

WASHINGTON UNIVERSITY - IN - ST. LOUIS
DEPARTMENT OF ELECTRICAL ENGINEERING

MATLAB
AN INTRODUCTION

BY

MICHAEL DEVORE
M. ALI RAUF

1.0 Starting a Matlab Session

To start a Matlab session, type Matlab at the Unix prompt:

```
clarion.cec> matlab
```

If Matlab is not already part of your account package, you should add it by using the `pkgaddperm` command: `'pkgaddperm matlab.'` Once you have added the Matlab package to your account start a Matlab session.

On the PCs, Matlab can be accessed from the start menu. Go on to Engineering and then select Matlab.

2.0 Using Matlab: Fundamentals

Matlab has essentially one kind of object, a rectangular matrix. All elements created in Matlab are rectangular matrices. Even scalars are created as 1×1 matrices. All operations and commands are treated in a matrix sense, almost exactly as they are usually written on paper.

2.1 Creating Simple Matrices

Matrices can be created in Matlab by several methods:

- Entering explicitly
- Generated by statements and functions
- Created in m-files
- Loaded from external data

Specify an array with brackets `[]`, use commas to separate elements and semi-colons to separate rows. You can generate a small matrix explicitly by entering the following at the Matlab command prompt:

<pre>>> A=[1,2,3,4; 4,2,1,7; 9,8,9,3; 5,2,8,6]</pre>	$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 4 & 2 & 1 & 7 \\ 9 & 8 & 9 & 3 \\ 5 & 2 & 8 & 6 \end{bmatrix}$
--	--

The following command creates a row vector `t = [-5.00, -4.99 -4.98 4.98 4.99 5.00]`.

```
>> t = -5:0.01:5;
```

This is a vector starting at -5 and we add 0.01 to it until we reach 5 which is the last element of the vector. By default Matlab creates row vectors (row vector = $1 \times N$ matrix). Curious about what the semi-colon is for? Try the same command without it! Try the following command.

```
>> B=[1,4,9,5; 2,2,8,2; 3,1,9,8; 4,7,3,6]
```

Matrix elements can be expressions, complex numbers, or strings. For example, use symbols `i` or `j` to denote imaginary numbers:

<pre>>> 5+7i</pre>	<code>ans = 5.0000 + 7.0000i</code>
<pre>>> C=[1+2i,2+3i,3+4i,4+i; 4+2j,2+j,1+7j,7+4j; 9+8i,8+9i,9+3i,3+9i; i 1 j 2]</pre>	$C = \begin{bmatrix} 1.0000 + 2.0000i & 2.0000 + 3.0000i & 3.0000 + 4.0000i & 4.0000 + 1.0000i \\ 4.0000 + 2.0000i & 2.0000 + 1.0000i & 1.0000 + 7.0000i & 7.0000 + 4.0000i \\ 9.0000 + 8.0000i & 8.0000 + 9.0000i & 9.0000 + 3.0000i & 3.0000 + 9.0000i \\ 0 + 1.0000i & 1.0000 & 0 + 1.0000i & 2.0000 \end{bmatrix}$

Some constants already defined in Matlab include: `pi`, `eps`, `Inf`, `NaN`.

You can look at variables in the workspace with 'whos'...

>> whos	Name	Size	Bytes	Class
	A	4x4	128	double array
	B	4x4	128	double array
	C	4x4	256	double array (complex)
	ans	1x1	16	double array (complex)
	x	1x4	32	double array
Grand total is 53 elements using 560 bytes				

2.2 Matrix and Array Operations

Arrays are subscripted with () and indexed from 1.

>> A(1,3)	ans = 3
-----------	---------

>> A(1,:)	ans = 1 2 3 4
-----------	---------------

Arrays can be concatenated by [].

>> [A, B]	ans = 1 2 3 4 1 4 9 5 4 2 1 7 2 2 8 2 9 8 9 3 3 3 9 8 5 2 8 6 4 4 3 6
-----------	---

>> [A; B]	ans = 1 2 3 4 4 2 1 7 9 8 9 3 5 2 8 6 1 4 9 5 2 2 8 2 3 1 9 8 4 7 3 6
-----------	---

Matrix addition and subtraction is performed in the usual way:

>> A+B	ans = 2 6 12 9 6 4 9 9 12 9 18 11 9 9 11 12
--------	---

>> A-B	ans = 0 -2 -6 -1 2 0 -7 5 6 7 0 -5 1 -5 5 0
--------	---

Scalar operations apply to each element individually: 5+A, 5-A, A-5, 5*A, A/5 etc.

>> 5+A	ans = 6 7 8 9 9 7 6 12 14 13 14 8 10 7 13 11
--------	--

>> A/5	0.2000 0.4000 0.6000 0.8000 ans = 0.8000 0.4000 0.2000 1.4000 1.8000 1.6000 1.8000 0.6000 1.0000 0.4000 1.6000 1.2000
--------	--

With matrix multiplication and division, * and / operate on whole matrices while .* and ./ go element-by-element.

>> A/B	0.3519 -0.3677 0.2573 0.1529 ans = -0.7223 -0.3374 0.9126 0.6650 -1.8981 3.0073 -0.2427 1.4029 -0.8981 1.0073 0.7573 0.4029
--------	--

>> A./B	1.0000 0.5000 0.3333 0.8000 ans = 2.0000 1.0000 1.1250 3.5000 3.0000 8.0000 1.0000 0.3750 1.2500 0.2857 2.6667 1.0000
---------	--

If you want to see more detail, use 'format long'. To hide detail, use 'format short'.

>> format long >> A./B	ans = 1.0000000000000000 0.5000000000000000 0.3333333333333333 0.8000000000000000 2.0000000000000000 1.0000000000000000 0.1250000000000000 3.5000000000000000 3.0000000000000000 8.0000000000000000 1.0000000000000000 0.3750000000000000 1.2500000000000000 0.28571428571429 2.666666666666667 1.0000000000000000
---------------------------	--

Transpose operators: ' is conjugate-transpose and .' is transpose.

>> format short >> C'	ans = 1.0000 -2.0000i 4.0000 -2.0000i 9.0000 -8.0000i 0 -1.0000i 2.0000 -3.0000i 2.0000 -1.0000i 8.0000 -9.0000i 1.0000 3.0000 -4.0000i 1.0000 -7.0000i 9.0000 -3.0000i 0 -1.0000i 4.0000 -1.0000i 7.0000 -4.0000i 3.0000 -9.0000i 2.0000
--------------------------	---

>> C.'	ans = 1.0000 +2.0000i 4.0000 +2.0000i 9.0000 +8.0000i 0 +1.0000i 2.0000 +3.0000i 2.0000 +1.0000i 8.0000 +9.0000i 1.0000 3.0000 +4.0000i 1.0000 +7.0000i 9.0000 +3.0000i 0 +1.0000i 4.0000 +1.0000i 7.0000 +4.0000i 3.0000 +9.0000i 2.0000
--------	---

Matrix powers: ^ raise a matrix to a power; .^ raise each element to a power.

>> A^2	ans = 56 38 64 51 56 34 79 75 137 112 140 137 115 90 137 94
--------	---

>> A.^2	ans = 1 4 9 16 16 4 1 49 81 64 81 9 25 4 64 36
---------	--

Type 'help ops' for more help on operators.

2.3 Basic Operations

Matrix creation functions include: 'ones', 'zeros', 'rand', and 'randn'.

>> ones(3,5)	ans = 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
--------------	--

>> rand(3,5)	ans = 0.9501 0.4860 0.4565 0.4447 0.9218 0.2311 0.8913 0.0185 0.6154 0.7382 0.6068 0.7621 0.8214 0.7919 0.1763
--------------	---

Function 'sum' operates along rows unless you specify otherwise.

>> sum(A)	ans = 19 14 21 20
-----------	----------------------

>> sum(A,2)	ans = 10 14 29 21
-------------	-------------------------------

Use the : operator to select a sub-matrix.

>> sum(A(1:3,2:4))	ans = 12 13 14
--------------------	-------------------

Function 'prod' and 'mean' work similarly.

>> prod(A)	ans = 180 64 216 504
------------	-------------------------

>> mean(A)	ans = 4.7500 3.5000 5.2500 5.0000
------------	--------------------------------------

Function 'max' works along rows.

>> max(A)	ans = 9 8 9 7
-----------	------------------

Another way to use max is element by element.

>> max(A,5)	ans = 5 5 5 5 5 5 5 7 9 8 9 5 5 5 8 6
-------------	---

max can be used along columns as follows.

>> max(A,[],2)	ans = 4 7 9 8
----------------	---------------------------

Function 'diag' will extract the diagonal elements of a matrix. Function 'det' will compute the determinate.

>> diag(A)	ans = 1 2 9 6
------------	---------------------------

>> det(A)	ans = -824
-----------	---------------

2.4 Vectors and Matrix Manipulation

An advantage of Matlab is the ease of manipulations of vectors. For example, we can create a second vector $z = 15*t$ by typing:

>> t=5:0.01:t; >> z = 15*t;	
--------------------------------	--

Another convenience is the use of built in functions. To create a vector x with elements $x(i) = \sin(t(i))$, type the following at the prompt:

>> x = sin(t);	
----------------	--

or

>> x = 15*sin(2*t);	
---------------------	--

Play away with all the functions that you wanted to know about and see what they look like! In fact Matlab also lets you hear what these functions sound like! However, sound is usually disabled on CEC machines. You could try listening to your favorite functions at home if you have Matlab installed on your personal computer.

A host of trig functions are included (sin, cos, tan, etc.). Several exponential functions are included (exp, log, log10, sqrt, etc.)

>> sin(A)	ans = 0.8415 0.9093 0.1411 -0.7568 -0.7568 0.9093 0.8415 0.6570 0.4121 0.9894 0.4121 0.1411 -0.9589 0.9093 0.9894 -0.2794
-----------	---

>> log(A)	ans = 0 0.6931 1.0986 1.3863 1.3863 0.6931 0 1.9459 2.1972 2.0794 2.1972 1.0986 1.6094 0.6931 2.0794 1.7918
-----------	---

Several functions for complex numbers are available (abs, phase, imag, real, etc.)

>> phase(C)	ans = 1.1071 0.9828 0.9273 0.2450 0.4636 0.4636 1.4289 0.5191 0.7266 0.8442 0.3218 1.2490 1.5708 0 1.5708 0
-------------	---

Rounding and remainder functions are available (floor, ceil, round, mod, etc.)

>> mod(A,3)	ans = 1 2 0 1 1 2 1 1 0 2 0 0 2 2 2 0
-------------	---

Use 'help elfun' for help on elementary functions.

Use 'help datafun' for help on convolution and various discrete Fourier transform functions.

Use 'help' and 'lookfor' for more general help.
Complete documentation available online at <http://www.mathworks.com/support/>

3.0 Matlab Toolboxes

Matlab offers a number of toolboxes, collections of M-files, that extend Matlab's functionality into specific disciplines. Following is a list of some of the toolboxes available. For a complete list, go the Mathworks Inc. website.

- Signal Processing
- Control System
- Image Processing
- Communications
- Data Acquisition
- Database,
- Datafeed
- Filter Design
- Financial Derivatives,
- Fuzzy Logic
- Instrument Control
- Mapping
- Model Predictive Control
- μ -Analysis and Synthesis
- Neural Network, Optimization,
- Partial Differential Equation
- Robust Control
- Spline
- Statistics
- Symbolic Math
- System Identification
- Wavelet

4.0 Programming Notes

Matlab allows you to do low-level programming interactively.

4.1 If and For Constructs in Matlab

Lets create a signal $u[t]$, the unit step function ($u[t] = 0$ if $t < 0$, $u[t] = 1$ otherwise). To familiarize yourself with the FOR and IF constructs in Matlab, use the built-in help routine:

```
>> help if
>> help for
```

In general, typing "help <command name>" gives an explanation of command functions and syntax. Typing help with no command name gives a list of topics by which one can search for a desired command.

Lets create $u[t]$:

```
>> for i = 1:length(t)
    if (t(i) < 0)
        u(i) = 0;
    else
        u(i) = 1;
    end
end
```

Note several points in the lines above:

Matlab indexes vectors and matrices starting at 1 unlike C which starts at 0 for arrays and matrices.

IF commands require a terminating end (endif is pseudocode)

FOR commands require a terminating end (endfor in pseudocode)

By default, the FOR command increases i by 1 after each iteration. To change this increment, we use the double colon notation used to create vectors before. For example, to loop i from 1 to 10 in 0.5 increments we can say:

```
>> for i = 1:0.5:10
```

In creating the signal $z[t] = 15x[t]$ we multiplied the vector $x[t]$ by a scalar. Lets see how multiplication of vectors works by creating the signal $y[t] = z[t]u[t]$. What we really want to do is multiply each element of the vector $z[t]$ by the corresponding element of the vector $u[t]$, i.e. the same value of t . The Matlab command to do this is `.*`, as opposed to scalar multiplication which is `*`.

```
>> y = z.*u;
```

Elementwise vector division is performed by the operation `./`.

4.2 Conditional Execution

Some operators return logical arrays.

```
>> A>5
ans =
  0  0  0  0
  0  0  0  1
  1  1  1  0
  0  0  1  1
```

These can be used like ordinary arrays.

```
>> sum(sum(A>5))
ans =
  6
```

Bit-wise operators do what you would expect.

```
>> (A>5)&(A<8)
ans =
  0  0  0  0
  0  0  0  1
  0  0  0  0
  0  0  0  1
```

Other bit-wise operators are `|` (or) and `~` (not). Construct 'if expr, statement1, else, statement2, end' executes statement 1 if all elements of expr are non-zero, otherwise statement2 is executed.

```
>> if (A>5)
    disp('All elements are greater
than five');
else
    disp('Some elements are not
greater than five');
end
```

Some elements are not greater than five.

Construct 'for var=expr, statement1, end' assigns columns of expr to var in turn.

<pre>>> for v=1:5 disp(v); end</pre>	<pre>1 2 3 4 5</pre>
--	----------------------

Also available are 'while' and 'switch' constructs. See 'help lang' for more language constructs.

4.3 Character and String Manipulation

Strings are row vectors of characters and can be denoted by ' '.

<pre>>> mystr='Hello, Dave.'</pre>	<pre>mystr = Hello, Dave.</pre>
--	---------------------------------

They can be concatenated, subscripted, etc. just like other arrays.

<pre>>> [mystr, ' How are you?']</pre>	<pre>ans = Hello, Dave. How are you?</pre>
--	--

<pre>>> mystr(8)</pre>	<pre>ans = D</pre>
------------------------------	--------------------

The function 'num2str' will convert numeric arrays to text arrays formatted nicely.

<pre>>> mystr=num2str(A)</pre>	<pre>mystr = 1 2 3 4 4 2 1 7 9 8 9 3 5 2 8 6</pre>
--------------------------------------	--

Function 'disp' will display any array on the command window. Function 'sprintf' will allow more careful formatting and works very much as in C. For more on strings, see 'help strings' or 'help strfun'.

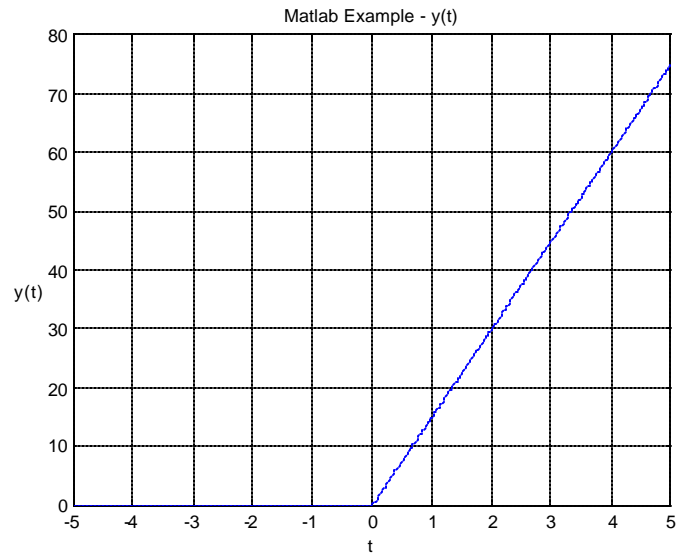
5.0 Graphing in Matlab

We have created several signals. To list them all use the "whos" command:

<pre>>> t = -5:0.01:5; >> z=15*t; >> x=sin(t); >> for i = 1:length(t) if (t(i) < 0) u(i) = 0; else u(i) = 1; end end >> y = z.*u; >> whos</pre>	<pre>Name Size Bytes Class t 1x1001 8008 double array u 1x1001 8008 double array x 1x1001 8008 double array y 1x1001 8008 double array z 1x1001 8008 double array Grand total is 5005 elements using 40040 bytes</pre>
--	--

You should see, t, u, x, y, z. Let's plot the signal u[t] (Y-axis) versus t (X-axis).

```
>> plot(t,y)
>> title('Matlab Example - y(t)')
>> xlabel('t')
>> ylabel('y(t)')
>> grid
```



The plot of $y(t)$ should replace the previous plot in the Figure 1 window on the screen. For more information on how the commands `title`, `xlabel`, `ylabel`, and `grid` work, use `help` as before.

5.2 Figures and Plotting

Create a figure window with 'figure.'

```
>> myfig=figure;
>> myfig2=figure;

Activate a figure window with
'figure'.

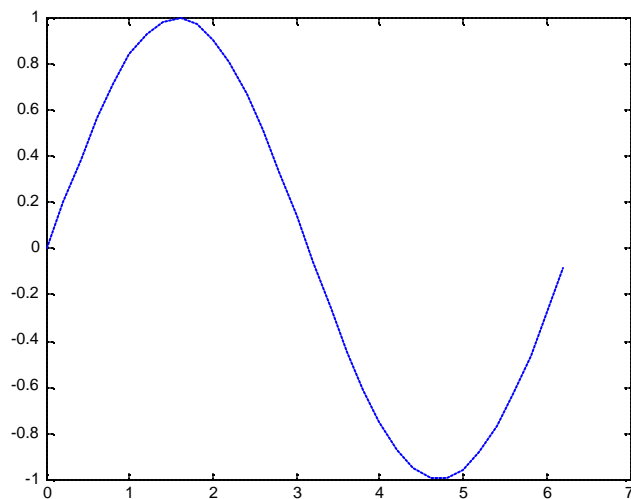
>> figure(myfig)

Close a figure window with
'close'...

>> close(myfig2)

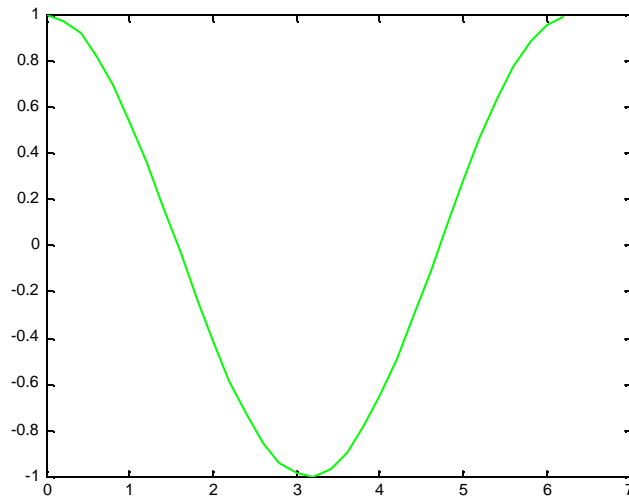
Make a plot in the current figure
window with 'plot'...

>> theta=0:0.2:2*pi;
>> plot(theta,sin(theta))
```



Simply plotting again will replace the old plot.

```
>> plot(theta,cos(theta),'g')
```

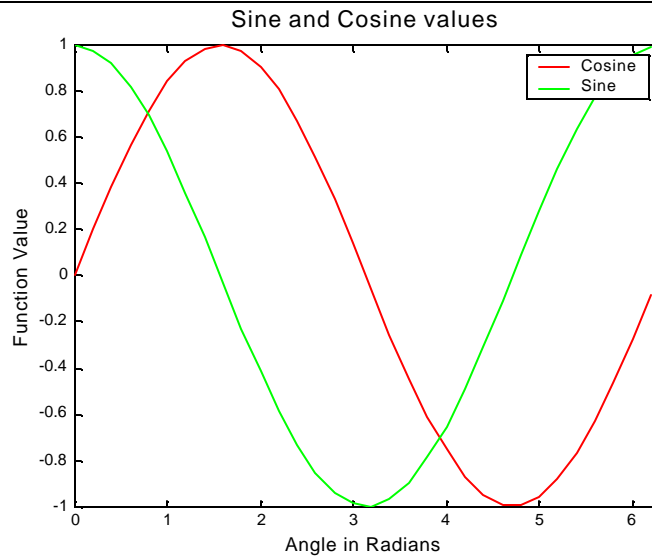


To overlay a plot, use 'hold on'.
To replace again, use 'hold off'.

```
>> hold on  
>> plot(theta,sin(theta),'r')
```

We can make the plot more pleasing by removing the blank space on the right.
>> axis([0,2*pi,-1,1])
One should ALWAYS carefully label a plot...

```
>> title('Sine and Cosine  
values','FontSize',14)  
>> legend(char({'Cosine','Sine'}))  
>> xlabel('Angle in  
Radians','FontSize',12)  
>> ylabel('Function  
Value','FontSize',12)
```



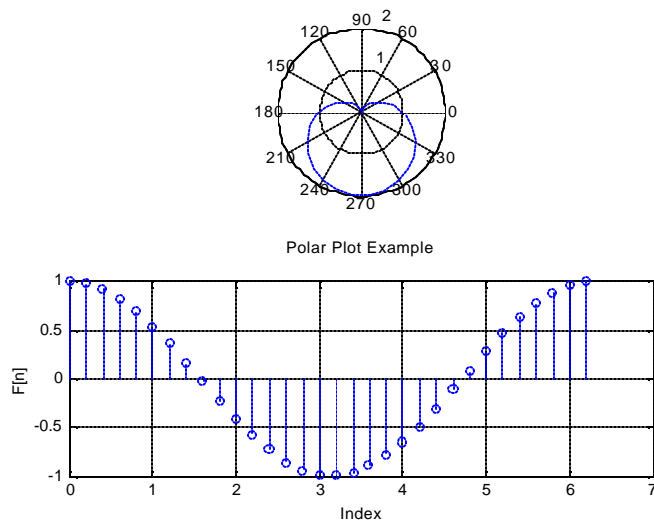
We can put more than one plot on a figure with 'subplot'. Here we make a polar plot and a stem plot.

```

>> myfig2=figure;
>> subplot(2,1,1);
>> title('Advanced Plotting')
>> polar(theta,1-sin(theta));
>> xlabel('Polar Plot Example')

>> subplot(2,1,2);
>> stem(theta,cos(theta))
>> xlabel('Index')
>> ylabel('F[n]')
>> grid

```



6.0 Printing a Plot

To print a plot, there are two common options. First we may use the print command to send the current figure to the default printer. To do this type:

```
>> print
```

The second option is to create a postscript file of the current figure. To do this type

```
>> print <filename>
```

The file, <filename>.ps will be created in the current directory. To print the postscript file, at a Unix prompt type:

```
clarion.cec> lpr -P<printer name> <filename>
```

On PC's you can select print from the figure menu. To import figures into MSWord do the following.

Open MSWord. Go back to Matlab and select 'copy figure' from the edit menu of the figure. Switch to MSWord and paste. Once the figure is pasted in MSWord, right click on it and select 'format picture.' Press the 'position' tab and make sure 'float over text' is unchecked. Click 'ok' to accept the option. You can resize the figure as you wish.

7.0 Saving work in .mat Files

A convenient feature of MATLAB is the ability to save work from one session and load in a later session (for those problems with which you lose patience for a short time). To practice this, type the following:

```
>> save output t x u z y
```

This command creates output.mat in the current directory storing the vectors t, x, u, z, y. Now exit Matlab by typing quit or exit at the Matlab prompt. Start a new session and type 'who' at the Matlab command. There should be no variables stored. Now type the following to retrieve the variables from output.mat.

```

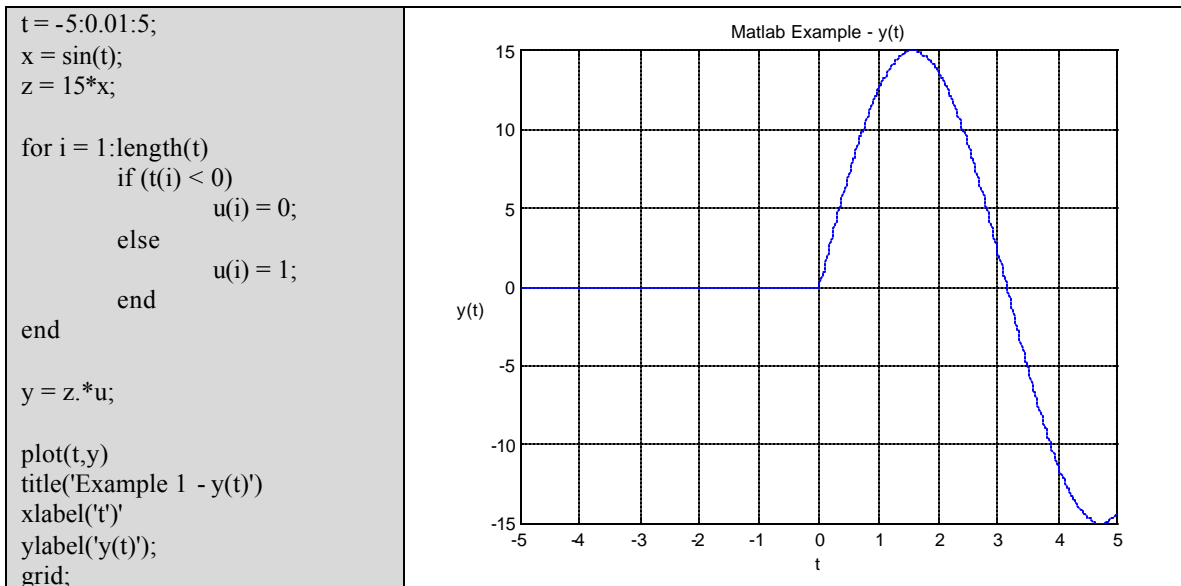
>> load output
>> who

```

For more information on the load and save commands, use the help command as before.

8.0 Creating .m Files (Scripts)

Matlab supports a limited programming environment. To take advantage of this, create a file containing a series of Matlab commands using your favourite Unix editor (vi, emacs, pico, etc.). After exiting Matlab, create a file named "example.m" containing the following:



Now start a new Matlab session, typing "matlab" at the Unix prompt. At the matlab prompt, type:

```
>> example
```

The previous plot of $y[t]$ should appear as before. In general, scripts may be created with any sequence of Matlab commands. The scripts must have the suffix ".m" and be in the current directory to be accessible within Matlab. Then type the prefix of the filename (the filename without the .m) inside Matlab to execute the script.

8.1 Scripts and Functions

Scripts execute as if typed from the command line - same context, no parameters, no return values.

Functions execute in a separate context, accept parameters, and (optionally) return values. Use Matlab's command 'edit' or your favorite text editor to create & use the extension '.m'.

```
>> edit
```

Insert comments with the '%' character. Comment lines beginning in the first line of a script file or immediately after the function declaration are presented when you request 'help' for the m-file.

Function files begin with the syntax:

```
function [ret1, ret2, ....] = myfun(param1,param2,....)
```

Parameters are passed by reference unless they are modified within the function. The type command will show you what is inside a '.m' file.

>> type myfun	<pre>function [r_sum,r_diff] = myfun(p_a,p_b) % % This function returns the sum and difference % between the two parameters. r_sum = p_a + p_b; r_diff = p_a - p_b;</pre>
---------------	--

>> help myfun	<pre>This function returns the sum and difference between the two parameters.</pre>
---------------	---

Invoke a function as you would expect.

>> [thesum,thediff] = myfun(A,B)	<pre>thesum = 2 6 12 9 6 4 9 9 12 9 18 11 9 9 11 12 thediff = 0 -2 -6 -1 2 0 -7 5 6 7 0 -5 1 -5 5 0</pre>
----------------------------------	--

Scripts and functions must be in the working directory or on the Matlab search path. Use 'pwd' to see your working directory and 'cd' to change it. Function 'dir' does what you'd expect.

>> pwd	<pre>ans = h:\dos</pre>
--------	-------------------------

Use 'path' to see the search path, 'addpath' to add directories, and 'rmpath' to remove directories.

9.0 Further Reading and Advanced Topics

Matlab provides file I/O functions similar to C - see 'help iofun'.

Matlab provides support for multi-dimensional arrays

Matlab provides structured variables and objects similar to C - see 'help datatypes'.

Matlab provides cell arrays which are arrays of other arrays - see 'help datatypes'.

10.0 References

[1] The Student Edition of Matlab manual, The Mathworks Inc.

[2] Computer Based Exercises Using Matlab, Oppenheim.