MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science

6.003: Signals and Systems — Fall Term 1994

Pico-course in MATLAB for 6.003

(by Deron Jackson)

Starting MATLAB

Several 6.003 specific functions have been written for MATLAB. In order for these functions to be accessible, it is important that you start MATLAB as follows:

```
athena% add 6.003
athena% /mit/6.003/startup
```

MATLAB will open its own window and the soon to be familiar ">>" prompt will appear.

Variables: scalars, vectors and matrices

MATLAB is designed to perform numerical manipulations. All numbers are stored in variables that make up what is called the "workspace". A variable can be a scalar, a vector, or a matrix and it may be real or complex. For example:

Real scalar	`>> x = 10.5`
Complex scalar	`>> x = 3 + 4*i`
Row Vector	`>> x = [1 2 3]`
Column Vector	`>> x = [1;2;3]`
Matrix	`>> x = [1 2 3;4 5 6;7 8 9]`

Note that the semicolon is used to indicate a new row. Both the variables "`i`" and "`j`" are predefined as the complex number `i = j` $= \sqrt{-1}$. It is important not to accidentally redefine these variables. Complex numbers are easily manipulated using several built-in functions. For example:

Complex scalar	`>> x = 3 + 4*i`		
Real part of x	`>> real(x)`	$\Rightarrow$	3
Imaginary part of x	`>> imag(x)`	$\Rightarrow$	4
Magnitude of x	`>> abs(x)`	$\Rightarrow$	5
Phase of x (radians)	`>> angle(x)`	$\Rightarrow$	0.9273

Basic Math

Basic arithmetical operations such as addition, subtraction, multiplication, etc. are defined using the usual symbols (+, -, *, /, ^). All of these are treated as *matrix* operations unless one of the

variables is a scalar. For example:

```
>> x = [1 2;3 4]
```
$\Rightarrow \quad x = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ Define x

```
>> y = x*2
```
$\Rightarrow \quad y = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times 2 = \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix}$ Scalar Mult.

```
>> y = x*[2 2;2 2]
```
$\Rightarrow \quad y = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix} = \begin{bmatrix} 6 & 6 \\ 14 & 14 \end{bmatrix}$ Matrix Mult.

An error will occur when the matrix dimensions are improper. For example the two vectors [1 2] and [2 3] cannot be multiplied as matrices.

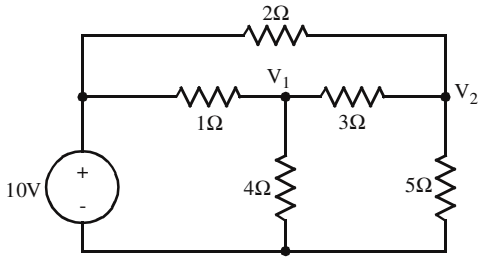
```
>> y = [1 2] * [1 2]
??? Error using ==> *
Inner matrix dimensions must agree.
```

However, MATLAB will multiply their elements if the operation is proceeded by a ".". This feature will prove valuable when manipulating vectors. For example:

```
>> y = [1 2] .* [1 2]
```
$\Rightarrow \quad y = \begin{bmatrix} 1 & 4 \end{bmatrix}$

Solving Equations

MATLAB makes it easy to solve simultaneous equations as long as they are represented in matrix form. For the sample circuit below it is possible to write two simultaneous equations in terms of V_1 and V_2. The two equations represent KCL performed at the nodes V_1 and V_2.



Eq. #1: $\Rightarrow \quad 10 - V_1 + \frac{V_2 - V_1}{3} - \frac{V_1}{4} = 0 \quad \Rightarrow \quad -\frac{19}{12}V_1 + \frac{1}{3}V_2 = -10$

Eq. #2: $\Rightarrow \quad \frac{V_1 - V_2}{3} - \frac{V_2}{5} + \frac{10 - V_2}{2} = 0 \quad \Rightarrow \quad \frac{1}{3}V_1 - \frac{31}{30}V_2 = -5$

The two equations can be rewritten in matrix form.

$$\begin{bmatrix} -19/12 & 1/3 \\ 1/3 & -31/30 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} -10 \\ -5 \end{bmatrix}$$

Using MATLAB it is possible to solve for V_1 and V_2 by finding the matrix inverse.

```
>> A = [-19/12 1/3;1/3 -31/30];
>> b = [-10; -5];
>> v = inv(A)*b
v =
    7.8689    ⇒    V1
    7.3770    ⇒    V2
```

Working with Polynomials

Polynomial expressions show up everywhere in 6.003. Luckily, MATLAB is quite adept at handling them. A polynomial is represented in MATLAB by a vector containing its coefficients. For example:

$s^3 - 5s^2 + 8s - 6$ ⇒ `>> p1 = [1 -5 8 6]`

The three roots, s = 3, – 1 + j, – 1 – j , can be found using the "roots" command.

```
>> roots(p1)
ans =
   3.0000
   1.0000 + 1.0000i
   1.0000 - 1.0000i
```

The "polyval" command evaluates a polynomial at a value "s" or a vector of values. For example:

```
>> polyval(p1,1)              ⇒    -2
>> polyval(p1,[1 2 3])        ⇒    [-2 -2 0]
```

Other useful polynomial commands:

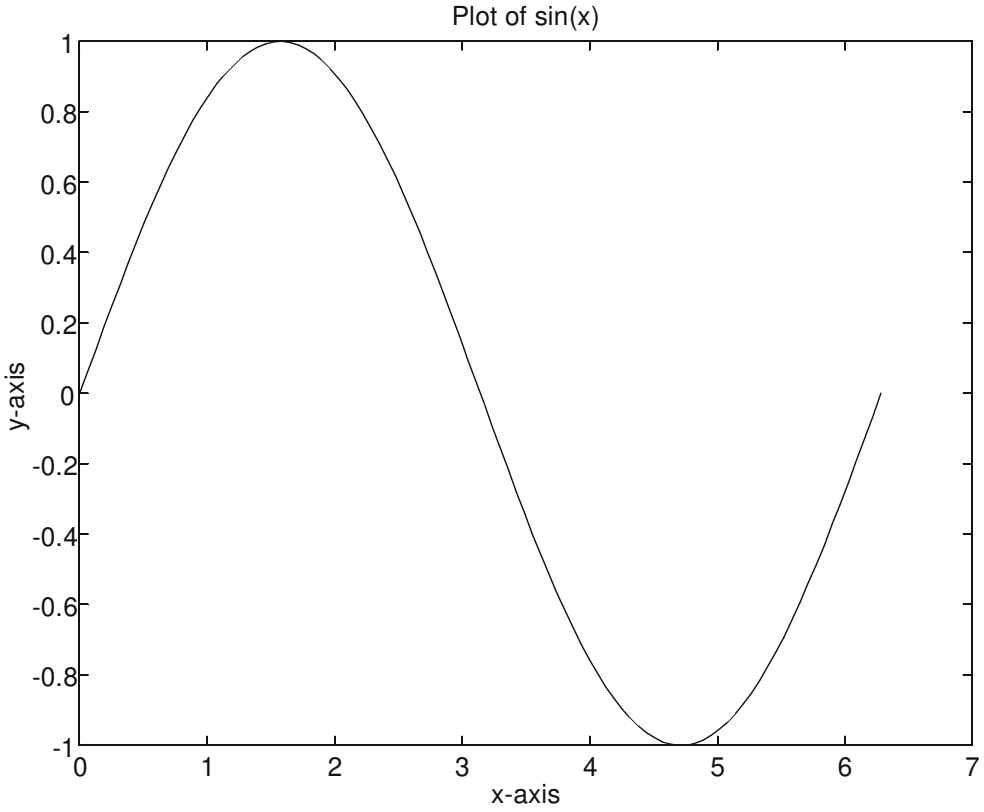
```
>> conv(p1, p2)           Compute the product of two polynomials.
>> poly2str(p1,'s')       Display polynomial as an equation.
>> poly(zero_locations)   Compute polynomial given zeros.
```

Graphics/Plotting

MATLAB has extensive graphics capabilities which can be used to graph just about anything. The most commonly used command is "plot(x,y)". The plot command draws a line between points listed in the vectors "x" and "y". For example:

```
>> x = linspace(0,2*pi,100);
>> y = sin(x);
>> plot(x,y)
>> title('Plot of sin(x)')
>> xlabel('x-axis')
>> ylabel('y-axis')
```

Generate a 100 point vector from $0 \rightarrow 2\pi$
Find sin(x)
Plot the sin(x) versus x
Add a title
Add an x-axis label
Add a y-axis label



Adding additional parameters to the plot command (i.e. `plot(x1,y1,x2,y2,....)`) allows for multiple plots on the same axis. Each coordinate pair is automatically plotted in a different color. Dotted, dashed, and other linestyles are possible as well. Type "help plot" for details.

Here are several additional commands that you should look into.

```
>> loglog(x,y)
>> semilogx(x,y)
>> semilogy(x,y)
>> axis([xmin xmax ymin ymax])
>> subplot(x,n,m)
>> figure
>> hold
```

Same as plot but uses log x and y axes.
Same as plot but uses log x-axis only.
Same as plot but uses log y-axis only.
Force a particular axis scaling.
Partition the window for multiple plots.
Generate a new figure window.
Hold the current plot. Prevents the screen from clearing before the next plot.

Printing

Printing the current figure window is simple. Typing:

```
>> print -Pprinter_name
```

sends the screen directly to the named printer. By default the figure window will occupy the middle portion of a full page. Typing:

```
>> print -dps filename.ps
```

saves the figure window to a file for later printing using "lpr".

The orientation and size of the printout can be adjusted using the following commands:
Use the MATLAB help facility for more information on the following commands.

```
>> orient portrait or tall or landscape
>> subplot(x,n,m)
```

Loading/Saving Work

There are two ways to save your MATLAB work. The first is to use the "save" and "load" commands. The "save" commands stores all the variables currently in your "workspace". For example, if you defined variables "x", "y", and "z". Their values are saved by typing:

```
>> save filename
```

At a later MATLAB session the values may be recalled by typing:

```
>> load filename
```

It is also possible to keep a diary of the screen input/output as you use MATLAB. This is very useful when including MATLAB work in a written report. Typing

```
>> diary filename
```

starts recording all text to the named file. Typing

```
>> diary off
```

ends the process and closes the file.

Scripts and Functions: ".m" Files

Scripts and functions are powerful features in MATLAB. MATLAB scripts or functions are referred to as ".m" files (dot-m-files). Both are nothing more than text files named "filename.m".

Typing the filename at the ">>" prompt executes the respective script or function. You may create or edit your own files using emacs. This can be done from the "athena%" prompt by typing

```
athena% emacs filename.m &
```

or from within MATLAB by typing

```
>> !emacs filename.m &
```

Either command will open a new emacs window in which to edit the text file.

A script is simply a file containing a set of commands that might otherwise be executed at the ">>" prompt. Compiling them in a script file prevents repeatedly typing the same commands and provides a convenient way to save your work.

On the other hand, a function can accept and return a series of variable parameters. Variables not directly passed to the function are independent of any already defined in the workspace. Therefore a function which uses a variable "x" will not alter the value of "x" if it is already defined in the workspace.

Below is a sample function "para_res.m". This function takes two resistances as its inputs and returns the equivalent resistance of their parallel combination.

```
function equiv = para_res(r1, r2)
% equiv = para_res(r1, r2)
%
% equiv = Equivalent resistance of r1 an r2 in parallel.
% r1    = Resistance r1.
% r2    = Resistance r2.

% equiv=(r1*r2)/(r1+r2);
```

The function is executed at the ">>" prompt as follows.

```
>> para_res(2,4)
ans =
    1.3333
```

With the addition of a series of programming constructs it is possible to turn a MATLAB function into a full-fledged program. Some of the available constructs are:

FOR loops:

```
for i=1:4
     command
     command
end
```

WHILE loops:

```
while i<100
     command
     command
end
```

IF then ELSE conditions:

```
if i<100
     command
else
     command
end
```

Getting On-Line Help

There are several ways to get on-line help for MATLAB. Within MATLAB, help on any command can be found by typing:

```
>> help command-name
```

For example:

```
» help roots
 ROOTS Find polynomial roots.
   ROOTS(C) computes the roots of the polynomial whose coefficients
   are the elements of the vector C. If C has N+1 components,
   the polynomial is C(1)*X^N + ... + C(N)*X + C(N+1).

   See also POLY.
```

If you cannot remember the name of the command, type the following.

```
>> lookfor keyword
```

This searches for any commands containing the given keyword. For example:

```
» lookfor roots
ROOTS Find polynomial roots.
RLOCFIND Find the root locus gains for a given set of roots.
FILTFUN Returns frequency response and roots for DFILDEMO.
FILTFUN2 Return frequency response norm and roots for DFILDEMO.
ROTVAR computes the standard deviations of roots to polynomials
```

More general help is available from the athena prompt. Try the following:

```
athena% olc answers
```

or

```
athena% help @matlab:main_menu
```