product of a and b. The ambiguity is intentional!

6. Scalars are special cases of vectors just as vectors are special cases of matrices. Scalars have just one element. Yet the right and left divide operations still work. Before entering these into Matlab, predict the results of computing `8/2` and `8\2 and 2\8`. Hint, which number is in the denominator?

B. Scripts (m-files) and Data Display

If $y_n = f_n(x)$, find y_n from x for $n = 1, 2$. First, open an m-file using the editor, then type or copy and paste the following (being careful to note that some ASCII characters do not transfer accurately). Give your m-file a name by clicking SAVE AS.

```
%
clear all; close all;
x1 = 0:0.1:1; x2 = linspace(0,1,11); dx=x1-x2                    %(1)
y1 = x1.^2; y2=sin(2*pi*x2/x2(11)); plot(x1,y1,x2,y2)            %(2)
str1=['Maximum value in y1 ' num2str(max(y1))]; disp(str1);      %(3)
z=max(y1); fprintf('Maximum value in y1 is %1.2f\n' ,z)          %(4)
str2 = ['Maximum value in y2 is ' num2str(max(y2)) ' volts.']; ...
    disp(str2)                                                   %(5)
%
```

There are two equations that relate <u>independent</u> floating point variable x to a <u>dependent</u> floating point variable y. The integer subscript, denoted subscript variable n, differentiate the two equations.

1. The same x-axis was generated two ways. We saw they are the same within round off error.

2. We found $y_1=f_1(x_1)$ and $y_2=f_2(x_2)$ and plotted both 1-D functions. How do I know these are 1-D?

3. Find the maximum value of `y1`, convert the numerical value to a string for display using `num2str`, define string `str1`, and display the result using `disp`.

4. Display the same value using the `fprintf` command. Notice the different representation: `1.00` from `%1.2f/n` syntax, where `%` is a control character, `1.2f` prints a FP number with one digit to the left and two to the right, and `/n` imposes a CR because `fprintf` does not supply one.

5. Repeat for `y2` using `disp`. What do you expect for the maximum value? Why is the computed value less than your expected number? Hint, look at the plot. Note that **...** continues the function on the next line, which in this case is completely unnecessary! Do you see why?

C. Partitioning Scripts and Saving and Loading Data

```
%%                          %SECTION 1
clear all; close all;
a = [1 2 3 4 5];            %generate a row vector (1-D array)
dlmwrite('temp.txt',a,'\t'); %save a to disk as text file w/tab delimiters
whos                        %see what is in the workspace
%%                          %SECTION 2
b = load('temp.txt'); whos;  %read text file and convert to DPFP values
c = b.^2;                   %c is the square of each element of vector b
save('tempx.mat','b','c'); clear all;   %save b and c and clear workspace
load('tempx.mat'); b,c      %read the saved mat file and display b and c
%
```

How this partitioned m-file works: Script sections are divided by the double percent signs %%. Type or copy/paste the above script into an m-file using the editor. While in the editor tab, click on the top half of the script so it lights up. Then in the editor bar at the top, click on **run section**. Finally click the **advance** button and again on **run section**.

What this m-file does: The first section generates row vector a and writes the 1-D array into a txt file. Look for it in the **current folder** section. This type of write uses delimiters so the file can be read by other programs, like Excel. This delimiter is a tab as specified by `'/t'`. Use `','` to apply a comma delimiter. The second section loads the variables from a mat-file into a structure array. In this case, it is loading into memory data from an ASCII file into a double-precision floating point array. The data type allows mathematical operations on b to generate `c.` We then store those values as strings in a mat-file. You can choose to save the entire work space using `save('FILENAME.mat')`. Finally we clear the workspace and reload the b and c data arrays.

D. Keyboard Input, IF statements, and FOR loops

```
%% %SECTION 1
clear all; close all;
%Place an IF statement within a FOR loop.
for j=1:100      %display end of line text only when j=100
    if (j > 99)
        disp(['End of the line! ' num2str(j)]);
    end
end
%
%%  %SECTION 2
%Place a FOR loop within an IF statement.
%Request that user input an integer to calculate the factorial m!
m = input('Enter an integer: ');  %input value from keyboard.
fac = 1;                          %next line, m <= 21 to remain accurate
if (m >= 0 && m-floor(m) == 0 && m <= 21) %check three conditions
    for j=2:m                 %FOR loop computes m! if input qualifies
        fac = fac*j;
    end
    disp(['The factorial of ' num2str(m) ' is ' num2str(fac)]);
    disp(['Matlab function gives ' num2str(factorial(m))]);
else
    disp(['Cannot compute the factorial of ' num2str(m)]);
end
%
```

The <u>first section</u> of the script above looks at 100 numbers and tells you when it reaches the last number, i.e., 100.
The <u>second section</u> computes the factorial of a positive integer within the range of accuracy for double precision floating point numbers (15 digits).

E. Filling a 2-D Array with Different 1-D Arrays

```
%
clear all; close all;
x = -10:0.01:10; X0 = length(x); X2 = ceil(X0/2);Y = zeros(X0);%parameters
                                  %Look for these variables in Workspace
for j = 1:X0
    m = x(j);     % slope for each row starts negative and increases
    Y(:,j) = m*x; % zero intercept on linear function
end
imagesc(Y); colormap gray; axis square %create grayscale image of result
figure; subplot(1,3,1);plot(Y(1,1:2000));     %first row or Y
subplot(1,3,2);plot(Y(X2,1:2000))            %second row of Y
subplot(1,3,3);plot(Y(X0,1:2000))            %third row of Y
%
```

How can I tell a 1-D variable from 2-D variable? This section applies a FOR loop to fill each column of matrix Y with a linear function `m*x` where slope $-10 \le m \le 10$ steps from negative to positive with each column. Note that `Y(:,j)` addresses all column elements at the jth column given by the row counter in the second position of the 2-D array. We also introduce the `subplot` for making arrays of plots. In this case, we make a row vector of three plots. The third element in the argument addresses each plot element. <u>Use second line</u> in the code above to define frequently used variables, so when parameters change only one line changes.

F. Creating Functions in 1-D and 2-D. Visualization Options with Clear Labels and Titles

1. Start by generating and plotting a simple cosine wave, $A \cos(2\pi u_0 x)$.

2. Modulate the amplitude of the cosine by multiplying by a Gaussian function,
$$A \cos(2\pi u_0 x) \exp(-(x-a)^2/(2\sigma^2)), \qquad \text{(1-D Gabor pulse).}$$

3. Generate and visualize a 2-D Gabor pulse using
$$A \cos(2\pi u_0 x) \exp(-[(x-a)^2/(2\sigma^2)+ (y-b)^2/(2(2\sigma)^2)]), \quad \text{(2-D Gabor pulse).}$$

This 2-D function is a little harder to visualize, so we generate a grayscale image of the resulting array in 2-D using `imagesc` and a 3-D plot using `mesh`. The mesh plot is rotated using a mouse.

```
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Programming Lab 1   mfi 8/6/2017 (Signals)%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
clear all; close all;
%
%% SECTION 1. Create and plot a cosine wave
dx=0.01;X0=10;u0=1;x=0:dx:X0;A=2; %set parameters, generate 10mm x axis
f=A*cos(2*pi*u0*x);              %generate f(x)
subplot(2,1,1);plot(x,f,'k','linewidth',2)  %2x1 plot matrix, first element
title('Cosine wave: f(x) = Acos(2\piu_0x)');
xlabel('position x [mm]');ylabel('f(x)') %label axes
%
```

```
%% SECTION 2.  Create and plot a Gabor function
s1=0.8;                            %set pulse-width parameter
g=exp(-(x'-X0/2).^2/(2*s1^2)).*f'; %amplitude modulate the cosine wave
subplot(2,1,2);plot(x,g,'k','linewidth',2)
title('1-D Gabor pulse: g(x) = f(x) exp(-(x-x_0)^2/2\sigma^2)');
xlabel('position x [mm]');ylabel('g(x)')
%
%% SEACTION 3.  2-D Gabor pulse with visualization options
N=length(x);h=zeros(N);s2=2*s1;
for j=1:N   %form 2D pulse from 1D function y
    h(:,j)=g*exp(-(x(j)-X0/2)^2/(2*s2^2)); %note everything after g is scalar
end  %Note how x works for both spatial axes if square pixels
figure;subplot(1,2,1);imagesc(h);colormap gray;axis square
title('2-D Gabor pulse')
subplot(1,2,2);mesh(x,x,h)%click on '3D rotate' button on window to rotate
title('2-D Gabor pulse')
%
```

We can adjust the plot options using the example below. Here the cosine wave is replotted using blue dotted lines. I also increase the font size. Plots below this one used default sizes, which might not always be clear.

```
%
%% SECTION 4. Adjusting plot axis properties in script
%
figure;plot(x,f,'--','linewidth',2)
title('Cosine wave');xlabel('position x [mm]');ylabel('f(x)')
ax = gca; % current axes
ax.FontSize = 20;
%
```

The function below has the name `myplot`. It is a user-defined function that we write in a separate m-file that is located in the same folder as the script that calls `myplot`. Use this function to repeatedly send new data to a plotting routine having the same axis labels and other plot options. Note this function has input that changes each time it is called but no output except the plot.

```
%
%% A function to plot data using the same labels and other options but different data.
myplot(x,g)
%
%
%In a separate m-file labeled myplot we have the following…
function myplot(x,y) %place this function in the same folder
%
figure; plot(x,y,'r','linewidth',2) %use figure to avoid over-writing last plot
title('Gabor pulse via function myplot');xlabel('position [mm]');ylabel('g(x)')
ax = gca; % current axes
ax.FontSize = 20;
end
%
```

Lab 1 Concepts:

- Scalars, vectors, matrices, transpose operations, and matrix arithmetic. Numbers versus strings.
- Writing m-file scripts and displaying numerical results. `linspace, fprintf, disp, num2str`
- Methods for writing data to a file and reading the files (txt and m-files) `dlmwrite, save, load.` We also describe sectional scripts defined using %%.
- Keyboard input to an m-file, simple IF statements and FOR loops. Logical operators such as && and rational operators such as >= and ==. Type: `help &&` to see the list.
- Using a FOR loop to fill a 2-D array/matrix with 1-D arrays/vectors.
- Generation and visualization of 1-D and 2-D functions. `figure, plot, subplot, imagesc, mesh` and associated display functions mostly to adjust labels and fonts.
- Generating a new `function` called from a script.

Assignment:

A far-field light pattern from a rectangular aperture may be described using sinc functions. Let's simulate and display these 2-D light patterns to gain some intuition about their appearance.

First note that sinc is shorthand for, $\text{sinc}(x) = \sin(\pi x)/\pi x$. Plot the following 2-D light-field pattern,

$$A\,\text{sinc}\left(\frac{wx}{\lambda z}\right)\text{sinc}\left(\frac{hy}{\lambda z}\right) \quad .$$

Let A = 10V (peak voltage on recording device), w = 50 micrometers (μm, width of aperture), h = 35 μm (height of aperture), λ = 633 nanometers (nm, wavelength of light), and z = 0.2 mm (axial distance from the aperture). Select a range in the x,y plane and sample it appropriately. In a 1x2 subplot frame, display the result as a 2-D image in the left frame and give it a title but no axis labels. In the right frame, display the result as a 3-D mesh plot, giving it the same title and label the axes with units.

Suggestion: It is helpful to either set all parameters near the beginning of the script or to input the parameters from the keyboard. These enable your search.

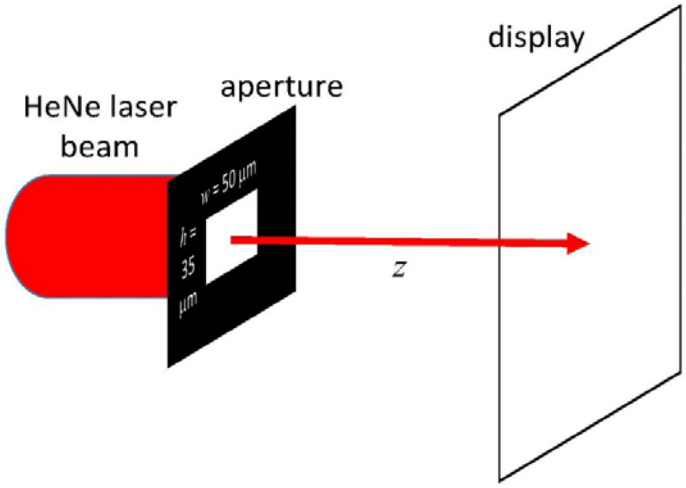

Figure 1. Illustration of the homework assignment.

Discussion Questions:

1. What length did you select for your x and y dimensions? Why?

2. What sampling frequency (dx, dy) did you select for your x and y dimensions? Why?

3. Explain your indices on x, y, h and how they correspond to units.