Laboratory Assignment 1
MATLAB Introduction

In this lab, you will be introduced to MATLAB by using it to evaluate expressions containing complex numbers. Using MATLAB, you can easily express these complex numbers in either rectangular or polar form. MATLAB's plotting capabilities will be introduced and used extensively. You will also use M-files and create some simple signal processing functions that are used in later laboratory assignments.

1.1 OBJECTIVES

By the end of this laboratory assignment, you should be able to:

1. Use MATLAB to perform complex arithmetic.
2. Generate and plot signals and complex valued functions.
3. Confidently develop MATLAB M-files and save results of computations from a MATLAB session.

1.2 REFERENCE

Review Topics:

1. Algebra of complex numbers
2. Sketching of discrete and continuous time signals
3. Vector and matrix algebra

1.3 INTRODUCTION

As you work through the problems and experiments in this course, you will be using MATLAB - a powerful computing environment for numeric computation and visualization - quite frequently. MATLAB is designed for ease of use and behaves as a high-level programming language that is tailored for signal processing, communication, and control tasks. It is used by professionals in industry and academia worldwide in research, development, and design. An open source program having very similar capabilities and commands is also available called Octave.

MATLAB, short for MATrix LABoratory, works on matrices of numbers. We focus mostly on one-dimensional matrices called vectors that contain signal samples, or on multiple-dimensional matrices containing several signals or the parameters of a system. For example, a vector could contain just a list of values from a mathematical function that

you wish to plot. We will first focus on familiarizing you with the matrix notation in MATLAB, and get you used to working with vectors and matrices in arithmetic operations.

This introduction has three parts: reading, reading while doing, and doing guided by knowing expected results. Since the main difficulties in learning MATLAB are in learning the syntax and (in some cases) learning to program, we hope that this format will remove some of the trauma. Solutions to the questions in the "Quiz" section should be handed in as part of your Lab 1 report.

This introduction is not intended to present you with everything you need to know about MATLAB; it is merely to bring you to a point where you can do the following labs in the course. Use the on-line help, and references for additional information.

You should follow this text with MATLAB running, and work through the examples and questions. MATLAB is available on many other platforms. MATLAB runs under Microsoft Windows, Linux X Windows, or Mac OS.

All versions of MATLAB are compatible in file storage format and M-file format, so data stored on one system can be transferred to another without loss. Each MATLAB session has at least two windows: a text window, where commands are typed and data is displayed, and a graphics window, where graphics appear.

Octave has a virtually identical user interface to MATLAB. If you plan to do work at home on labs and assignments and do not have a license of MATLAB, it is recommended that you repeat some or all of the activities below using Octave on your home computer.

1.4 Tutorial, Part I

The following sections highlight some useful MATLAB commands by working through some example problems. You should work along with the text.

1.4.1 Start MATLAB

Find the MATLAB icon on the desktop and double click. Two windows will open.

MATLAB has on-line help for all functions and a set of demos; we recommend that you try the demos so you can see some of MATLAB's computing power and some of the functions available to you.

When the text window opens, a prompt appears:

```
>>
```

All commands will be entered after a prompt like this one.

1.4.2 Evaluating Complex Variables and Expressions

In the following text, information that you enter will be preceded by the MATLAB prompt; information not preceded by a prompt is printed by MATLAB as a computation result or other information.

Problem 1: Express each of the following complex numbers in Cartesian form, i.e., $s = a+jb$, where $a = \mathrm{Re}\{s\}$ and $b = \mathrm{Im}\{s\}$. Plot part (a) in the complex plane.

a. $je^{j11\pi/4}$

b. $(1-j)^{10}$

Part a: You can find the Cartesian form by typing the expression, using standard symbols for arithmetic operations.

```
>>    j*exp(j*11*pi/4)

ans = -0.7071- 0.7071i
```

Note that MATLAB has evaluated the expression and echoed the result to the screen, expressed in Cartesian form as the variable ans. Also, just like any programming language, exp (x) returns e^x. Other standard functions, including trigonometric functions, are available; type **help elfun** for a list. Additionally, **pi** is defined as a special variable having the value π, and **j** is defined as $\sqrt{-1}$. Any special variable will act as defined until you change its value by assigning a new value to it. For example, to change **pi** to 3, issue the command:

```
>> pi = 3

pi = 3

>> pi
pi = 3
```

As you can see, pi has been changed. If you clear your definition, the old one reappears:

```
>> clear pi
>> pi

ans =
```

```
3.1416
```

Other variables can be set in the same way:

```
>>    z = 3+4*j
z=
3.0000      + 4.0000i
```

So, to set **z** to the solution of part b:

```
>> z = (1-j)^10
z=
0.0000      -32.0000i
```

1.4.3 Plotting Complex-Valued Functions

Plots in MATLAB are generated using the plot function.

plot (x, y) generates a plot where the values of the vector **x** indicate points along the horizontal axis corresponding to the values in the vector **y** that are to be plotted on the vertical axis. Vectors **x** and **y** must have the same number of elements.

Since complex values have two components corresponding to **a+jb**, MATLAB provides the **real** and **imag** functions to separate the real and imaginary parts of an imaginary number:

```
>>    z = 3+4*j;
>>    zr = real(z);
>>    zi = imag(z);
```

real and **imag** break **z** into real and imaginary parts in the variables **zr** and **zi**, respectively.

Note that typing a semicolon at the end of the command line prevents MATLAB from echoing the result back to your screen. This will be important when you create large matrices and vectors. In this example, the value of **z** was not printed as it was above.

To plot a complex number, we can either plot the real parts vs. the imaginary parts or let MATLAB do it for us. Here, we supply **zr** and **zi** to the plot function:

```
>>    plot (zr,zi,'x')
```

and MATLAB generates a plot:

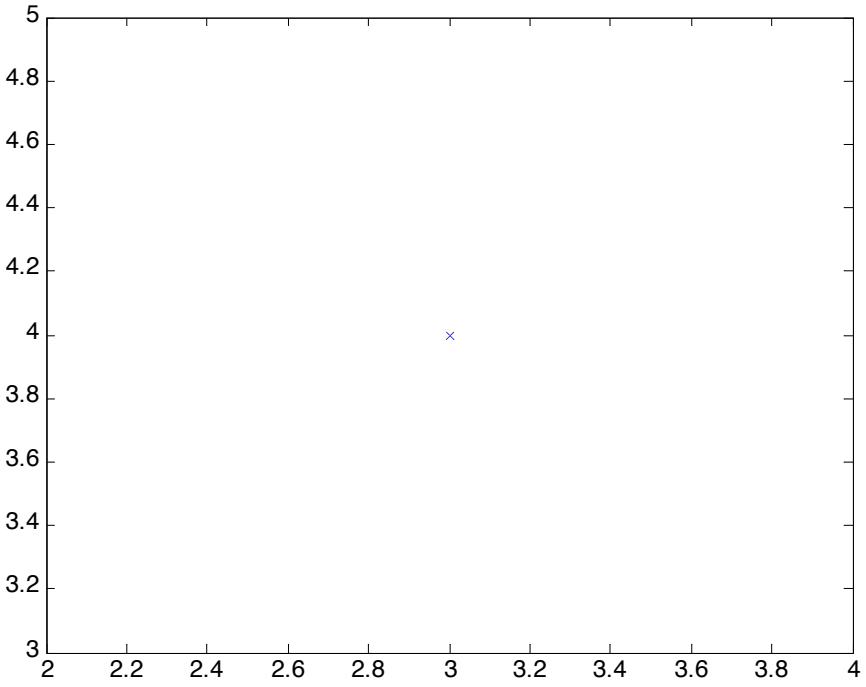


Figure 13.1 Example plot of a complex number

The '**x**' parameter to the plot function tells MATLAB to generate an x shape for each data point instead of a 'connected-dot' display. Since we only plotted one data point, this is extremely useful. In general you should always label axes on your plot and include a title. **help plot** shows you the other characters that can be used as well as the different colors that can be used on the plot.

This plot would be exactly the same if we had entered:

```
>>    plot(z,'x')
```

since the MATLAB default for plotting complex numbers is to plot the real parts on the horizontal axis and the imaginary parts on the vertical axis.

Multiple sets of parameters can be given to **plot**; each pair of arguments is taken as *x* and *y* data pairs.

If you wish to have several plots shown at once on different sets of axes, use **subplot** - see the on-line help - or open new figures by typing **figure** and creating a whole new plotting window.

1.5 Vectors and Matrices: Tutorial, Part II

Matrices and vectors make up the heart of MATLAB computations. In this section, matrix and vector manipulations will be introduced. A vector is a one-dimensional list of values, an m x 1 or 1 x *m* matrix. Vectors hold single signals or lists of data. They can be assigned a name and treated as any other variable in MATLAB; however, operations performed on vectors are done element by element.

As an example of this, consider the function $y = 3x+2$. If we want to plot y as a function of *x*, we first create an *x* vector containing data points in the range of interest. Suppose the range is 0 to 5, using every integer point. There are several ways to create this data set in MATLAB. The first way is to type in every point:

```
>>    x = ( 0 1 2 3 4 5 ];
```

This generates a row vector **x**, i.e. a 1 x 6 matrix containing six elements, the integers 0 through 5. Note that the semicolon keeps MATLAB from echoing the results of your command back to the screen. An easier way to generate this same vector is to use a range-generating statement:

```
>>    x=0:5
x =
012345
```

The colon operator acts like the word "to", in effect generating the function "0 to 5". A step size of 1 is the default. A different step size, positive, negative, real or integer, can be specified by placing the step value between the beginning and end of the range, as in **z** below:

```
>>    z = 0:0.01:5;
```

generates 501 data points that are 0.01 apart, starting from 0 and ending at 5.

The next step is to evaluate the function y, using the x as defined above:

```
>> y=3*x+2

y=
2 5 8 11 14 17
```

This statement instructs MATLAB to multiply every element in **x** by 3 and then add 2 to every element, storing the results in **y**. Thus 3***x** is treated as scalar multiplication of a vector and the 2 is implicitly treated as a vector of the same length as **x** comprising all 2s.

Since MATLAB is based on matrix operations, it is important to recall that you can only add or subtract matrices having the same dimensions, e.g., the addition of a 3x2 matrix

with a 2x3 matrix is undefined. Matrix multiplication requires that the number of columns in the first matrix be the same as the number of rows in the second matrix. For example, multiplication of a 2x5 matrix A with a 5x3 matrix B results in a 2x3 matrix **C=AB**, whereas the multiplication **BA** is undefined. However, the multiplication **D=B'A** is defined, where **'** denotes the transpose operation in MATLAB.

1.5.1 Generating Complex Functions

Let's generate values for the complex function $f(t) = 3e^{j3\pi}$ for t ranging from 0 to 1 in 0.001 increments. The first step is to create a time variable; note the use of the **:** operator with a noninteger step size.

```
>>    t = 0:0.001:1;
```

Here the semicolon at the end is especially important, unless you really want to see and wait for all 1000 values to be echoed back on your screen.

Next, construct a vector containing values of this function for each time value in t:

```
>> f = 3*exp(j*3*pi*t);
```

It should be pointed out that transcendental functions (e.g. sin, cos, exp) in MATLAB work on a point-by-point basis; in the above command, the function exp computes a vector where each element is the exponential of its corresponding element in **j*3*pi*t** (1001 total elements).

1.5.2 Accessing Vectors and Matrices

The data in vectors can be viewed and displayed in several different ways: it can be plotted, printed to the screen, printed on paper, and saved electronically. It is not, however, always desirable to access the entire vector at once when displaying the information in it. To access single elements or ranges of a vector, an index element or list that identifies which elements are of interest is needed.

Elements in MATLAB vectors are identified by the vector name and an integer number or index, much in the same way that DT signals are indexed by integer values. However, in MATLAB, only positive integer indices are used. Thus the first element in a row or column vector f is denoted by **f (1)**, the second element by **f(2)**, and so forth. To access specific elements in a vector, you need to use the name of the variable and the integer index numbers of the elements vector you wish to access. Range statements can be used for indices to access the indexed elements much in the same way that range statements are used to define vectors comprising values in a specified range. For example,

```
>>    f(25);
```

```
>>    f(3:10);
>>    f(1:2:50);
```

The first line accesses the 25th element of **f**. The second accesses elements 3 through 10, inclusive; and the third statement returns the odd-numbered elements between 1 and 50.

Elements in matrices require use of two-dimensional indices for identification and access. For example, **f (3,2)** returns the element in the third row, second column; ranges can also be used for any index. For example, **f (1:3, 4:8)** defines a matrix that is equivalent to a section of the matrix f containing the first, second, and third rows, and the fourth through eighth columns.

If a **:** is used by itself, it refers to the entire range of that index. For example, a 3 x 5 matrix could have its fifth column referenced by **f (: , 5)**, which means "all rows, 5th column only," as well as **f (1:3, 5)**, which means rows 1 to 3, 5th column only.

The index number can be another variable as well. This is useful for creating programming loops that execute the same operations on the elements of a matrix.

From other computer programming experiences, you should be familiar with the idea of creating a loop to repeat the same task for different values. Here's an example of how to do this using MATLAB. Suppose we want to generate an output vector where each element is the sum of the current element and the element from 10 back in an input vector. The task to be repeated is the sum of two elements; we need to repeat this for each element in the vector past 10. The elements of the vector **x** define the input ramp function to be integrated, **y** will hold the result, and **k** is the loop index:

```
>>x = 3*(0:.1:5) + 2;
>>y=zeros(size(x));
>>for k=11:length(x)
y(k)=x(k-10) + x(k);
end
>>
```

When a loop structure is entered in MATLAB, the body of the loop does not have a prompt printed; the command line acts the same as with a prompt. Note that the loop will not be executed until the **end** command is entered and followed by a carriage return. You will wait forever for the loop results if you leave off end (indicating to MATLAB that you have done all you wanted to in the loop).

As you can see, the variable **k** is set to range from 11 to the length of **x**; this allows **k** to index all elements in **x**. **k** is set to increment by integers.

Loops generally are not desirable, since they take a very long time to run. If possible, rewrite your operations in terms of vector additions and multiplications instead of

looping.

For example, we can rewrite this problem to use vector addition by creating two new vectors, one which is **x** offset by 10 and the other which is **x** padded with 10 zeros (since we can only add vectors of similar lengths).

```
>>x = 3*(0:.1:5) + 2;
>>x1 = [zeros(1,10) x];
>>x2 = [x  zeros(1,10)];
>>y = x1+x2;
```

Note that using vector addition in MATLAB is significantly more efficient than using for loops.

1.5.3 Storing Results and M-Files

Usually you want to save the results that you have generated during a MATh1~ session, including data vectors created and commands used to process them. This can be accomplished by:

1. Using the **diary** command to save a record of all commands typed. Enter **help diary** to learn how to use this command.
2. Using M-files that you have created using a text editor; such files have a .m extension and contain a list of MATLAB commands to be executed when you type the root filename.
3. Saving the contents in variables for the next session using the **save** command.

It is recommended that you become familiar with M-files. They are extremely useful and will save you much time and effort. There are two types of M-files: scripts, which are essentially a series of commands typed into a file instead of typed at a MATLAB prompt; and functions, which allow you to create new MATLAB functions.

Some of the later problems will use M-files as part of the solution. To edit M-files, any text editor can be used - not word processors like MS Word or Framemaker. The Macintosh versions of MATLAB have a built-in editor; PC and X Windows versions rely on outside editors such as emacs. Note that these editors can be run in a separate window to allow editing and processing to occur concurrently.

1.6 ASSIGNMENT PART I

Now that you have been introduced to MATLAB, larger and more complex problems can be handled. This section will present an example of how to solve a problem using MATLAB. In addition, you will be given several problems with answers, but not the steps taken to generate the answers. You should work through these problems during the

laboratory session.

1.6.1 Question 1

Add the functions $p = 3\sin(x^2) + 2\cos(y^3)$ and $q = 3\cos(xy) + 2y^2$ for x in the range 0 to 5. $y = 0.05x + 2.01$ in all cases. Use increments of 0.01. Plot all three functions.

Solution:

This can be done in two different ways. The easiest way is to make two vectors containing the individual functions and a third vector containing the sum. The alternative is to algebraically substitute in x, y, p and q into the total expression and reduce. This is extremely unpleasant.

Note that when a dot precedes an operator, as using **.*** for multiplication, it implies that each element in the vector (matrix) results from applying that operator to corresponding elements in the first and second vectors (matrices). For example, dot multiplication of two m x 1 vectors results in an m x 1 vector where each element is the product of the corresponding elements in the first and second vectors. Note that **.*** is equivalent to the "dot-product", or inner product operation from linear algebra. This type of multiplication requires that the vectors (matrices) must be of the same size and is called pointwise, rather than vector or matrix, multiplication.

```
>>x =  0:0.01:5;
>>y = .05*x+2.01;
>>p =3*sin(x.*x)+2*cos(y.*y.*y);
>>q=3*cos(x.*y) + 2*y.*y;
>>z = p + q;
>>plot (p);
>>plot (q);
>>plot(z);
```

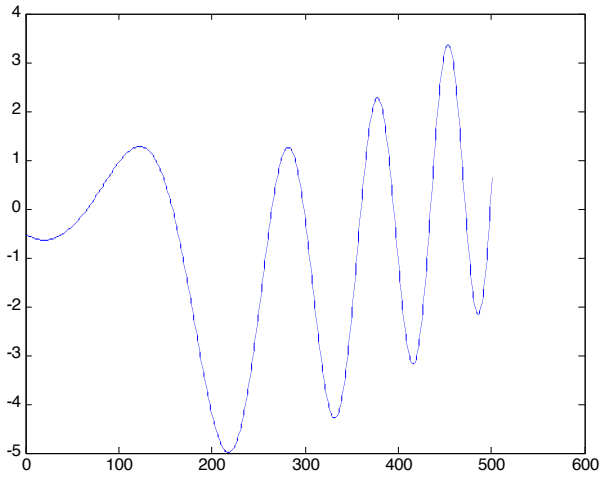


Figure 1.4.1 Plot of p

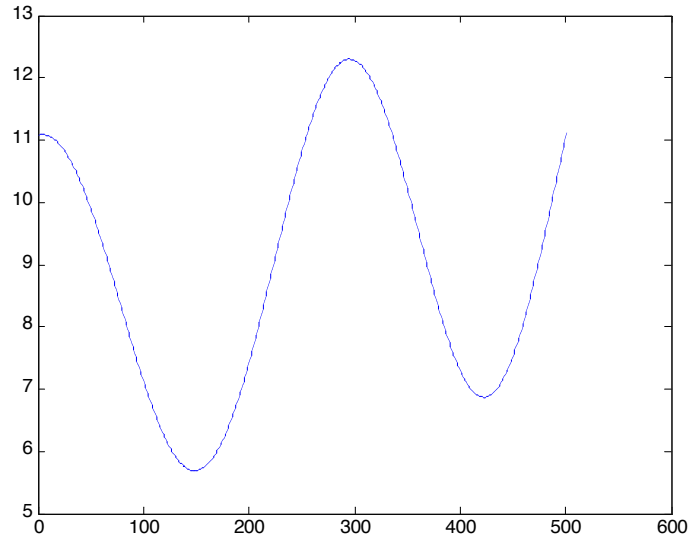


Figure 1.4.2 Plot of q

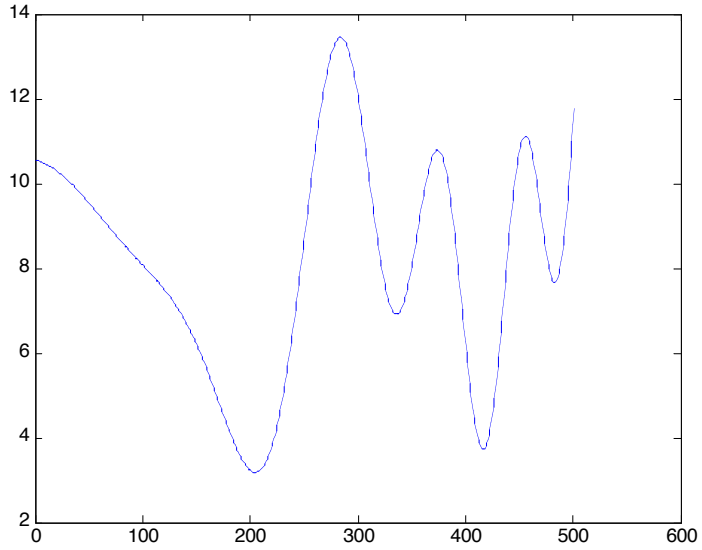


Figure 1.43 Plot of z

1.6.2 Question 2

Plot the function $y(t) = \left(1 - e^{2.2t}\right)\cos(60\pi t)$. Use t from 0 to 0.25 in 0.001 increments.

Answer:

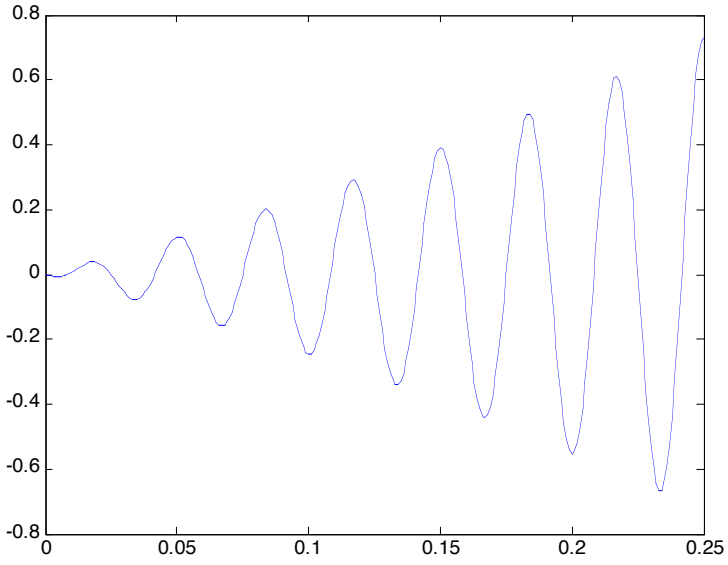


Figure 1.4.4 Solution to Problem 2

1.6.3 Question 3

A polynomial function has roots at -2, 2, -2+3*j*, -2-3*j*. Determine the polynomial, plot the four roots in the complex plane, and plot the polynomial function for the range $x \in$ (-5,5) in steps of 0.01. You may wish to use **help** to look up the functions **poly**, **roots**, **residue** and **polyval**.

Answer: The polynomial is $x^4 + 4x^3 + 9x^2 - 16x - 52$

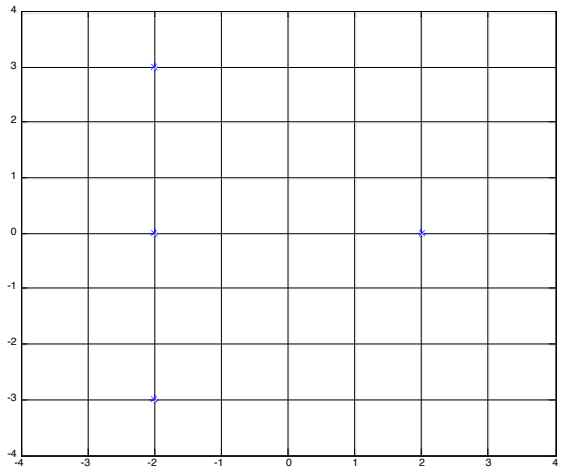


Figure 1.4.5 Roots of Problem 3

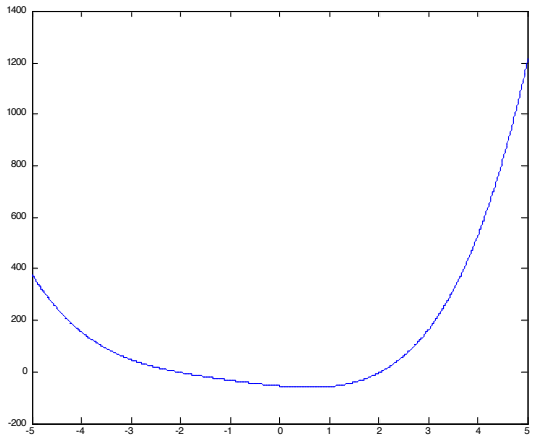


Figure 1.4.6 Plot of Problem 3

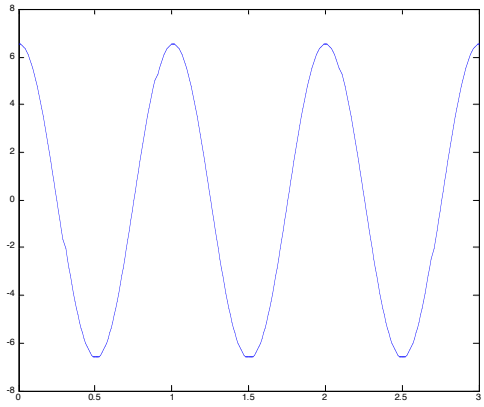
1.6.4 Question 4

The complex function *f(t)* has the form: $f(t) = 3e^{-j2\pi t + \pi/4}$.

Plot the real and imaginary parts as a function of time from 0 to 3 seconds in 0.01-second increments. Also plot the magnitude and phase of *f* as a function of time. You may wish to look up the functions **subplot, title**, and **plot** to see how to generate more than one plot in the graphics window at the same time. You will also need **abs** and **angle**. These commands are very useful in signals and systems.

Answer:

Re(*f(t)*): Im(*f(t)*):



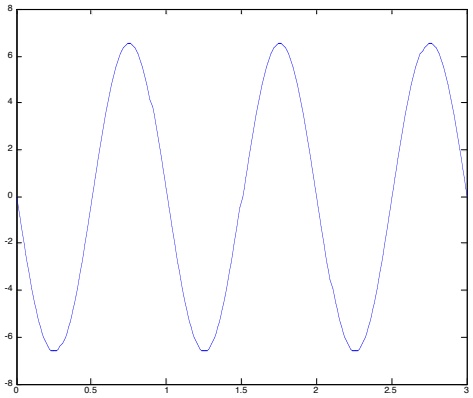
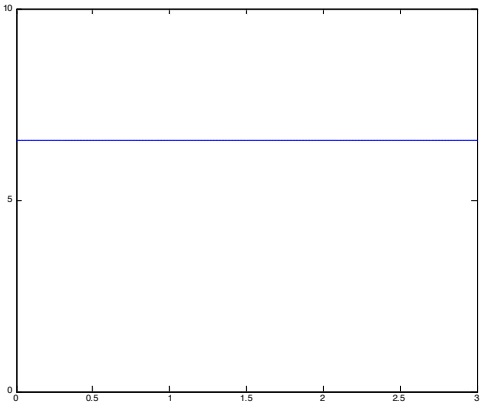


Figure 1.4.7 Real and Imaginary plots

Magnitude of *f(t)*: Phase of *f(t)*):



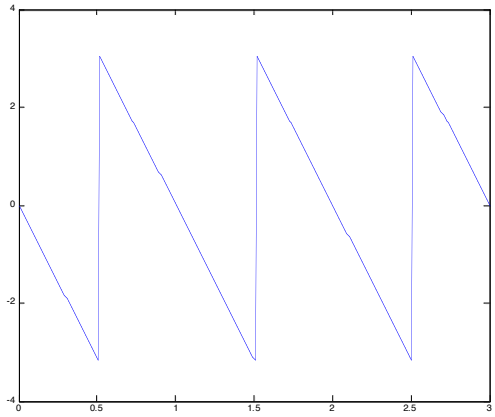


Figure 1.4.8 Magnitude and Phase plot

1.7 ASSIGNMENT PART II

This section will focus on the creation of MATLAB M-files. The solution to a problem is the M-file itself. We have provided you with the output of our M-file (without echoing the generation commands to the screen). This is done by using the command **echo off** at the beginning of the M-file.

1.7.1 Creation of a Function in MATLAB

A function in MATLAB is a special kind of M-file. The first line in the file defines the function, both giving it a name and indicating what values are to be passed as arguments to the function and those that are to be generated by the function, much like a subroutine in other programming languages. Once you have created a function you can use it just as would use a MATLAB-supplied function.

A useful feature of function files is that the lines that follow the function definition and begin with a comment symbol (%) are printed when help is requested for your function (**help yourfunction** prints these lines).

Make a function that takes two variables as arguments, adds 1 to the first variable, multiplies the second variable by two, and returns the product of the two variables. This function is called **blackbox**; the file is called **blackbox.m**

The following M-file is generated.

```
function [output] = blackbox(a,b)
% Adds 1 to argument a and multiplies b by 2
% Returns the product of a and b in the output.
%
% If the two variables are not of the same size,
% the larger variable is stripped to be the same
% size as the smaller.

% Determine the lengths of each vector
la = length(a);
lb = length(b);

% Add 1 to a
a = a+1;

% Multiply b by 2
b = b*2;

% Compare the lengths: if a is shorter than b, truncate b
% Otherwise, truncate a.
if la < lb
output = a .~ b(1:la);
else
output = a(1:lb) .~ b;
end
```

The elements in this file that are required in order for MATLAB to recognize it as a valid function are (1) the filename and the function name must be the same, (2) the vector **[output]** is the function output, (3) and, finally, theM-file must start with the word

function to indicate to MATLAB that it is a function that may require inputs.

1.7.2 Question 5

Generate a file that calculates a sine wave of 5 Hz for 3 seconds using 0.001-second increments and plots the sine wave versus time with all axes labeled. Display the length of the time sample and the length of the sine wave calculated.

Answer:

```
>> sine
Length of time vector
3001
Length of sine vector
3001
>>
```

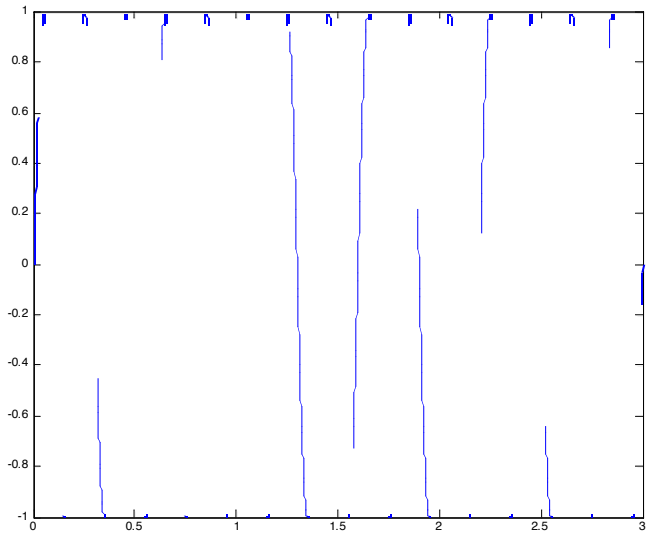


Figure 1.5.1 Solution to Question 5

1.7.3 Question 6

Generate a function called squarer that returns the vector with each element squared. Use this function with the sine wave from the previous problem and display the sum of the elements in both the sine wave vector and the vector composing square of the sine wave.

Answer: the file prob5.m

```
>> prob5
Sum of sine
9. 5372e-14
Sum of sine squared
1.5000e+03
>>
```

Note that the sum of a sine squared should be 0.5 if we interpret the sum as an approximation to an integral. The function returns 1500 because there are 1000 samples per period and three periods. 1500 / 3000 is 0.5. You must be cautious when interpreting results because you are dealing with discrete elements in the vector instead of a true continuous function.

Note also that if the sum of the sine function accurately approximates the integral, it should be 0. It is, instead, a very small number. This error is due to rounding errors in the least significant place of every sine value calculated and is an artifact of numerical computing. If you expect a sum to be equal to zero, don't be surprised if it is very small instead of zero.

A brief word about help: MATLAB's help can be useful, but it can also be confusing. If you type help by itself, you will see a list of directories and a description of their contents. **help <directoryname>** will show you the contents of a directory; **help <function>** gives you the help associated with the function. Sometimes these are cross-referenced, sometimes they aren't.

1.8 Lab Problems

Submit solutions to the following problems with your lab write-up.

1.8.1 Problem 1

Compute $\frac{(2+j5)(1-j5)}{(1+j7)(3+j2)}$ and express your answer in both rectangular and polar coordinates.

1.8.2 Problem 2

An interesting plot is generated by the complex function $r = 1 - \cos\theta$, where r is the radius of a complex number z expressed in polar coordinates and θ is the angle. Sweep θ from 0 to 2π and plot the real and imaginary parts of $z = r\ exp(j\theta)$ as x and y coordinates, respectively.

1.8.3 Problem 3

In communications, a signal is called a power signal if it has a zero time average

$\bar{x} = \frac{1}{N}\sum_{k=0}^{N-1} x[k]$ and a nonzero, finite time average of its square $\bar{x}^2 = \frac{1}{N}\sum_{k=0}^{N-1} x^2[k]$; in contrast, a message signal has a nonzero time average and a nonzero, finite time average of its square. Create a function that determines if a signal is a message signal or a power signal. It should return the value 1 if the signal is a message, and 0 if the signal is power. Test your function with the power signal sin(10t) and the message signal random(*t*). Let t = [0,1] and use increments in time of 0.01. Use the **randn** function to get a normal (i.e., Gaussian) distribution (see the on-line help for **randn**).

1.8.4 Problem 4

Some MATLAB functions require that a vector representing time samples be in the first column of a matrix and the corresponding signal values at each time be in the second column. Such a matrix, called A, has been created. Write down three MATLAB expressions to extract the time vector into the variable *t*, the signal vector into the variable *x*, and the 100th time/signal pair into the matrix *y*. *Hint: Use* **:** *notation*.

1.8.5 Problem 5

Write a MATLAB function **half** that removes every other element from an arbitrary length vector, creating a shorter vector made of only the odd-numbered elements of the original vector; and a MATLAB function **double** that creates a longer vector by adding an additional element between neighboring elements in the original vector. Each new element should equal the average of its neighboring elements. Use only matrix/vector manipulations; do NOT use loops. Test your solution by applying **half** and then **double** to the vector **x = [1 2 … 6 7 6 …. 2 1]**. What happens after the execution of **half** followed by **double**, and **double** followed by **half**?