ECE 2713
Homework 3

Spring 2024 Dr. Havlicek

DUE: 03/07/2024, 11:59 PM

Introduction:

You will need to use Matlab to complete this assignment. So the first thing you need to do is figure out how you are going to do that.

The best option is to install Matlab on your own computer. There is a handout on the course web site that explains how to do this. As explained in the handout, for this option to work from off campus you have to start the OU VPN client *before* you start Matlab.

The second option is to buy the student version of Matlab. It costs $99. If you want to do this, you can find a link for it by googling for "purchase matlab student version." In general, the student version will keep on working until you are not a student any more. At times, you may have to provide *The Mathworks* with documentation to prove that you are still a student.

The third option is to run Matlab on the OU CoE Virtual Labs. You can use the OU VPN to do this from home. It sometimes runs *really* slowly! For this reason, I would consider this option mainly as a backup in case your first choice fails on the night that something is due. There is an excellent handout *How to Use Matlab on the Virtual Labs* available on the course web site that explains how to make this option work; it was written by Dr. Chad Davis.

Finally, there is a fourth option: Matlab is available through the Virtual Labs on all of the computers located in the CoE computer labs. Since this option requires you to be physically sitting in one of the on-campus labs to do any work, you will probably want to consider it only as a "last ditch effort" in case all of the other options fail you on the night that something is due.

Matlab is an interpreted computing environment and programming language. This means that there is a command prompt and you can type Matlab statements directly at the prompt. They get compiled and executed as soon as you hit the `Enter` key. Because it is an interpreted language (instead of a *compiled* one), you are free to make new variables, change the sizes of arrays, and define new data structures as you go along. One of the biggest advantages of Matlab is that it provides lots of powerful built-in functions that you can call to do complicated things like plot functions, read and write multimedia files, or even design sophisticated filter banks.

Instead of typing your Matlab statements at the command prompt, you can also put them in a file and then run the file from the command line. This is called an *m*-file. The file extension should always be "`.m`," as in `program1.m`. Then, if you type `program1` at the command prompt, the Matlab statements in the file will be executed. This is especially

useful if you are solving a complicated problem that requires hundreds or thousands of Matlab statements. When you type the name of an *m*-file at the command prompt, Matlab looks for the file in the current directory. The current directory is displayed near the top of the main Matlab window. There is also a search path that you can configure to tell Matlab where to look for your *m*-files, but we will not need to do this for ECE 2713.

You can use any editor you want to make an *m*-file. It must be a plain ascii file. So if you use MS WORD as your editor, then you need to save the *m*-file as plain text. Matlab also provides a specialized editor for creating and editing *m*-files. To start a new *m*-file using the Matlab editor, click the "New Script" button on the HOME tab. By clicking the "Open" pulldown on the HOME tab, you can open an existing *m*-file for editing in the Matlab editor, including an *m*-file that you created using some other editor. One of the nice things about the Matlab editor is that it has a built-in debugger that lets you do things like set breakpoints, single step your *m*-file, and examine the values of variables.

Matlab provides powerful syntax that can be used at the command prompt or in *m*-files to define and call functions and define abstract data types and objects. But we will not need much of that for ECE 2713.

Matlab also provides help for all of the statements and built-in functions. For help on the special variable $j = \sqrt{-1}$, type `help j` at the command prompt. For help on the built-in cosine function, type `help cos` at the command prompt. As you work this assignment, you should read the help for all of the functions that are being called in the problem you are working on.

You should go ahead and start Matlab now.

Note: If you are on the Virtual Labs, click the "Start" menu and type `matlab` in the search box – this should start Matlab (or at least put an icon for it on your desktop). This will work on your laptop or home computer too, but if you installed Matlab properly then you should already have a desktop icon.

The default view in Matlab will have a command window and several panes such as "Current Folder" and "Workspace." I usually get rid of the "Workspace" pane and add a docked "Command History" pane. This can be done by clicking the "Layout" button on the HOME tab. The reason I like to have a "Command History" pane is because you can click previous commands there and drag them to the command window. Then you can execute them again. You can also edit them before you execute them. This can save a lot of typing.

Preliminaries:

Before you get started on the assignment, let's go over some Matlab basics. To make a comment in Matlab, you type a percent sign (%). You can put comments in your *m*-files and you can also type comments at the command prompt. Matlab provides a command `whos` that lists your variables along with their data types. The default type for variables in Matlab is double precision matrix. Scalars like 5 will be stored in a 1×1 matrix. Try typing the following lines at the command prompt:

```
% Make a variable x and set it equal to 5
x = 5;          % everything after this "%" sign is a comment
whos
```

If you end a Matlab statement with a semicolon (;) like we did above, then nothing gets printed when the statement executes. If you leave off the semicolon, then the result of the statement will be printed to the command window. Try typing in this Matlab code:

```
y = 5
z = 6;
% I wonder what z is equal to?
z
```

To clear your workspace and start over fresh, use the `clear` command. Type in the following Matlab code:

```
whos
clear
whos
z
```

The last statement above makes an error because there is no longer any variable named `z`. The `clear` command destroyed it.

Row vectors are stored as $1 \times N$ matrices and column vectors are stored as $N \times 1$ matrices. The semicolon is used to end a matrix row. To transpose a vector or matrix, use a single quote. Try this Matlab code:

```
x = [1 2 3]
y = [4; 5; 6]
z = y'
```

Addition and subtraction in Matlab work the way you think they should. The two things that you are adding must have the same size. By default, multiplication in Matlab is matrix multiplication and the two matrices must be conformable. That means that the number of columns in the first matrix (or vector or scalar) must be equal to the number of rows in the second one. Try typing these lines:

```
a = x * y
b = x * z
```

The last line made an error because `x` and `z` are not conformable. There is also a `.*` operator that performs "pointwise" or "element-by-element" multiplication. This is known as the *Hadamard product* or *Schur product*. There is a good explanation of it on Wikipedia. Type these Matlab statements:

```
b = x .* z
c = x + z
```

Use three periods (...) to continue a Matlab statement across multiple lines like this:

```
d = ...
  b - z
```

Type in the following Matlab code that illustrates how to make matrices, multiply them pointwise, and perform matrix multiplication (**note:** Matlab variable names are case sensitive):

```
A = [1 2; 3 4];
A
B = [4 5; 6 7];
whos
C = A * B
D = A .* B
```

The entries of a Matlab scalar, vector, or matrix variable are allowed to be complex. By default, both `i` and `j` can be used for the imaginary unit $\sqrt{-1}$. You can also redefine `i` and `j` to be variables. Make sure that you don't redefine them both! For ECE 2713, you should use `j` for the imaginary unit. Try this Matlab code:

```
z1 = 1 + 2*j;
z2 = 3 + 4*j;
whos
z3 = z1 + z2
z4 = z1 * z2
z5 = conj(z4) % complex conjugate
real(z4)    % real part of z4
imag(z4)    % imaginary part of z4
abs(z4)     % magnitude of z4
angle(z4)   % angle of z4 (in rad)
```

When you call functions like `conj` and `real` on a matrix or vector, they are applied to all of the elements individually. The syntax to address individual elements of a Matlab array (vector or matrix) looks like this: `b(4)`, `A(2,3)`. For matrices, the first index is the row and the second index is the column. Matlab array indices start at one (this is different from languages like C where the array indices start at zero). You can also have loops in Matlab. Type in this Matlab code:

```
i = 2;
b(i)
b(2)
```

```
for row=1:2
  for col=1:2
    A(row,col) = A(row,col)*2;
  end
end
A
```

Notice that the loops don't execute until you have typed the last `end` statement.

There is a colon operator (:) that can be used to generate a range of integers like this:

```
n = -2:2
```

With two colon operators, you can generate an equally spaced vector of real numbers like this:

```
p = [-1:0.5:1]
```

When used as an array index, an expression like `2:3` extracts a range of array elements. When a colon is used by itself as an array index, it extracts all elements along one or more dimensions. Try these tricky Matlab statements:

```
G = [A B; B A];
whos G
G
G(1,:)
G(:,1)
G(2:3,1)
G(2:3,3:4)
G(:)
G(:)'
```

Now clear your workspace and let's get started with the assignment.

What to Turn In:

Submit your solution for this assignment electronically on Canvas by uploading a file to the "HW03" page.

To find the "HW03" page you can either scroll down on the "Home" page for this course or you can use the navigation links on the left side of the "Home" page to go to

```
ECE-2713-001 > Assignments > HW03
```

You can make your turn-in file with MS WORD or with any other editor that you prefer. Your turn-in file must be an MS WORD ".docx" file or a PDF file. To create PDF from MS WORD, print the file to PDF.

If you are using the Virtual Labs, make sure to save all your files before you log out!

As you work the problems, you can use the mouse to cut your Matlab code and resulting output from the command window and paste them into your turn-in file. You can also use the Matlab `diary` command to save a session log from the command window to a file like this:

```
diary my.txt
x = [1 2 3];
x(1)
diary off
```

This will save your command window session to a file called `my.txt` in the current directory. You can then open it with WordPad or WORD and paste it into your turn-in file.

As you create figures and graphs, they will show up in Matlab Figure Windows. You can use the File pulldown menu of any Figure Window to save the graph or figure in that window as a JPEG file or a BMP file. Then, you can insert the saved JPEG or BMP file into your turn-in file as a picture.

Note: To make the color work on the Virtual Labs, I had to open "Export Setup" from the File menu of the Matlab Figure window and **uncheck** the "custom color" box. If you are using the Virtual Labs, you may have to this too if you see that color is not working.

Make sure to include your name in your turn-in file and add a title at the top of the first page that says "ECE 2713" and "Homework 3." Number the problems and paste in your Matlab code and the resulting command window output. Paste in the figures and graphs and make sure to include answers for any discussion questions.

The Assignment:

1. In this problem, you will use Matlab to generate and plot the discrete-time unit impulse signal $\delta[n]$ and unit step function $u[n]$. Matlab provides two built-in functions that will be useful. The call `zeros(m,n)` returns an array of zeros with m rows and n columns. Similarly, `ones(m,n)` returns an $m \times n$ array of ones.

   Our first challenge is that Matlab array indexing starts at one; but for plotting $\delta[n]$ and $u[n]$ we will want to have the time variable n start at some negative integer. So we will have to use one array (let's call it `n`) to hold the values of n and another array to hold the values of the signal. In other words, the array `n` will hold the values from the *domain* of the signal (i.e., the "independent variable" values) and we will make a second array (`delta_n` for example) to hold the values of the signal from the *range*.

   Consider the following Matlab code which generates the signal $\delta[n]$ and plots it:

```
%-----------------------------------------------------
% P1a
%
% generate the signal \delta[n] and plot it
%
n = -10:10;          % values of the time variable
delta_n = [zeros(1,10) 1 zeros(1,10)];
stem(n,delta_n);
axis([-10 10 0 1.5]);
title('Discrete Unit Impulse Function');
xlabel('Time index n');
ylabel('\delta[n]');
```

(a) Type in the code and run it. You can type it in line-by-line at the command prompt or you can create an *m*-file (see page 2 above for how to create an *m*-file).

(b) Modify the code above to generate and plot $\delta[n-2]$ for $-10 \leq n \leq 10$. If it seems unclear how to do this, then remember that back on page 2 I recommended for you to read the Matlab help for every function that is being called as you work through this assignment. If you did that (i.e., if you read the Matlab help for all the functions that are called in the code above), then it should become clear to you how to modify the two calls to the `zeros` function in the code above so that it will generate $\delta[n-2]$ instead of $\delta[n]$.

(c) Use the Matlab functions `ones` and `zeros` to generate and plot the signal $u[n]$ for $-10 \leq n \leq 10$.

(d) Generate and plot $u[-n-3]$ for $-10 \leq n \leq 10$.

2. Consider the following Matlab code, which generates a discrete-time cosine signal $x[n]$ and plots it:

```
%-------------------------------------------------------------------
% P2a
%
% generate and plot a discrete-time cosine signal
%
n = 0:40;                                  % values of the time variable
w = 0.1*2*pi;                              % frequency of the sinusoid.
phi = 0;                                   % initial phase offset.
A = 1.5;                                   % amplitude.
xn = A * cos(w*n + phi);
stem(n,xn);
axis([0 40 -2 2]);
grid;
```

```
title('Discrete Time Sinusoid');
xlabel('Time index n');
ylabel('x[n]');
```

(a) Type in this code and run it.

(b) What is the length of the signal $x[n]$? In other words, for how many values of the time variable `n` does the signal assign a value? Hint: you can use the matlab `length` function to answer this question like this: `length(xn)`.

(c) What is the fundamental period of $x[n]$?

(d) What is the purpose of the `grid` command?

(e) Modify the code so that it generates a second sinusoid with length 50, frequency $0.4 \times 2\pi$ radians per sample, amplitude 2.5, and a phase offset of $-\pi/2$ radians. Run the modified code to generate and plot this second discrete-time sinusoid.

3. (a) Use Matlab to generate and plot the discrete-time signal

$$x[n] = \sin(\omega_0 n)$$

for the following values of ω_0:
$\frac{-29\pi}{8}$, $\frac{-3\pi}{8}$, $\frac{-\pi}{8}$, $\frac{\pi}{8}$, $\frac{3\pi}{8}$, $\frac{5\pi}{8}$, $\frac{7\pi}{8}$, $\frac{9\pi}{8}$, $\frac{13\pi}{8}$, $\frac{15\pi}{8}$, $\frac{33\pi}{8}$, and $\frac{21\pi}{8}$.

- Plot each signal for $0 \leq n \leq 63$.
- Label each graph with the frequency.
- Use the `subplot` function to plot four graphs per figure.
- Here is some code that will do all of this. You can type it in line-by-line at the command prompt or you can create an m-file.

```
%-------------------------------------------------------
% P3a
%
% plot a bunch of discrete-time sine signals
%

% The frequency will be w = k*pi/8.
% Load up the k's into a vector:
%
kvals = [-29 -3 -1 1 3 5 7 9 13 15 33 21];

% make a counter to index the "next" k to use:
next_k = 1;

n = 0:63;    % the time variable
```

```
% There are 12 k values.  We will plot four per
% figure.  So we will need three figures all
% together.  Loop on figures.

for Fig_num=1:3
  figure(Fig_num);   % selects the "current" figure

  % each time through this loop, we are going to do
  % 4 of the k's.  Loop on Sub Figure number:

  for SubFig_num = 1:4
    k = kvals(next_k);
    next_k = next_k + 1;
    w = k * pi/8;          % the frequency
    xn = sin(w*n);
    subplot(4,1,SubFig_num);
    stem(n,xn);
    title(sprintf('%d%s',k,'\pi/8'));
  end   % for SubFig_num
end   % for Fig_num
```

(b) Are any of the graphs from part (a) identical to one another? Explain.

(c) How are the graphs of $x[n] = \sin(\omega_0 n)$ for $\omega_0 = \frac{7\pi}{8}$ and $\omega_0 = \frac{9\pi}{8}$ related? Explain.

4. (a) Modify the code from Problem 3 to generate and plot the discrete-time signal

$$x[n] = \cos(\omega_0 n)$$

for the following values of ω_0:
$\frac{-29\pi}{8}$, $\frac{-3\pi}{8}$, $\frac{-\pi}{8}$, $\frac{\pi}{8}$, $\frac{3\pi}{8}$, $\frac{5\pi}{8}$, $\frac{7\pi}{8}$, $\frac{9\pi}{8}$, $\frac{13\pi}{8}$, $\frac{15\pi}{8}$, $\frac{33\pi}{8}$, and $\frac{21\pi}{8}$.

- Plot each signal for $0 \leq n \leq 63$.
- Label each graph with the frequency.
- Use the `subplot` function to plot four graphs per figure.

(b) Are any of the graphs from part (a) identical to one another? Explain.

5. (a) Use Matlab to generate and plot the discrete-time signal

$$x[n] = \cos(0.09n)$$

for $0 \leq n \leq 120$. For your plot, turn the grid on and scale the axes using the Matlab statements

```
axis([0 120 -1.0 1.0]);
grid;
```

(b) Is this signal periodic? Explain.

6. There are two main ways to represent continuous-time signals in Matlab. One way is using symbolic math, which we'll do later. For this assignment, we'll use the other way, which is: make a Matlab vector that actually contains *samples* of the signal, but where the samples are spaced so densely that it looks like a continuous-time signal when we plot it.

   Consider the Matlab code below which generates a continuous-time complex exponential signal and then graphs the real and imaginary parts in one figure and the magnitude and phase in another figure.

```
%-----------------------------------------------------------------------
% P6a
%
% generate and plot a continous-time complex sinusoid
%
t = -4:0.01:4;                              % values of the time variable
w = 2.2;                                    % frequency of the sinusoid.
xt = exp(j*w*t);
xtR = real(xt);
xtI = imag(xt);
figure(1);              % make Fig 1 active
plot(t,xtR,'-b');       % '-b' means 'solid blue line'
axis([-4 4 -1.0 2.0]);
grid;
hold on;                % add more curves to the same graph
plot(t,xtI,'-r');                    % 'r' = red
title('Real and Imaginary parts');
xlabel('Time t');
ylabel('x(t)');
legend('Re[x(t)]','Im[x(t)]');
hold off;

mag = abs(xt);
phase = angle(xt);
figure(2);              % make Fig 2 active
plot(t,mag,'-g');       % '-' = solid line; 'g' = green
grid;
hold on;                % add more curves to the graph
plot(t,phase,'-r');              % 'r' = red
title('Magnitude and Phase');
legend('|x(t)|','arg[x(t)]');
```

```
xlabel('Time t');
ylabel('x(t)');
hold off;
```

(a) Type in and run this code.

(b) Use similar Matlab statements to generate the continuous-time damped exponential signal

$$x(t) = 3e^{-t/2}e^{j8t}$$

for $0 \leq t \leq 4$. Plot the real part, imaginary part, magnitude, and phase.

Hint: for multiplying the two exponentials, you need to use the `.*` operator like this:

```
xt = 3.0*exp(-t/2).*exp(j*w*t);
```

If you try to just use `*` instead, you will get an error because `*` means *matrix* multiplication – and the "matrices" `3.0*exp(-t/2)` and `exp(j*w*t)` are both actually vectors, so they are not conformable and can't be multiplied as matrices (see notes pages 1.79 - 1.85 if you have forgotten what *conformable* means).