MATLAB Exercises

Exercise C1

[20pts] Find all of the *n*th roots of a complex number and draw them using MATLAB, as outlined below.

Consider a complex number $\mathbf{z} = a + \boldsymbol{j}b = Ae^{j\varphi}$.

(a) Derive an expression for ALL of the *n*th roots of **z**, i.e.,

$$\mathbf{z}^{1/n} = Re^{j\beta} = X + \boldsymbol{j}Y$$

where *n* is an integer. In other words, treating parameters *a*, *b*, *A*, and φ as known, derive expressions for *R*, β, *X*, and *Y* suitable to be used to draw the complex numbers. You will also need to derive an expression for $d\beta$, the phase difference between two adjacent *n*th roots of **z**.

(b) Use your expressions from part (a) to write the missing expressions in the MATLAB code on the next page. (You can also use the "Getting Started" M file on Moodle.)

(c) Develop a check to verify that <u>each</u> calculated value of *X* and *Y* satisfies $\mathbf{z} = (X + \boldsymbol{j}Y)^n$.

(d) Run your fully functioning script to determine the following:

1. $(3 + j4)^{1/5}$
2. $(2 - j5)^{1/4}$
3. $(-1)^{1/3}$
4. $(-1 - j0.0001)^{1/3}$
5. $(-j)^{1/9}$
6. $(5 + j15)^{1/7}$

(e) Explain why the principal cube root of –1 is not –1. HINT: Compute it by hand.

(f) Explain why the principal cube root in d3 is different from the principal cube root in d4. HINT: Compute it by hand, treat the amplitudes as equal, and highlight the major differences.

NOTES:

- In Electrical and Computer Engineering, we generally write complex numbers as $a + \boldsymbol{j}b$, but the input to MATLAB will need to be in the form a+bj. (You may also use the form a+bi, but I recommend that you start getting accustomed to "j" as the imaginary unit, as we generally use *i* as the symbol for current.) For instance, enter "3+4j", "2-5j", "-1", …; that is, no parentheses or * operators are needed.
- Add a few lines at the beginning of your script with your name, the date you started, the context (ECE 201 Winter 2022), and a very brief description of this assignment.
- The script is set up to work in degrees, so you should use the functions COSD and SIND to compute in degrees, not radians.
- The checks for part (c) will output automatically as a column array of very small complex numbers, once you have written suitable expressions.
- The answers to parts (e) and (f) should be added as comments to the end of your script. Use the exponential form of each complex number throughout your explanations.
- Print your final script to PDF, in color, with line numbers and a proper header. Print the Command Window also to PDF with a proper header. Suppress output for everything but the checks.
- Export each figure using "Save As…" PNG or JPG. In other words, you should not take any screen shots. You might need to adjust the size of the Figure window to fit the title. Adjust the window also to make sure the font sizes are appropriate. Combine the PDFs of your script, your hand calculation, and the Command Window with all six figures, e.g., using http://online2pdf.com, into one PDF.
- The older version of "Finishing B1" in the Lecture Links on OWL is very useful.

Exercise C1 (continued)

```
% Getting Started with C1 in ECE 201 / C9 in ECE 296C

clf   % clear all figures
clear   % remove all variables from the workspace

% needed to draw axes
ax = [-100,100]; ze = [0,0];

hold on
% plot x- and y-axes (draw the axes before using QUIVER)
plot(ax,ze, 'k', 'LineWidth', 1)
plot(ze,ax, 'k', 'LineWidth', 1)

z = input('Input a complex number z as a+bj: ');
n = input('Input the power of the root, i.e., n of z^1/n: ');  %
a = real(z); b = imag(z);

% z = a+jb = A*exp(j*phi)

A = norm(z);   % amplitude of z
phi = rad2deg(angle(z));   % phase of z in degrees, -180deg < phi <= 180deg

R = ***expression***;   % amplitude R of z^1/n
B = ***expression***;   % phase angle associated with principal value of z^1/n
dB = ***expression***;   % difference between phase angles, in degrees

X = ***expression***; Y = ***expression***;

% plot an arrow to represent the principal value of z^1/n, in red
quiver(0, 0, X, Y, 0, 'r', 'LineWidth', 3)

check = zeros(n,1);   % initialize n checks as a column vector

check(1) = ***expression for first check***;   % first check, output at the end

for i = 2:n   % cycle through the other values to make the rest of the arrows

    B = B + dB;   % add dB to find the next root of z
    X = ***expression**; Y = ***expression***;

    % plot an arrow for the next root, in blue
    quiver(0, 0, X, Y, 0, 'b', 'LineWidth', 3)

    check(i) = ***expression for check***;

end

check   % output the n checks; each should be close to 0+j0

% make the figure look nicer

grid on; axis equal;

ac = gca; ac.FontSize = 16; ac.GridAlpha = 0.5;   % change all fonts to 16pt; make the grid darker

xlabel('Re({\textbf{z}})', 'FontSize', 20, 'Interpreter', 'latex');
ylabel('Im({\textbf{z}})', 'FontSize', 20, 'Interpreter', 'latex')

% determine the sign of b, so that we can include z in the title
bSgn='+';
if b<0
    bSgn='-';
end
bMag = norm(b);   % magnitude of b

title({'ECE 201/296C Exercise C1/C9', ...
    sprintf('Finding and drawing the %d values of $(%d%sj%d)^{1/%d}$', ...
    n,a,bSgn,bMag,n)}, 'FontSize', 24, 'Interpreter', 'latex')

max = ceil(R+0.1);   % round up R to the next integer
axis([-max max -max max])   % set the upper and lower limits of the axes

hold off
```

Exercise L1

[20pts] Consider the following matrices **A**, **B**, and **C**, and use them in all parts below.

$$\mathbf{A}=\begin{pmatrix} 6 & -5 & -2 \\ 2 & -4 & 5 \\ 0 & 1 & -8 \end{pmatrix};\ \mathbf{B}=\begin{pmatrix} 2 & 3 & 5 \\ -1 & 1 & 6 \\ 8 & -5 & -3 \end{pmatrix};\ \mathbf{C}=\begin{pmatrix} -1 & -4 & 1 \\ 2 & 2 & 0 \\ -4 & 3 & 2 \end{pmatrix}$$

(a) What is **AB**? What is **BA**? Show that **AB** ≠ **BA**.

(b) Show that (**AB**)**C** = **A**(**BC**).

(c) Show that (**A** + **B**)**C** = **AC** + **BC**.

(d) Show that $(\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T$. Show that $(\mathbf{AB})^T \neq \mathbf{A}^T\mathbf{B}^T$.

(e) What is $\mathbf{A}^{-1}$? What is $\mathbf{B}^{-1}$? What is $(\mathbf{AB})^{-1}$?
Show that $(\mathbf{AB})^{-1} \neq \mathbf{A}^{-1}\,\mathbf{B}^{-1}$.
Show that $(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\,\mathbf{A}^{-1}$.

(f) What is det(**A**)? What is det(**B**)? What is det(**AB**)?
Show that det(**AB**) = det(**A**) det(**B**).

(g) Show that $\det(1.2\mathbf{A}) = (1.2)^3 \det(\mathbf{A})$.

NOTES:

- Write a MATLAB script to answer these questions. Be sure to include your name, a date, a context, and a description.
- Your script should be efficient. This means you should define variables that you are using more than once, e.g., the matrix **AB** should be defined, as it can be used several times.
- Note that the best way to "show that" something is true or not true is to construct an expression that is output to the Command Window. For instance, for part (b), rearrange the equation so that something is equal to zero, compute that expression, and an output of an array of 0s is sufficient to show that the left side is equal to the right side. DISP commands are not needed, especially if you choose meaningful variable names.
- If possible, write each check in one expression, i.e., you don't need to define something, then use it once. For example, once **AB** and **BC** are defined, you can write the check for parts (b) and (c) in one expression.
- On the other hand, if you need something multiple times, define it then use it. For example, in part (d), you will use $\mathbf{B}^T$, $\mathbf{A}^T$, and $(\mathbf{AB})^T$ twice each, so you should define each before writing the checks.
- Comments should be specific, telling the reader exactly what you expect the result of each check to be. For instance, if you expect a 3 x 3 array of 0s, that's what you should write as your comment.
- The output needs to be meaningful, either with meaningful variable names or using the DISP function. (Meaningful variable names are much more convenient, e.g., check_a for the first check.) Suppress output for any line of code that does not specifically answer one of the questions above. For instance, you do not need to output $(\mathbf{AB})^T$ in part (b).
- Combine the MATLAB script and output from the Command Window into a single PDF before you submit it to Moodle. Clear the Command Window before your last run.

Exercise L2

[30pts] Write a robust, efficient MATLAB script to find the eigenvalues and eigenvectors of any 2 x 2 matrix input by the user.

You should test out your script using the following matrices.

$$\mathbf{P} = \begin{pmatrix} 3 & 2 \\ 4 & 1 \end{pmatrix} \quad \mathbf{Q} = \begin{pmatrix} 3 & -1 \\ -2 & 4 \end{pmatrix} \quad \mathbf{R} = \begin{pmatrix} 1 & -2 \\ -4 & 8 \end{pmatrix} \quad \mathbf{S} = \begin{pmatrix} 2 & 3 \\ -3 & 8 \end{pmatrix}$$

You may not use any special MATLAB tools. Instead, work symbolically and derive general expressions for the eigenvalues and the eigenvectors. For instance, you may use TRACE and DET (determinant), but you may not use the SOLVE function. If you are not sure whether or not a particular function is available to you, just ask. Include your hand calculations when you submit your solution to be graded.

Assuming that each eigenvector can be written as the column vector [a; b], derive TWO expressions for the ratio a/b, one using the top row of the input matrix and one using its bottom row, compute their difference (which should be equal to 0), and output the result as a quick check that the eigenvalue is correct. Repeat for the second eigenvalue.

"Fix" each eigenvector by making the magnitude of each eigenvector equal to 1 (i.e., normalized), with the first element positive. Output the eigenvalues and "fixed" eigenvectors for each matrix.

Check your fixed eigenvectors by making sure that $\mathbf{M}\mathbf{x}_n = \lambda_n \mathbf{x}_n$, where $\mathbf{M}$ is the matrix you are inputting, $\mathbf{x}_n$ is the nth eigenvector, and λ_n is the nth eigenvalue. (Again, a proper check will return an array of zeros.)

The output should be meaningful, i.e., either using meaningful variable names, or using DISP.

Once you have a functioning script, answer the following questions. You might need to try other matrices before you see the pattern. You can also look at the equations you have derived for insight into what is controlling the eigenvalues.

(a) Under what conditions will both eigenvalues be positive?

(b) Under what conditions will one eigenvalue be positive and the other be negative?

(c) Under what conditions will one eigenvalue be equal to zero?

(d) Under what conditions will there be only one eigenvalue?

Write your answers to these questions as comments at the end of your script.

Combine your hand calculation, MATLAB script, and output from the Command Window into one PDF before you submit it to Moodle for grading.

NOTES:

- Use `DET` (determinant) and `TRACE` (sum of the diagonal elements) to make the script more efficient. These are also very useful for answering the four questions. Specifically, you should be able to write simple expressions purely in terms of the determinant d and trace t.
- Run your script with all four given matrices above.
- The hand calculation should be done symbolically, using variables for the elements of the given matrix. The hand calculation should end with expressions suitable to: (1) find the eigenvalues and then (2) find <u>two</u> expressions for the ratio a/b representing the elements of each eigenvector. The script should agree with your hand calculation.
- Choose short, meaningful variable names, such as e1 and e2 for the eigenvalues, x1 and x2 for the unnormalized eigenvectors, and n1 and n2 for the normalized eigenvectors.

Exercise L2 (continued)

Here is an example of one way to find and check eigenvalues and eigenvectors. It takes full advantage of the feature that ANY multiple of an eigenvector is also an eigenvector. However, this also means it will be impossible to derive an exact equation for the elements of the eigenvector. The best you can ever do is find the ratio of the elements of the eigenvector.

You will need to do this symbolically, following the same steps. I will do it for a specific matrix **M**:

$$\mathbf{M} = \begin{pmatrix} 1 & 6 \\ 8 & -1 \end{pmatrix}$$

One eigenvalue is –7, which means **Mx** = –7**x**. Letting **x** be the column vector [a; b]…

$$\begin{pmatrix} 1 & 6 \\ 8 & -1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = -7 \cdot \begin{pmatrix} a \\ b \end{pmatrix}$$

Again, I cannot solve for a and b independently, so I am looking for a relationship between a and b. Using the top row of **M**, the equation becomes:

$$1a + 6b = -7a$$

or a/b = 6/(–8) = –3/4. Therefore, <u>every</u> vector for which the ratio a/b is equal to –3/4 will be an eigenvector of **M** with an eigenvalue equal to –7. As a quick check, use the bottom row of **M** to derive a second equation:

$$8a + (-1)b = -7b$$

or a/b = (–6)/8 = –3/4, which, of course, is the same ratio found using the top row of **M**. In other words, even though it might seem like we have two different equations to solve for two unknowns a and b, in fact, these two equations are identical in their mathematical content, because they are not "independent" in the formal sense. Therefore, we always have two ways of solving for one eigenvector. If these two ways do not give us the same relationship, then the eigenvalue is wrong. Therefore, the first and quickest check is to compare a/b calculated two different ways.

With a/b = –3/4, we can write **x** using integers:

$$\mathbf{x} = \begin{pmatrix} -3 \\ 4 \end{pmatrix}$$

The magnitude of **x** is 5, so the first normalized eigenvector (with a positive first element) is:

$$\mathbf{n} = \begin{pmatrix} 3/5 \\ -4/5 \end{pmatrix} = \begin{pmatrix} 0.6 \\ -0.8 \end{pmatrix}$$

As a second check, insert the fixed eigenvector into the original eigenvector equation, and construct an expression that should be an array of zeros:

$$\mathbf{Mn} - \lambda\mathbf{n} = \begin{pmatrix} 1 & 6 \\ 8 & -1 \end{pmatrix} \begin{pmatrix} 0.6 \\ -0.8 \end{pmatrix} - (-7) \begin{pmatrix} 0.6 \\ -0.8 \end{pmatrix} = \begin{pmatrix} 1(0.6) + 6(-0.8) = -4.2 \\ 8(0.6) + (-1)(-0.8) = 5.6 \end{pmatrix} - \begin{pmatrix} -4.2 \\ 5.6 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Therefore <u>any</u> vector proportional to **n** = [0.6; –0.8] is an eigenvector of **M**, with an eigenvalue of –7.

Exercise L3

[20pts] This is a continuation of the previous exercise. It is strongly recommended to wait until you are finished with exercise L2 before attempting this exercise.

First determine the conditions for which a matrix is diagonalizable. If a matrix is not diagonalizable, output a comment to say so. If a matrix is diagonalizable, find the matrix **X** that will transform the input matrix **M** into the diagonal matrix **D**. You may use the normalized or unnormalized eigenvectors to construct **X**, and note that this can be done very efficiently in MATLAB.

Run your script for matrices **P**, **Q**, **R**, and **S** from L2.

Output the eigenvalues for each input matrix. If **M** is diagonalizable, compute then output matrix **X** and matrix **D**. (Make sure to compute **D** using **X** and the input matrix **M**, so that you can see that the result is, in fact, a diagonal matrix with eigenvalues along the diagonal.)

What can you say about the determinant of the matrix input as compared to the determinant of its diagonalized version? Explain why this is true, i.e., prove it, step by step, in a comment at the end of your script. (One of the results from L1 should help make the explanation particularly efficient.)

Diagonalizing a matrix

We can use the eigevectors to construct a matrix **X** that will diagonalize **M**. Specifically, if the eigenvectors become the columns of **X**, then we can show that:

$$\mathbf{D} = \mathbf{X}^{-1}\mathbf{M}\mathbf{X}$$

where **D** is a diagonal matrix with the eigenvalues along the diagonal. Thus **X** must be invertible for a matrix to be diagonalizable. Note that the eigenvectors can be normalized or unnormalized.

Continuing the example from L2, consider the matrix **M**:

$$\mathbf{M} = \begin{pmatrix} 1 & 6 \\ 8 & -1 \end{pmatrix}$$

M has two eigenvalues, $\lambda_1 = 7$ and $\lambda_2 = -7$. The unnormalized eigenvectors are:

$$\mathbf{x}_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad \mathbf{x}_2 = \begin{pmatrix} -3 \\ 4 \end{pmatrix}$$

which means $\mathbf{M}\mathbf{x}_1 = 7\mathbf{x}_1$ and $\mathbf{M}\mathbf{x}_2 = -7\mathbf{x}_2$. The diagonalization matrix **X** and its inverse are:

$$\mathbf{X} = \begin{pmatrix} 1 & -3 \\ 1 & 4 \end{pmatrix} \quad \mathbf{X}^{-1} = \begin{pmatrix} 4/7 & 3/7 \\ -1/7 & 1/7 \end{pmatrix}$$

To see why this transformation works, it is useful to evaluate **MX** first:

$$\mathbf{D} = \mathbf{X}^{-1}\mathbf{M}\mathbf{X} = \begin{pmatrix} 4/7 & 3/7 \\ -1/7 & 1/7 \end{pmatrix}\begin{pmatrix} 1 & 6 \\ 8 & -1 \end{pmatrix}\begin{pmatrix} 1 & -3 \\ 1 & 4 \end{pmatrix} = \begin{pmatrix} 4/7 & 3/7 \\ -1/7 & 1/7 \end{pmatrix}\begin{pmatrix} 7 & 21 \\ 7 & -28 \end{pmatrix} = \begin{pmatrix} 7 & 0 \\ 0 & -7 \end{pmatrix}$$

In other words, **MX** consists of $7\mathbf{x}_1$ in the first column and $-7\mathbf{x}_2$ in the second column. Further, because we are guaranteed that $\mathbf{X}^{-1}\mathbf{X} = \mathbf{I}$, the first row of $\mathbf{X}^{-1}$ multiplied by the first column of **MX** must be equal to $7(1)$, and the first row of $\mathbf{X}^{-1}$ multiplied by the second column of **MX** must be equal to $-7(0)$. Likewise, the second row of $\mathbf{X}^{-1}$ multiplied by the first column of **MX** must be equal to $7(0)$, and the second row of $\mathbf{X}^{-1}$ multiplied by the second column of **MX** must be equal to $-7(1)$.

Thus, assuming that the diagonalization matrix **X** has an inverse, we can construct the matrix $\mathbf{D} = \mathbf{X}^{-1}\mathbf{M}\mathbf{X}$ which will be diagonal with the eigenvalues along the diagonal.

Exercise D1

[20pts] Let's imagine that we have a first-order differential equation that is hard or impossible to solve. The general form is:

$$\frac{df}{dt} + g(t) \cdot f(t) = h(t)$$

where $g(t)$ and $h(t)$ are understood to be known.

It turns out that any first-order differential equation is relatively easy to solve using computational techniques. Specifically, starting from the definition of the derivative…

$$\frac{df}{dt} = \frac{f(t+dt) - f(t)}{dt} \qquad (dt \text{ small})$$

… we can rearrange the equation to become…

$$f(t+dt) = f(t) + dt \cdot \frac{df}{dt} \qquad (dt \text{ small})$$

In other words, if you know the function and its derivative at any time t, and if you choose a time interval dt that is small enough, then you can find the value of the function at the next instant of time $(t + dt)$.

In this case, because $g(t)$ and $h(t)$ are known, we can easily rearrange the differential equation to be in the form, df/dt = [functions of time]. Thus, the derivative is known at any time t, as long as $f(t)$ is known at time t.

Therefore, set up a robust, efficient script to numerically solve a differential equation of this type. For now, we will use $g = 5$, $h = -6$, and $f(0) = 3$. You will need a FOR loop.

Plot the resulting $f(t)$ from $t = 0$ until $t = 1$, using at least 1000 points. Add appropriate axis labels and a meaningful title, and adjust the line thickness and font sizes. Include the differential equation in the title or in a TEXT command. Refer to the numerical solution in the title and in the legend text.

As a check, solve the differential equation analytically, and compare it to your numerical solution. This should be done robustly, i.e., the parameters of the analytic solution should depend on what's given (g, h, and $f(0)$). Then, construct a function that you believe should be an array of zeros, and plot it on the same axes as the numerical solution. Make the legend meaningful, e.g., it could say "check (should be zero)".

Exercise D2

[20pts] This is a continuation of the previous exercise. It is strongly recommended to finish D1 before attempting this exercise.

In this case…

$$\frac{df}{dt} + At^n \cdot f(t) = B \cdot \cos(\omega t)$$

with $A = 5$, $n = 2$, $B = -6$, $\omega = 12$, and $f(0) = 3$. Add user inputs for all of these quantities.

As in D1, derive an expression for *df/dt* as a function of known variables and functions, then construct $f(t)$ as before. Make the ending time a user input as well, and plot the resulting function again from $t = 0$ until $t = 1$, then run it again and plot the function from $t = 0$ to $t = 8$.

Make sure the title is meaningful and robust, i.e., the title (or a subtitle) should include the differential equation being used and, as A, n, B, and ω change, the title should change. (You might instead choose to add a `TEXT` box within the frame of the graph with the differential equation.) LaTeX is recommended.

As a check, input $n = 0$ and $\omega = 0$ in order to recover the previous results, which you can again compare to the analytic solution. This is a useful check on your script. NOTE: You should plot the checking function ONLY when both $n = 0$ and $\omega = 0$. And you only need a legend when there are two or more plots on the chart. Think about how to do this most efficiently and robustly.

Exercise D3

[30pts] Consider the forced, second-order, linear differential equation with constant coefficients…

$$\frac{d^2f}{dt^2} + g \cdot \frac{df}{dt} + h \cdot f(t) = a \cdot \cos(\omega t) + b \cdot \sin(\omega t)$$

where t is measured in seconds, so ω is in rad/s.

Make g, h, a, b, and ω user inputs.

Determine the steady-state response (particular solution). Do this by deriving expressions for the most general solution, setting up a matrix equation yourself, and solving it. In other words, the particular solution is a sinusoid with two unknown coefficients, so you will need two equations to solve for them. This is a great opportunity to apply linear algebra to help solve a differential equation.

Work symbolically using the parameters above, so that your script is robust. Do not use any specialized functions in MATLAB. (You also should not attempt to solve for the two unknown coefficients yourself. Let MATLAB do it!)

Output the values of the coefficients of the particular solution. Make sure the comments in your script are useful and sufficient for interpreting the output.

Run the script with the following values…

g	h	a	b	ω
10	16	3	4	5
10	25	3	4	20
10	25	3	4	200
10	25	3	4	20,000
2	3	4	5	0.1
3	1.44	3	4	2

OPTIONAL: Rather than running the script over and over again, you might want to put everything inside a WHILE loop, with the question, "Would you like to run the script again?" at the end (with appropriate instructions for what the user should enter to continue or stop). You should put the CLEAR and CLF commands inside the WHILE loop.

Exercise D4

[20pts] This is a continuation of the previous exercise.

For each set of input parameters, plot the forcing function and the particular solution on the same chart, with an appropriate title.

The tricky part here is that the scale of the response (particular solution $f(t)$) can be very different than the forcing function, so one might easily dwarf the other. One way to fix this is to scale the response by some factor of 10. To find the appropriate factor of 10, use this line of code:

```
nscale = round(log10(ratio));
```

where "ratio" is the ratio of the amplitude (maximum value) of the forcing function to the amplitude of the response. Then, multiply the response by a factor of `10^nscale` when you plot it.

Include this information in the legend text, e.g., if nscale = 2, then the corresponding text might be "response (x10^2)", where the "2" is robust. And if there is no scaling, leave it out of the legend, i.e., the legend text should be simply "response".

Also, choose the axis limits robustly, adding a little space vertically (e.g., usually about 30% of the amplitude of the largest sinusoid) and showing about 1½ cycles of the sinusoids starting at $t = 0$.

Once you have chosen an upper limit in time, adjust the time scale so that this upper limit is an integer between 1 and 1000. For example, if the limits of the time axis are between 0s and 0.5s, then plot in ms, so that the limits are instead between 0ms and 500ms. This should be done dynamically. Don't neglect the units in the X label. You may use "u" to represent μ in μs. Typically, you will use s, ms, or μs. Specifically, if "tmax" is less than 1s, use ms, and define a suitable time array in ms that you can use for plotting. If "tmax" is also less than 0.001s, use μs, and define a suitable time array in μs instead.

Finally, add the differential equation in LaTeX as part of the title or using the TEXT command, with (known) input values of all the parameters. Note that you can render only part of the title as an equation by placing $ around the equation. Let me know if this is not working for you. If you use a TEXT command, make sure there is enough space for it.

Submit your M file and 6 plots (as one PDF). No Command Window is needed.