MATLAB® INTRODUCTION

MATLAB® (matlab from now on) is a powerful software program for writing numerical analysis software used in simulating, analyzing and designing systems. In fact, it has become a *de facto* standard in many engineering communities. Because it is so easy to play "what if?" with this program, we will use it extensively throughout the semester. First, we need to become familiar with the software.

Objectives:

1. Use matlab to perform complex arithmetic
2. Generate and plot signals and complex valued functions
3. Develop matlab programs and save and load data
4. Develop some programming 'savvy'

First, you should learn how to invoke matlab. This will depend on which version you are using and what machine you are using. ECN has the windows version for the PC, and you can purchase the Student edition (windows or non-windows) at the bookstore. In addition, some of you may have access to Unix versions for various machines or the Mac version. Each computer system is slightly different, but once you are in matlab, the look and feel will be the same. The first time you invoke matlab, you should run several of the demonstration programs, invoked by typing "demo" at the matlab prompt (indicated by the >>).

Matlab provides on-line help using the "help" command. For example, consider the command:

```
» help filter

 FILTER One-dimensional digital filter.
    Y = FILTER(B,A,X) filters the data in vector X with the
    filter described by vectors A and B to create the filtered
    data Y.  The filter is a "Direct Form II Transposed"
    implementation of the standard difference equation:

    a(1)*y(n) = b(1)*x(n) + b(2)*x(n-1) + ... + b(nb+1)*x(n-nb)
                          - a(2)*y(n-1) - ... - a(na+1)*y(n-na)

    If a(1) is not equal to 1, FILTER normalizes the filter
    coefficients by a(1).

    When X is a matrix, FILTER operates on the columns of X.  When X
    is an N-D array, FILTER operates along the first non-singleton
    dimension.

    [Y,Zf] = FILTER(B,A,X,Zi) gives access to initial and final
    conditions, Zi and Zf, of the delays.  Zi is a vector of length
    MAX(LENGTH(A),LENGTH(B))-1 or an array of such vectors, one for
```

```
    each column of X.

    FILTER(B,A,X,[],DIM) or FILTER(B,A,X,Zi,DIM) operates along the
    dimension DIM.

    See also FILTER2, FILTFILT (in the Signal Processing Toolbox).
```

At this point, it does not need to be clear just what the actual usage of this command is, just recognize that the help facility gives you all of the necessary input/output information to use the command.

Matrices: variables and operations

Now, notice that the inputs and outputs are defined as vectors. This is the strength of matlab. For example:

```
» M = [1 2 6; 5 2 1]

M =

     1     2     6
     5     2     1

» N = [2 2 2; 1 1 1; 3 3 3];
» whos
  Name      Size         Bytes  Class

  M         2x3             48  double array
  N         3x3             72  double array

Grand total is 15 elements using 120 bytes
```

The function “whos” lists all variables in the workspace. Note that the “;” suppresses printing the result! The size of any variable can be obtained at any time using the “size” command:

```
» size(M)

ans =

     2     3
```

The matrix M has 2 rows and 3 columns, and is listed by whos as a 2x3 matrix variable. We can create a vector using the “:” operator:

```
» firstrow=M(1,:)

firstrow =

     1     2     6
```

```
» thirdcolumn=M(:,3)

thirdcolumn =

     6
     1
```

This operator can be very useful in addressing portions of a matrix. We will also often use it to define regularly spaced vectors using a shorthand notation:

```
» t=0:0.01:1;
» size(t)

ans =

     1   101
```

The t vector is the vector (of times) that is evenly spaced $\frac{1}{100}$ seconds between 0 and 1 second inclusive. We'll print the first few elements to get a better feel of what this means:

```
» t(1:5)

ans =

         0    0.0100    0.0200    0.0300    0.0400
```

Matlab operates on vector variables directly. Let's recall matrix multiplication. If we want to multiply two matrices, let's say the matrix M and N defined above, what is the first thing that we must check? Note that the number of columns of M is equal to the number of rows of N. Thus, $M \times N$ is defined, and the result has a size calculated

$$(2x3)\times(3x3)=(2x3)$$
$$\uparrow \qquad \uparrow$$

The computation of $N \times M$ is not defined because the number of columns of N do not match the number of rows of M. Matrix multiplication is defined using the dot product of two vectors:

$$\langle x,y\rangle \equiv \begin{bmatrix} x_1 & x_2 & \dots & x_N \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$
$$= x_1y_1 + x_2y_2 + \cdots + x_Ny_N$$

Then, each element of the resultant matrix is the dot product of the corresponding row and column of the two multiplied matrices. In detail, the element in the 1st row and 1st column of the resulting matrix is the dot product of the 1st row of left hand multiplier "dotted" with the 1st column of the right hand multiplier, and so forth. In our example, the result is:

```
» P=M*N

P =

    22    22    22
    15    15    15
```

Note that the result in the 2nd row and 3rd column can be obtained:

```
» M(2,:)*N(:,3)

ans =

    15
```

Subtraction and addition are supported, the matrix sizes must be identical in each case for the operation to be defined. The result is a point-by-point subtraction or addition. Matlab also provides a point-by-point multiplication (and division and power) operator via the .* (or ./ or .^) operators. In our example, the two matrices are of unequal length. However, we can use a portion to provide an example of this powerful contruct:

```
» M

M =

     1     2     6
     5     2     1

» N

N =

     2     2     2
     1     1     1
     3     3     3

» M.*N(1:2,:)

ans =

     2     4    12
     5     2     1
```

Matrix division involves the notion of "inverse" which we do not use in this class. However, the easiest manner to divide appropriate sized matrices is to use the "\" operator, as in the example

```
» [1 2 3;4 5 6;7 8 10]\[1;2;3]

ans =

   -0.3333
```

```
    0.6667
         0
```

Sometimes when using the division, you'll get something like

```
» [1 2 3;4 5 6;7 8 9]\[1;2;3]

Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 2.055969e-018.
```

This means that all results are subject to user scrutiny; often the results are incorrect!

Predefined variables

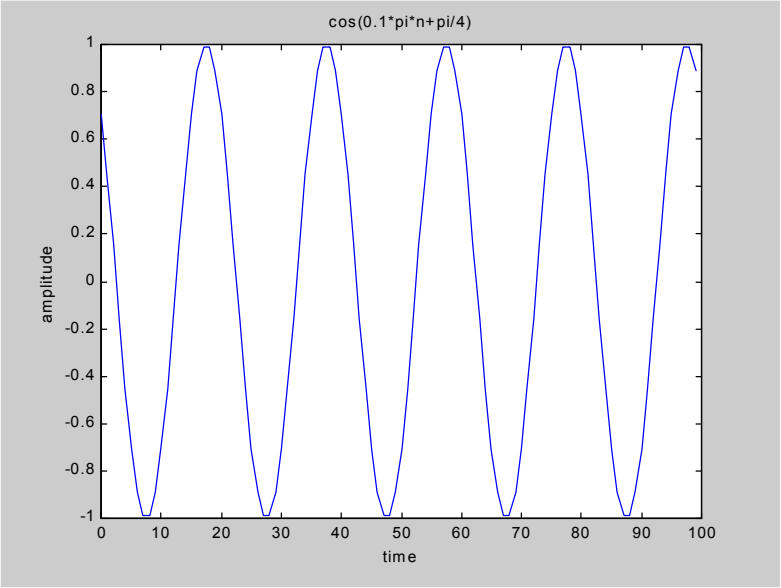
The variables "j" and "pi" are pre-defined variables in matlab which are set every time matlab is invoked. They can be changed during a session, so be careful! If you change a predefined variable, you can reset the value simply by typing the "clear x" command, where 'x' is the variable name, e.g. "clear pi" resets the variable pi to its correct value of 3.1416.

Graphics

In this class, we'll use two plotting commands extensively: "plot" and "stem." Consider the following matlab code:

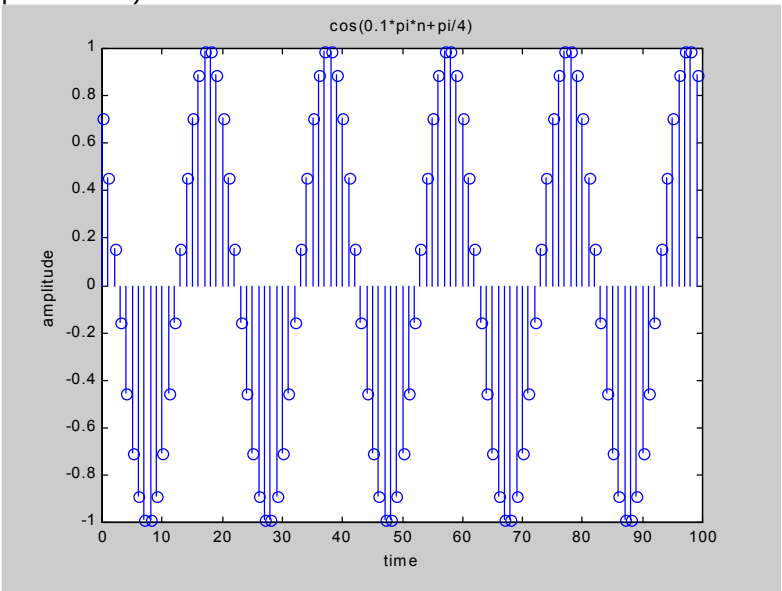
```
» n=0:99;
» x=cos(0.1*pi*n+pi/4);
» plot(n,x)
» title('cos(0.1*pi*n+pi/4)')
» xlabel('time')
» ylabel('amplitude')
```

that produces the plot



Note that this plot produces a "continuous-time" plot of the waveform. A "discrete-time" plot of the waveform is produced using the following

```
» n=0:99;
» x=cos(0.1*pi*n+pi/4);
» stem(n,x)
» title('cos(0.1*pi*n+pi/4)')
» xlabel('time')
» ylabel('amplitude')
```



Options such as line colors, types and end heads can be changed, see "help plot" and "help stem" for details. Matlab can draw to multiple figure windows using the "figure" command. Furthermore, multiple plots in a single window can be drawn using the "subplot" command. The "print" command allows graphics windows to be printed or saved to a file. On ECN, this command seems to cause the most problems for students. Ask the SA on duty for help when needed. If you can't get satisfaction from the SA, report the problem, the time, the room, and the SA's name to me. The director of ECN, John Hawley, has directed the faculty to report this information so that he can improve service. The "copy" command in the graphics window menu can also be used to import the graph into windows documents such as word processors.

Programming constructs and flow

Sequences of commands that you plan to commonly use can be saved in an ".m" file. M-files come in two types: Command sequences and Functions. Command sequences may be edited in any file with the .m extension, and run from matlab simply by typing the name of the file without the extension. In version 5, the matlab-provided editor is preferred. All variables created in the execution of the M-file will be kept in the matlab memory (and consequently seen by executing the "whos" command). Names of M-files in the current directory can be obtained by typing the "what" command, which also shows the names of .mat (data) files. Functions are also typed in a text file with the extension .m. However, functions are like program subroutines--- variables are passed

and returned using a conventional syntax, and any variables created inside the function are not available outside of the function.

For example, let's consider a timing problem. A Command sequence file is written and stored in the file timing, the text of which is shown in Figure 1.

```
clear
N=6000;
t=cputime;
for k=1:N
    x(k)=0;
end % for
loop_time=cputime-t
t=cputime;
x=zeros(1,N);
com_time=cputime-t
```

Figure 1: Text of timing.m

Typing “timing” at the matlab prompt will run this matlab script. Notice that the commands are echoed to the screen. You may want to include the line “echo off” to suppress this feature of matlab (See help on “echo.”). Typing “whos” after executing this file (i.e. after typing “timing” at the matlab prompt will show that all declared variables are available in the workspace!

Let's take an example of a matlab function. In fact, all of the matlab commands you have been using so far are functions. Many (though not all) are even M-files, which you can copy over to your working directory and change! An example of a user-defined M-file is the Sa(x) function written using the built-in Sinc(x) function from matlab:

```
function y=Sa(x)
% function y=Sa(x)
% compute sine-over-argument
y=sinc(x/pi);
```

Figure 2. Example Functions

Notice that the input and output variables are defined in the function call. The text on a line following a "%" character is a comment. Typing “help Sa” at the matlab prompt after saving this file results in the comment lines immediately after the function declaration line (the first line) and before the first executable matlab command being echoed to the screen. You should always provide for this facility when you write M-files. Type “whos” and you will see that no internal variables (like y) are available.

This example function doesn't show it, but advanced programmers always check input and output arguments as much as possible to prevent misuse of the function and to provide programming flexibility. Matlab provides the built-in variables “nargin” and

"nargout" to keep track of the number of input and output variables. You can use the nargin variable to build subroutines that use a variable number of inputs, where the number of inputs depends on the particular use of the routine.

Finally, matlab control functions follow the well-developed outline in the class text:

- if-else
- for loops
- while loops

Explanation and examples will follow throughout the semester via examples.

Tips and programming savvy

Data may be saved and loaded using the "save" and "load" commands (see help on the precise syntax you may want to use). Data files are saved with the extension ".mat."

Some very useful commands for initializing vectors and matrices are the functions "zeros" and "ones." The size of a matrix or vector is obtained using the commands "size" or "length." Command flow may also be directed by matlab commands. For instance, we may initialize a vector of zeros using the following sequence of commands:

```
>> for k=1:1000
>> x(k) = 0;
>> end
```

We may also use the single command

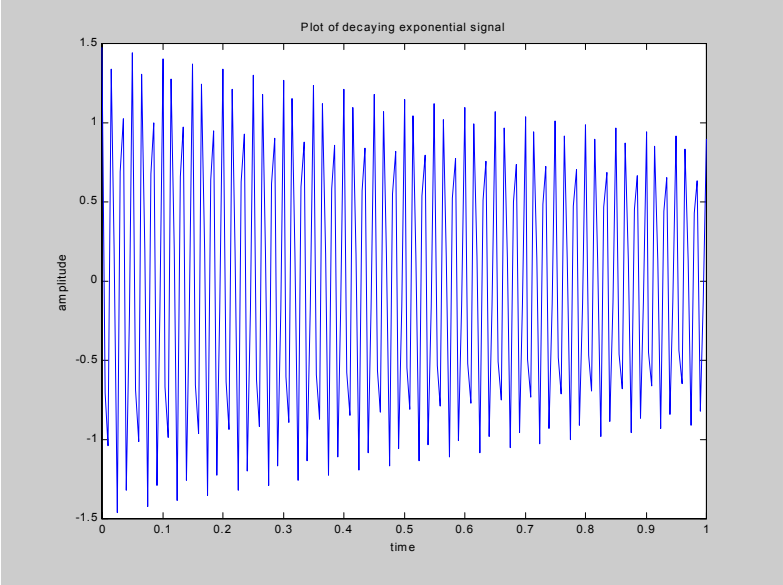
```
>> x = zeros(1,1000);
```

The second method is preferred because matlab is an interpretive language. Because matlab interprets commands during run-time, matlab must determine the end of the loop each time control passes through the loop (you may have conditions to break out of the loop), and matlab must also allocate one new memory location during each pass to accommodate the increasing size of the vector x. This means that loops should be avoided if at all possible in matlab, and they can almost always be avoided by careful use of the indexing options and the "." operators (.*, ./ and .^). For instance, suppose that we want to generate the decaying sinusoid:

$$x(t) = 1.5e^{-0.5t}\cos\left(2\pi 60t + 10°\right), t \geq 0$$

We could use the loop sequence:

```
» t=0:0.005:1;
» for i=1:length(t)
      x(i)=1.5*exp(-0.5*t(i))*cos(2*pi*60*t(i)+pi/18);
  end % for i
» plot(t,x)
» title('Plot of decaying exponential signal')
» ylabel('amplitude')
» xlabel('time')
```



This for loop requires 0.76 seconds to run on my 200MHz pc. A better matlab construct uses the ".*" command:

```
» x=1.5*exp(-0.5*t).*cos(2*pi*60*t+pi/18);
```

This option only requires 0.01 seconds to run on my 200MHz pc, a savings of 76 times! In more sophisticated programs, this time savings becomes extremely significant.

Matlab maintains several ways to obtain hardcopy results. The "diary" command can be used to record your keystrokes and matlab results in an ASCII text file. See help to learn how to use this command.