Introduction to MATLAB

What is MATLAB?

MATLAB ("MATrix LABoratory") is a tool for numerical computation and visualization. The basic data element is a matrix, so if you need a program that manipulates array-based data it is generally fast to write and run in MATLAB (unless you have very large arrays or lots of computations, in which case you're better off using C or Fortran).

Getting started

MATLAB is available on department machines. You can also download MATLAB for your personal machine from http://software.caltech.edu.

Type "matlab" at the Unix prompt to start. This will open the MATLAB desktop, which includes interactive menus and windows in addition to the command window. You can also start a command prompt-only version of MATLAB (useful if you are logged in remotely) by typing "matlab –nodesktop".

Using MATLAB

The best way to learn to use MATLAB is to sit down and try to use it. In this handout are a few examples of basic MATLAB operations, but after you've gone through this tutorial you will probably want to learn more. Check out the "Other Resources" listed at the end of this handout.

The Beginning

When you start MATLAB, the command prompt ">>" appears. You will tell MATLAB what to do by typing commands at the prompt.

Creating matrices

The basic data element in MATLAB is a matrix. A scalar in MATLAB is a 1x1 matrix, and a vector is a 1xn (or nx1) matrix.

For example, create a 3x3 matrix A that has 1's in the first row, 2's in the second row, and 3's in the third row:

```
>> A = [1 1 1; 2 2 2; 3 3 3]
```

The semicolon is used here to separate rows in the matrix. MATLAB gives you:

```
A =

     1     1     1
     2     2     2
     3     3     3
```

If you don't want MATLAB to display the result of a command, put a semicolon at the end:

```
>> A = [1 1 1; 2 2 2; 3 3 3];
```

Matrix A has been created but MATLAB doesn't display it. The semicolon is necessary when you're running long scripts and don't want everything written out to the screen!

Suppose you want to access a particular element of matrix A:

```
>> A(1,2)

ans =

    1
```

Suppose you want to access a particular row of A:

```
>> A(2,:)

ans =

    2     2     2
```

The ":" operator you have just used generates equally spaced vectors. You can use it to specify a range of values to access in the matrix:

```
>> A(2,1:2)

ans =

    2     2
```

You can also use it to create a vector:

```
>> y = 1:3

y =

    1     2     3
```

The default increment is 1, but you can specify the increment to be something else:

```
>> y = 1:2:6

y =
```

```
    1     3     5
```

Here, the value of each vector element has been increased by 2, starting from 1, while *less than* 6.

You can easily concatenate vectors and matrices in MATLAB:

```
>> [y, A(2,:)]

ans =

    1     3     5     2     2     2
```

You can also easily delete matrix elements. Suppose you want to delete the 2nd element of the vector y:

```
>> y(2) = []

y =

    1     5
```

MATLAB has several built-in matrices that can be useful. For example, zeros(n,n) makes an nxn matrix of zeros.

```
>> B = zeros(2,2)

B =

    0     0
    0     0
```

A few other useful matrices are:

zeros – *create a matrix of zeros*
ones – *create a matrix of ones*
rand – *create a matrix of random numbers*
eye – *create an identity matrix*

Matrix operations

An important thing to remember is that since MATLAB is matrix-based, the multiplication operator "*" denotes matrix multiplication. Therefore, A*B is *not* the same as multiplying each of the elements of A times the elements of B. However, you'll probably find that at some point you want to do element-wise operations (array operations). In MATLAB you denote an array operator by playing a period in front of the operator. The difference between "*" and ".*" is demonstrated in this example:

```
>> A = [1 1 1; 2 2 2; 3 3 3];
>> B = ones(3,3);
>> A*B

ans =

    3    3    3
    6    6    6
    9    9    9

>> A.*B

ans =

    1    1    1
    2    2    2
    3    3    3
```

Other than the bit about matrix vs. array multiplication, the basic arithmetic operators in MATLAB work pretty much as you'd expect. You can add (+), subtract (-), multiply (*), divide (/), and raise to some power (^).

MATLAB provides many useful functions for working with matrices. It also has many scalar functions that will work element-wise on matrices (e.g., the function sqrt(x) will take the square root of each element of the matrix x). Below is a brief list of useful functions. You'll find many, many more in the MATLAB help index, and also in the "Other Resources" listed at the end of this handout.

Useful matrix functions:

- A' – *transpose of matrix A. Also* transpose(A).
- det(A) – *determinant of A*
- eig(A) – *eigenvalues and eigenvectors*
- inv(A) – *inverse of A*
- svd(A) – *singular value decomposition*
- norm(A) – *matrix or vector norm*
- find(A) – *find indices of elements that are nonzero. Can also pass an expression to this function, e.g.* find(A > 1) *finds the indices of elements of A greater than 1.*

A few useful math functions:

- sqrt(x) – *square root*
- sin(x) – *sine function. See also* cos(x), tan(x), *etc.*
- exp(x) – *exponential*
- log(x) – *natural log*
- log10(x) – *common log*

abs(x) – *absolute value*
mod(x) – *modulus*
factorial(x) – *factorial function*
floor(x) – *round down. See also* ceil(x), round(x).
min(x) – *minimum elements of an array. See also* max(x).
besselj(x) – *Bessel functions of first kind*

MATLAB also has a few built-in constants, such as pi (π) and i (imaginary number).

Symbolic math

Although MATLAB is primarily used for numerical computations, you can also do symbolic math with MATLAB. Symbolic variables are created using the command "sym."

```
>> x = sym('x');
```

Here we have created the symbolic variable x. If it seems kind of lame to you to have to type in all this just to create "x", you're in luck—MATLAB provides a shortcut.

```
>> syms x
```

This is a shortcut for x = sym('x').

Symbolic variables can be used for solving algebraic equations. For example, suppose we want to solve the equation "x^4 + 3*x^2 + 3 = 5":

```
>> y = solve('x^4 + 3*x^2 + 3 = 5',x)

y =

 -1/2*(-6+2*17^(1/2))^(1/2)
  1/2*(-6+2*17^(1/2))^(1/2)
 -1/2*(-6-2*17^(1/2))^(1/2)
  1/2*(-6-2*17^(1/2))^(1/2)
```

Since MATLAB is solving for x, a symbolic variable, it writes the answer in symbolic form. That means that the solutions is written out as an expression rather than computed as a decimal value. If you want the decimal value, you need to convert to a double:

```
>> double(y)

ans =

  -0.7494
   0.7494
        0 - 1.8872i
```

```
0 + 1.8872i
```

However, sometimes you may want to see the symbolic expression. In that case, you might want MATLAB to write it out in a way that's easier to read. For this, use the command "pretty":

```
>> pretty(y)

[                    1/2 1/2]
[- 1/2 (-6 + 2 17   )      ]
[                           ]
[                  1/2 1/2 ]
[ 1/2 (-6 + 2 17   )       ]
[                           ]
[                   1/2 1/2]
[- 1/2 (-6 - 2 17   )      ]
[                           ]
[                  1/2 1/2 ]
[ 1/2 (-6 - 2 17   )       ]
```

Below are a few useful things you can do with symbolic variables in MATLAB. For more, look at the "Symbolic Math Toolbox" section of the MATLAB help.

solve – *symbolic solution of systems of algebraic equations*
int(f,x) – *indefinite integral of f with respect to x*
int(f,x,a,b) – *definite integral of f with respect to x from a to b.*
diff(f,'x') – *differentiate f with respect to x*
taylor – *Taylor series expansion of symbolic expression*
subs – *substitute values into a symbolic expression*

M-files and functions

If you are doing a computation of any significant length in MATLAB, you will probably want to make an m-file. Anything that you would type at the command prompt you can put in the m-file (for example, "script.m") and then run it all at once (by typing the name of the m-file, e.g. "script", at the command prompt). You can even add comments to your m-file, by putting a "%" at the beginning of a comment line.

You can also use m-files to create your own functions. For example, suppose you want to make a function that increments the value of each element of a matrix by some constant. And suppose you want to call the function "incrementor." You would make an m-file called "incrementor.m" containing the following:

```
function f = incrementor(x,c)
% Incrementor adds c to each element in the matrix x.
f = x + c;
```

When you pass a matrix x and value c to this function, the value of f = x+c is returned. You can now call this function from the command line or in another m-file.

```
>> incrementor(A,1)

ans =

   2   2   2
   3   3   3
   4   4   4
```

MATLAB provides several flow control statements that you can use in scripts. These include:

FOR loops:

```
for i = 1:10
        a(i) = 2;
end
```

WHILE loops:

```
while a(i) ~= 0
        b(i) = 1/a(i);
end
```

IF/ELSE statements:

```
if i > 0
        a(i) = 1;
else
        a(i) = 0;
end
```

The “break” statement can be used to exit from the current FOR or WHILE loop.

You may find it useful at some point in a script to return control to the keyboard, to examine variables or execute commands. Whenever the command “keyboard” is encountered in a script, MATLAB will return control to the keyboard. To return to the script, just type “return”. MATLAB can also prompt the user for input during a script. This is done with the “input” command:

```
x = input('prompt','s')
```

The string ‘prompt’ will be displayed to the user. The ‘s’ is an optional argument, used only if you want the input to be read in as a string.

Here’s an example m-file, “script1.m” that makes use of the function we created earlier:

```
% Script1 – increment a matrix of ones by some user-specified value, v
B = ones(3,3);
v = input('What value do you want to increment by? ');
```

```
B = incrementor(B,v)
```

When we run script1.m, this is what we get:

```
>> script1
What value do you want to increment by? 2

B =

    3    3    3
    3    3    3
    3    3    3
```

File I/O

MATLAB allows you to save matrices and read them in later. The simplest way to do this is using the commands "save" and "load". Typing in "save A" saves matrix A to a file called A.mat. If you want to read in matrix A later, just type "load A". You can also use the load command to read in ASCII files, as long as they are formatted correctly. Formatted correctly means that the number of columns in each line is the same and the columns are delimited with a space.

Suppose you have a file called "datafile.dat" that contains the following lines:

```
12.5 6 9
1 3.5 125
2 4 0
```

Notice that the columns do not have to be lined up exactly, as long as there are the same number of them on each line. You can read this into MATLAB by typing "load datafile.dat". MATLAB will read the file contents into a matrix called "datafile" (without the file extension).

If you want to read/write a binary data file, or if your ASCII file is not well-formatted, you will need to do something a little more complicated. The commands used are very similar to those in C:

fopen – *open a file for reading or writing*
fread – *read a binary file*
fscanf – *read an ASCII file*
fwrite – *write a binary file*
fprintf – *write an ASCII file*
fclose – *close a file (when you are done with it)*

Example: Suppose you want to read a binary data file called "file1.dat":

```
>> fid = fopen('file1.dat', 'r');
>> A = fread(fid);
```

```
>> fclose(fid);
```

The ‘r’ specifies that the file is to be opened for reading (‘w’ for writing, ‘r+’ for reading and writing, ‘a’ for appending). In this example, the contents of file1.dat were read into a single-column matrix A. If you wanted to read in A as a 3x3 matrix you would type:

```
>> fid = fopen(‘file1.dat’, ‘r’);
>> A = fread(fid, [3 3]);
>> fclose(fid);
```

You can also specify the data type, precision, etc. If you want to know more, there are several examples of reading and writing files in the MATLAB help. Other useful commands for file I/O are:

fgetl – *capture one line at a time from the input file*
feof – *check whether the end-of-file has been reached*
fseek/ftell – *get and set file position*
frewind – *go back to the beginning of the file*

Making plots

Now that you can load data into MATLAB and easily manipulate it, you’ll want to be able to display the results in a meaningful (and hopefully aesthetically pleasing) way. Let’s create some sample data for an example of a simple linear plot:

```
>> x = 1:10;
>> y1 = x.^2;
>> y2 = 2.*y1;
```

You can plot it using the “plot” command:

```
>> plot(x,y1)
```

Suppose you want to plot only the data points, without the line between them:

```
>> plot(x,y1,’ko’)
```

This will plot the data with black o’s (“k” for the color black, and “o” to plot o’s) instead of the default blue line. To see all the options for plotting colors/characters, type “help plot”.

Suppose you only want to look at the data between y = 0 and y = 50. Use the “set” command to change the limits on the y-axis:

```
>> set(gca, ’ylim’, [0 50])
```

The “gca” means “get handle to current axis”. The “set” command can be used to set any properties of your figure. This is useful if you are making figures using a script. However, if you are just making a plot from the command line, it may be easier to make changes directly in the figure window (for example, to change the y-limits go to the “Edit” menu and choose “Axes properties”). If you are planning on becoming an advanced user of MATLAB, it would be worthwhile to learn about how handles work in MATLAB. Check out the help sections on “get”, “set”, “gca”, and “gcf”.

Suppose you want to make a second plot, of y2. If you type “plot(x,y2)” it will overwrite the plot that we already have. If you want to be able to look at both plots at once, you can plot y2 in a new window. Type “figure” to open a new figure window:

```
>> figure
>> plot(x,y2)
```

However, you may want to plot y1 and y2 on the same set of axes. Use the “hold” command to hold the current plot and axes properties. First, return to figure 1:

```
>> figure(1)
>> hold on
>> plot(x,y2)
```

You can put multiple individual plots in the same figure window using the “subplot” command. Type “help subplot” for more information. Once you’re done with your plot, you’ll probably want to label the axes:

```
>> xlabel(‘x’)
>> ylabel(‘y’)
```

You can also give it a title:

```
>> title(‘My plot’)
```

Now, let’s do a 3-D example. First, generate some sample data:

```
>> z = peaks;
```

The “peaks” command generates a sample function. The default is a 49x49 matrix, but you can specify a different size. “Peaks” can be very useful if you want to test your plotting script or just play around with making plots.

We can make a 3-D shaded surface plot using the “surf” command:

```
>> surf(z)
```

If you don’t want to see the gridlines, you can type:

>> surf(z), shading flat

Or, if you want the colors to be smooth, you can type:

>> shading interp

If you don't like the colors that MATLAB is using, you can change them using the "colormap" command. MATLAB has several built-in colormaps, or you can make your own. For example, to change to a black-and-white color scheme, type:

>> colormap(gray)

If you want to see the color scale on the figure, type "colorbar."

Suppose you want to know the coordinates of a particular point in your plot. You can use the command "ginput" (graphical input) and point and click on the plot with the mouse.

>> [x,y] = ginput;

The above command obtains coordinates and stores them in x and y until the "Return" key is pressed. You can also specify the number of coordinates using ginput(n).

There are many more neat-o ways you can plot your data in MATLAB. A brief list of plotting commands for 2-D and 3-D data is below:

2-D
plot – *linear plot*
loglog – *log-log plot*
semilogx – *semi-log plot*
semilogy – *semi-log plot*
errorbar – *plot with error bars*
hist – *histogram plot*
polar – *polar coordinate plot*
rose – *angle histogram plot*

3-D
plot3 – *plot lines and points in 3-D space*
pcolor – *checkerboard plot*
contour – *contour plot*
surf – *3-D shaded surface*
surfc – *combination surf/contour plot*
surfl – *shaded surface with lighting*
mesh - *3-D mesh surface*
meshc – *combinantion mesh/contour plot*

meshz – *mesh with zero plane*
quiver – *plots arrows (e.g.. gradient, flow velocity)*

Once you're all done with your plots, you can save them to look at later—use "saveas" to save as a figure file. You can also make images of your figures to use in posters, publications, etc. Use the "print" command to export figures in any image format (e.g., ps, eps, tif, jpg).

Advanced operations

There's a lot more that you can do with MATLAB than is listed in this handout. Check out the MATLAB help or one of the "Other Resources" if you want to learn more about the following more advanced tools:

- Numerical integration (quad)
- Discrete Fourier transform (fft, ifft)
- Statistics (mean, median, std, var)
- Curve fitting (cftool)
- Signal processing (sptool)
- Numerical integration of systems of ODEs (ode45)

Assorted tips and tricks

- You can use some Unix commands at the MATLAB prompt, such as "ls" and "cd".
- You can use the "Tab" key to complete commands.
- You can use the up arrow to see previous commands you've typed in. Just the up arrow alone will go through previous commands in reverse order. If you type an "s", for example, and then the up arrow, you'll see previous commands that started with "s".
- You can access the Unix command line inside MATLAB by preceding commands with a "!". This is a good way to execute Unix scripts or simple Fortran/C codes from within MATLAB, especially if you are transferring data between MATLAB and external codes.
- You can use "eval" to execute a string as a MATLAB expression. This can be very useful in m-files, such as when the filenames are variables.
- If you ever find yourself questioning the meaning of things, type "why".

Getting help

MATLAB has an interactive "Help Desk" which includes documentation and examples for pretty much everything you'd want to do. However, it can be slow to navigate, so if you just want to find out how to use a particular function, it's easier to use the "help" command. Useful commands for accessing MATLAB help are:

helpdesk – *starts the MATLAB interactive help browser*
help name – *displays documentation for function/command 'name.'*
lookfor string – *searches all help files for 'string' in the description (first comment line), and displays the function names with the description.*

Suppose you want to find out how to use the tangent function, tan(x).

```
>> help tan
 TAN    Tangent of argument in radians.
    TAN(X) is the tangent of the elements of X.

    See also atan, tand, atan2.

    Overloaded functions or methods (ones with the same name in other directories)
      help sym/tan.m

    Reference page in Help browser
      doc tan
```

Suppose you want to find out how to use the tangent function, but you don't know what it is called.

```
>> lookfor tangent
ACOT   Inverse cotangent, result in radian.
ACOTD  Inverse cotangent, result in degrees.
ACOTH  Inverse hyperbolic cotangent.
ATAN   Inverse tangent, result in radians.
ATAN2  Four quadrant inverse tangent.
ATAND  Inverse tangent, result in degrees.
ATANH  Inverse hyperbolic tangent.
COT    Cotangent of argument in radians.
COTD   Cotangent of argument in degrees.
COTH   Hyperbolic cotangent.
TAN    Tangent of argument in radians.
TAND   Tangent of argument in degrees.
TANH   Hyperbolic tangent.
```

Other resources

Most of the material in this handout was taken from previous MATLAB resource seminars, which are available in /home/datalib/Resource_Class/Matlab.

Extensive documentation and examples are available on the web at the MATLAB website, http://www.mathworks.com, under "Support."

There are several books available that contain detailed information about all of MATLAB's capabilities with examples. A good one is "Mastering MATLAB." The book also has a website that has all of the example m-files available for download: http://www.eece.maine.edu/mm.