



Getting Started

MatLab runs on Windows and HP workstations. To start MatLab in Unix, enter: **matlab**. In Windows, run Winstall to install the program and then double click on its icon. This brings up a command window where you may enter your MatLab commands.

MatLab Basics

The basic structure in MatLab is the matrix: matrix algebra rules apply at all times. *MatLab is case sensitive*. Therefore, **a** is different from **A**.

To declare a matrix, enter the numbers in square brackets separating row elements by spaces and rows by semicolons.

```
mat1 = [1 2; 3 4];
```

Defines the 2 by 2 matrix named **mat1**. A semicolon after any command asks that the immediate result not be displayed. Use **mat1(1,2)** to specify the element in the first row and second column (2), and **mat1(:,1)** to specify the elements in the first column (1,3). Some basic operations on matrices **a** and **b** are:

Addition	a+b
Subtraction	a-b
Multiplication	a*b
Inverse	inv(a)
Determinant	det(a)
Division	a/b equivalent to a*inv(b) a\b equivalent to inv(a)*b
Identity matrix (n x n)	eye(n)
Eigenvalues	eig(a)

To assign the result of any operation you must assign it a name. The assignment statement is: *name = operation*. Some basic operations on scalar **a** are:

Absolute value	abs(a)
Trigonometric funcs.	sin(a); cos(a); tan(a)
Exponential	exp(a)
Natural logarithm	log(a)
Common logarithm	log10(a)
Square root	sqrt(a)

Solving Equations

The roots function in MatLab can find the roots of a polynomial if you form a coefficient matrix. For example, to find the roots of the polynomial $f(x)=7x^3+3x+10.5$, enter

```
f=[7 0 3 10.5];  
roots(f)
```

Systems of equations solved in matrix form. For example, to solve:

$$\begin{aligned} 6x + 5y + 10z &= 10 \\ 3x + 14y + 1.8z &= 5 \end{aligned}$$

$$8x + y + z = 3.86$$

enter:

```
a=[6 5 10; 3 14 1.8; 8 1 1];  
b=[10; 5; 3.86];  
x=inv(a)*b
```

The matrix **x** contains the solution. MatLab uses Gaussian elimination to find a solution and checks to see if the solution is correct. If errors are likely, MatLab prints a warning. Check the solution by entering **a*x**.

Graphing with MatLab

To construct 2D plots, a matrix of independent variables must be declared, a matrix of dependent variables, and, optionally, the plot character. For example,

Linear plot	plot(x, y, '+')
Semilogarithmic plot	semilogx(x, y, '.')
Log-log scale plot	loglog(x, y, '-')

The third parameter in the 2D plot functions is the plot symbol used. You may edit the graph with the following commands:

Title	title('<text>')
Label x-axis	xlabel('<text>')
Label y-axis	ylabel('<text>')
Grid	grid

Three-dimensional graphs may also be generated. The data must be arranged in an $m \times n$ array, where m and n are the number of x and y grid points respectively (the array value $A(i,j)$ is the z value at $x(i),y(j)$). Examples of commands to construct such plots are:

Mesh surface plot	mesh(x,y,z)
Mesh with contours	meshc(x,y,z)
Contour plot	contour(x,y,z)
Shaded surface plot	surf(x,y,z)

Printing from MatLab

Click on the graph window to be printed. Then enter **print** (or select Print... from the File menu) to print it on the default Printer. To save the graph to a postscript file, enter **print file_name.ps**. In Unix, you may then use **lpcae** from the Unix prompt to print the file, or you may import the file as a graph into a document.

Saving the Current Workspace

Enter **save file_name** to save the values in the active variables in your MatLab session in the file **file_name.mat**. This saves just the variable names and their values. To retrieve saved values, enter **load file_name**. The default file name in either case is **matlab.mat**. If you enter **diary file_name**, a copy of all subsequent terminal input and most of the resulting output is written to the named file. You must direct

Continued on Reverse

the diary to a file by including the filename in your statement. This function can be turned on and off using **diary on** and **diary off** respectively.

Using M-files: Scripts and Functions

To add a new function to MatLab, place its definition in a file named *function_name.m*. The first line must contain the syntax definition of the new function. Comment lines start with %. For example:

```
function y = mean(x)
% For vectors, MEAN(x) is the mean value
% of the elements in vector x. For
% matrices, MEAN(x) is a row vector con-
% taining the mean value of each column.
[m,n] = size(x);
if m == 1
    m = n;
end
y = sum(x)/m;
```

To add a new command to MatLab, place its definition in a script file named *command_name.m*. A script file contains a sequence of MatLab statements. When you enter the command name, MatLab executes the commands in the file. For example:

```
plot(x,ureal,'*',x,vreal,'+')
grid on
text(1,.9,'* Numerical')
text(1,.8,'+ Exact')
print
```

These can be created using a text editor such as vi or ex.

In Windows, MatLab does not recognize changes to these files unless you exit and restart MatLab or store the files in the C:\temp directory. To do your work in C:\temp, select Set Path... from the File menu and enter C:\temp as the Current Directory. When you are done, copy what you want to keep to your home directory.

MatLab Help

MatLab has a good on-line help system, if you have questions, enter **help** to view a list of MatLab commands. Then enter **help function_name**. There are general manuals available for checkout at the Consultant's Desk in room 170., as well as manuals for specific MatLab toolboxes.

The Control Systems Toolbox

The Control Systems Toolbox available with MatLab on the CAE Unix network provides many functions useful for control system design and analysis.

Transfer Functions

Plant descriptions may be entered into MatLab in transfer function form. For example: $H(s) = \frac{0.2s^2 + 0.3s + 1}{(s^2 + 0.4s + 1)(s + 0.5)}$

MatLab handles the numerator and denominator coefficients in descending powers of s. This can be accomplished by entering **num = [0.2 0.3 1]**, **den1 = [1 0.4 1]**, and **den2 = [1 0.5]** in three steps. To calculate the denominator as the product of the two terms using convolution (same as multiplying the two polynomials together), enter:

```
den = conv(den1, den2)
```

Natural Frequencies and Damping Factors

The **damp** function provides information about the natural frequencies and damping factors of plant poles. The syntax is **[wn, z] = damp(den)**. This results in **wn** containing the natural frequencies and **z** the damping factors.

Unit Step and Impulse Response

For systems described by transfer functions, the step and impulse functions plot the systems time response to a unit step or unit impulse input. Time increments are calculated automatically or you can specify a time matrix as **t = begin_time:time_inc:end**. For the first option, enter **step(num,den)** or for the second, enter **step(num,den,t)**. The output of each of these commands can be stored as a matrix by assigning it a name such as

```
y = step(num,den)
```

Root Locus Diagrams

The rlocus function calculates root locus values. MatLab expects the function to be of the form: $H(s) = 1 + k \frac{num(s)}{den(s)}$

where k is defined as the system gain. The range of gains may be defined by the user, analogous to the time matrix above. If so, the command **rlocus(num, den, k)** can be entered. If the command **rlocus(num, den)** is used instead, MatLab generates gain values.

Bode Plots and Nichols Charts

The bode function calculates magnitude and phase values for a range of frequencies. These values can be plotted in the form of a Nichols chart or Bode Plot. To calculate equally spaced points in logspace enter

```
w = logspace(d1, d2, int)
```

where *d1* is the exponent of the low decade, *d2* is the high decade exponent, and *int* is the integer number of points desired. For example, enter:

```
w = logspace(-1, 1, 50);
[mag, phase]=bode(num,den,w);
```

The magnitude and phase can both be plotted simultaneously using the subplot command. Enter

```
subplot(211), loglog(w,mag),
title('Magnitude Response'),
subplot(212), semilogx(w,phase),
title('Phase Response').
```

To plot a Nichols chart (magnitude vs. phase) with axis labels:

Is this document clear? Is it missing crucial information?
Please mail comments to the handout editor, CAE,
1410 Engineering Drive, or to: editor@engr.wisc.edu

```
plot(mag, phase), xlabel('Magnitude'),  
      ylabel('Phase').
```

Nyquist Plots

The nyquist function can be used to calculate values for a nyquist plot. As with the bode function, the values of w must be logarithmically spaced. To create a nyquist plot, enter

```
nyquist(num, den, w)
```