COMS W4737/E6737 Matlab Tutorial

Start Matlab

If you are running windows or Mac OSX, you can start matlab by choosing it from the menu. To start matlab on a unix system, open up a unix shell and type the command to start the software: matlab. This will start up the software, and it will wait for you to enter your commands. In the text that follows, any line that starts with two greater than signs (>>) is used to denote the matlab command line. This is where you enter your commands.

Matlab maintains a single global namespace for all variables. When you want to make the command window clean, use command clc to clear the window; when you want to free the variables in the workspace, you can use clear to clear them from memory.

Vectors and Matrices

The basic data type in Matlab is an n-dimensional array of double precision numbers.

- **Defining an array**

```
>> a = 2
a =
     2
>> x = [1;2;3]
x =
     1
     2
     3
>> A = [1 2 3;4 5 6;7 8 0]
A =
     1     2     3
     4     5     6
     7     8     0
```

Notice that the rows of a matrix are separated by semicolons, while the entries on a row are separated by spaces (or commas).

A useful command is ``whos", which displays the names of all defined variables and their types:

```
>> whos
  Name        Size            Bytes  Class
  A           3x3                72   double array
  a           1x1                 8   double array
  x           3x1                24   double array
Grand total is 13 elements using 104 bytes
```

Note that each of these three variables is an array: The scalar a is a 1x1 array, the vector x is a 3x1 array, and A is a 3x3 array (see the “size” entry for each variable).

A common task is to create a large vector with numbers that fit a repetitive pattern. Matlab can define a set of numbers with a common increment using colons. For example, to define a vector whose first entry is 1, the second entry is 2, the third is 3, up to 8, enter the following:

```
>> v = [1:8]
v =
      1      2      3      4      5      6      7      8
```

If you wish to use an increment other than 1, you have to define the start number, the value of the increment, and the last number. For example, to define a vector that starts with 2 and ends in 4 with steps of .25 you enter the following:

```
>> v = [2:.25:4]
v =
  Columns 1 through 7
    2.0000    2.2500    2.5000    2.7500    3.0000    3.2500    3.5000
  Columns 8 through 9
    3.7500    4.0000
```

Other ways to create matrices and arrays:

```
>> B=ones(3,3)   % define a 3-by-3 matrix of ones
B =
      1      1      1
      1      1      1
      1      1      1
>> C=zeros(1,3)   % define a 1-by-3 matrix (a vector) of zeros
C =
      0      0      0
```

■ Accessing elements within an array

You can view individual entries in a vector. For example, to view the first entry, just type in the following:

```
>> v(2)
ans =
     2.2500
```

This command prints out entry 2 in vector *v*. Also notice that a new variable called *ans* has been created. Any time you perform an action that does not include an assignment, matlab will put the label *ans* on the result.

Matlab will allow you to look at specific parts of the vector. If you want to only look at the first three entries in a vector you can use the command:

```
>> v(1:3)
ans =
       2.0000     2.2500     2.5000
```

For 2D matrix, you need to specify the row and column to index an element:

```
>> A = [ 1 2 3; 3 4 5; 6 7 8]
A =
       1     2     3
       3     4     5
       6     7     8
>> A(2,1)
ans =
       3
```

This command prints out entry (row 2, column 1) in matrix A

```
>> A(1:2, 2:3)
ans =
       2     3
       4     5
```

This command returns a sub matrix of A

■ Basic operations

Usually, you would like to get the size of an array. "size" command helps you do so:

```
>> [rows, cols] = size(A)
rows =
       3
cols =
       3
```

Arithmetic operations include +, -, *, /, ^. To compute the square root of number 3, just type:

```
>> 3^0.5
ans =
     1.7321
```

pi is a global constant in matlab for π:

```
>> pi
ans =
     3.1416
```

To compute exponential and logarithm, function *exp* and *log* (log10, log2…) are provided:

```
>> log(3)
ans =
      1.0986
>> exp(5)
ans =
  148.4132
```

For vectors and matrices, Matlab follows the standard notation used in linear algebra. Both addition and subtraction are defined in the standard way. For example:

```
>> v = [0:2:8]
v =
     0     2     4     6     8
>> v(1:3)-v(2:4)
ans =
    -2    -2    -2
```

To define a new vector u with the numbers from 0 to -4 in steps of -1, we do the following:

```
>> u = [0:-1:-4]
u =
     0    -1    -2    -3    -4
```

We can now add u and v together in the standard way:

```
>> u+v
ans =
     0     1     2     3     4
```

Additionally, scalar multiplication is defined in the standard way. Also note that scalar division is defined in a way that is consistent with scalar multiplication:

```
>> -2*u
ans =
     0     2     4     6     8
>> v/3
ans =
     0    0.6667    1.3333    2.0000    2.6667
```

Matrix vector multiplications :

```
>> A
A =
     1     2     3
     3     4     5
     6     7     8
>> b=[-1; 1; 2]
```

```
b =
    -1
     1
     2
>> A*b
ans =
     7
    11
    17
>> b'
ans =
    -1     1     2
>> A*b'
??? Error using ==> mtimes
Inner matrix dimensions must agree.
```

There are many times where we want to do an operation to every entry in a vector or matrix. Matlab will allow you to do this with "element-wise" operations. For example, suppose you want to multiply each entry in vector v with its corresponding entry in vector b. In other words, suppose you want to find v(1)*b(1), v(2)*b(2), and v(3)*b(3). It would be nice to use the "*" symbol since you are doing some sort of multiplication, but since it already has a definition, we have to come up with something else. The programmers who came up with Matlab decided to use the symbols ".*" to do this. In fact, you can put a period in front of any math symbol to tell Matlab that you want the operation to take place on each entry of the vector.

```
>> v = [1 2 3]'
v =
     1
     2
     3
>> b = [2 4 6]'
b =
     2
     4
     6
>> v.*b
ans =
     2
     8
    18
>> v./b
ans =
    0.5000
    0.5000
```

```
      0.5000
```

■ Matrix Functions

Once you are able to create and manipulate a matrix, you can perform many standard operations on it. For example, you can find the inverse of a matrix. You must be careful, however, since the operations are numerical manipulations done on digital computers. In the example, the matrix A is not a full matrix, but matlab's inverse routine will still return a matrix.

```
>> inv(A)
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 3.469447e-018.
ans =
   1.0e+015 *
    -2.7022     4.5036    -1.8014
     5.4043    -9.0072     3.6029
    -2.7022     4.5036    -1.8014
```

By the way, Matlab is **case sensitive**. This is another potential source of problems when you start building complicated algorithms.

```
>> inv(a)
??? Undefined function or variable a.
```

Other operations include finding an approximation to the eigen values of a matrix. There are two versions of this routine, one just finds the eigen values, the other finds both the eigen values and the eigen vectors. If you forget which one is which, you can get more information by typing help eig at the matlab prompt.

```
>> eig(A)
ans =
    14.0664
    -1.0664
     0.0000
>> [v,e] = eig(A)
v =
    -0.2656     0.7444    -0.4082
    -0.4912     0.1907     0.8165
    -0.8295    -0.6399    -0.4082
e =
    14.0664          0          0
          0    -1.0664          0
          0          0     0.0000
>> diag(e)
ans =
     14.0664
     -1.0664
       0.0000
```

Cell arrays

Cell array is an array whose entries can be data of any type. The index operator {} is used in place of () to indicate that an array is a cell array instead of an ordinary array:

```
>> x = [2;1];
>> b = true;
>> A{1}=x
A =
    [2x1 double]
>> A{2}=b
A =
    [2x1 double]    [1]
>> whos
  Name          Size                    Bytes  Class
  A             1x2                       137   cell array
  b             1x1                         1   logical array
  x             2x1                        16   double array
Grand total is 8 elements using 154 bytes
```

Another way to create the same cell array is to place the entries inside of curly braces:

```
>> B={x, b}
B =
    [2x1 double]    [1]
```

For more information about cell arrays, see “help cell”.

Conditionals and Loops

■ Conditionals

Matlab has a standard if-elseif-else conditional; for example:

```
>> t = rand(1);
>> if t > 0.75
        s = 0;
   elseif t < 0.25
        s = 1;
   else
```

```
            s = 1-2*(t-0.25);
      end
>> s
s =
          0
>> t
t =
        0.7622
```

The logical operators in Matlab are <, >, <=, >=, == (logical equals), and ~= (not equal). These are binary operators which return the values 0 and 1 (for scalar arguments):

```
>> 5>3
ans =
          1
>> 5<3
ans =
          0
>> 5==3
ans =
          0
```

Thus the general form of the if statement is:

```
if expr1
      statements
elseif expr2
      statements
  .
  .
  .
else
      statements
end
```

The first block of statements following a nonzero expr executes.

■ *for* Loops

The *for* loop allows us to repeat certain commands. If you want to repeat some action in a predetermined way, you can use the *for* loop. All of the loop structures in matlab are started with a keyword such as "for", or "while" and they all end with the word "end".

The *for* loop is written around some set of statements, and you must tell Matlab where to start and where to end. Basically, you give a vector in the "for" statement, and Matlab will loop through for each value in the vector:

For example, a simple loop will go around four times each time changing a loop variable, j:

```
>> for j=1:4,
      j
  end

  j =
         1
  j =
         2
  j =
         3
  j =
         4
```

When Matlab reads the "for" statement it constructs a vector, [1:4], and j will take on each value within the vector in order. Once Matlab reads the "end" statement, it will execute and repeat the loop. Each time the for statement will update the value of j and repeat the statements within the loop. In this example it will print out the value of j each time.

Another example, in which we want to perform operations on the rows of a matrix. If you want to start at the second row of a matrix and subtract the previous row of the matrix and then repeat this operation on the following rows, a *for* loop can do this in short order:

```
>> A = [ [1 2 3]' [3 2 1]' [2 1 3]']
A =
      1     3     2
      2     2     1
      3     1     3
>> B = A;
>> for j=2:3,
     A(j,:) = A(j,:) - A(j-1,:)
end

A =
      1     3     2
      1    -1    -1
      3     1     3
A =
      1     3     2
      1    -1    -1
      2     2     4
```

Loops can be nested, Gaussian Elimination can be performed using only two loops and one statement:

```
>> for j=2:3,
      for i=j:3,
            B(i,:) = B(i,:) - B(j-1,:)*B(i,j-1)/B(j-1,j-1)
         end
      end

B =
         1      3      2
         0     -4     -3
         3      1      3
B =
         1      3      2
         0     -4     -3
         0     -8     -3
B =
         1      3      2
         0     -4     -3
         0      0      3
```

■ *while* Loops

The *while* loop repeats a sequence of commands as long as some condition is met.

```
>> x=1;
>> while 1+x > 1
         x = x/2;
   end
>> x
x =
   1.1102e-16
```

Plotting

The simplest graphs to create are plots of points in the cartesian plane. For example:

```
>> x = [1;2;3;4;5];
>> y = [0;.25;3;1.5;2];
>> plot(x,y)
```

The resulting graph is displayed in Figure 1.

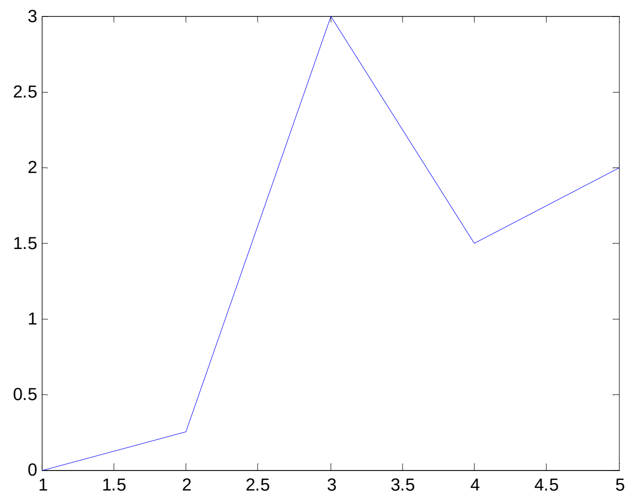

Figure 1

Notice that, by default, Matlab connects the points with straight line segments. An alternative is the following (see Figure 2):

```
>> plot(x,y,'o')
```

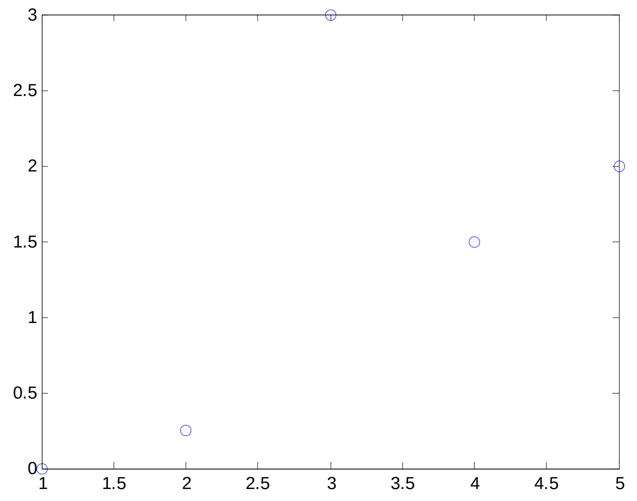

Figure 2

The following example also demonstrates one of the most useful commands in Matlab, the "help" command.

```
>> help plot

PLOT    Linear plot.
    PLOT(X,Y) plots vector Y versus vector X. If X or Y is a matrix,
    then the vector is plotted versus the rows or columns of the matrix,
    whichever line up.   If X is a scalar and Y is a vector, length(Y)
```

disconnected points are plotted.

PLOT(Y) plots the columns of Y versus their index.
If Y is complex, PLOT(Y) is equivalent to PLOT(real(Y),imag(Y)).
In all other uses of PLOT, the imaginary part is ignored.

Various line types, plot symbols and colors may be obtained with
PLOT(X,Y,S) where S is a character string made from one element
from any or all the following 3 columns:

b	blue	.	point	-	solid
g	green	o	circle	:	dotted
r	red	x	x-mark	-.	dashdot
c	cyan	+	plus	--	dashed
m	magenta	*	star	(none)	no line
y	yellow	s	square		
k	black	d	diamond		
		v	triangle (down)		
		^	triangle (up)		
		<	triangle (left)		
		>	triangle (right)		
		p	pentagram		
		h	hexagram		

….

See also **plottools, semilogx, semilogy, loglog, plotyy, plot3, grid, title, xlabel, ylabel, axis, axes, hold, legend, subplot, scatter**.

Scripts and functions

A script is simply a collection of Matlab commands in an m-file (a text file whose name ends in the extension ".m"). Upon typing the name of the file (without the extension), those commands are executed as if they had been entered at the keyboard. The m-file must be located in one of the directories in which Matlab automatically looks for m-files; a list of these directories can be obtained by the command path. (See help path to learn how to add a directory to this list.) One of the directories in which Matlab always looks is the current working directory; the command *cd* identifies the current working directory, and *cd newdir* changes the working directory to newdir.

For example, suppose that plotsin.m contains the lines

```
x = 0:2*pi/N:2*pi;
y = sin(w*x);
```

```
plot(x,y)
```

Then the sequence of commands

```
>> N=100;w=5;
>> plotsin
```

produces Figure 3.

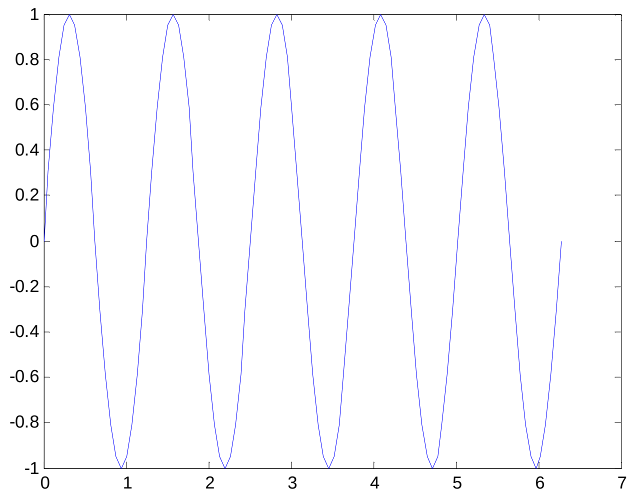

Figure 3: Effect of an m-file

As this example shows, the commands in the script can refer to the variables already defined in Matlab, which are said to be in the global workspace (notice the reference to N and w in plotsin.m). As I mentioned above, the commands in the script are executed exactly as if they had been typed at the keyboard.

Much more powerful than scripts are functions, which allow the user to create new Matlab commands. A function is defined in an m-file that begins with a line of the following form:

function [output1,output2,...] = cmd_name(input1,input2,...)

The rest of the m-file consists of ordinary Matlab commands computing the values of the outputs and performing other desired actions. It is important to note that when a function is invoked, Matlab creates a local workspace. The commands in the function cannot refer to variables from the global (interactive) workspace unless they are passed as inputs. By the same token, variables created as the function executes are erased when the execution of the function ends, unless they are passed back as outputs.

Here is a simple example of a function; it computes the function $f(x) = \sin(x^2)$, The following commands should be stored in the file fcn.m (the name of the function within Matlab is the name of the m-file, without the extension):

```
function y = fcn(x)
y = sin(x.^2);
```

Note that I used the vectorized operator **.^** so that the function fcn is also vectorized.) With this function defined, I can now use fcn just as the built-in function sin:

```
>> x = (-pi:2*pi/100:pi)';
>> y = sin(x);
>> z = fcn(x);
>> plot(x,y,x,z)
>> grid
```

The graph is shown in Figure 4. Notice how plot can be used to graph two (or more) functions together. The computer will display the curves with different line types--different colors on a color monitor, or different styles (e.g. solid versus dashed) on a black-and-white monitor. Note also the use of the grid command to superimpose a cartesian grid on the graph.

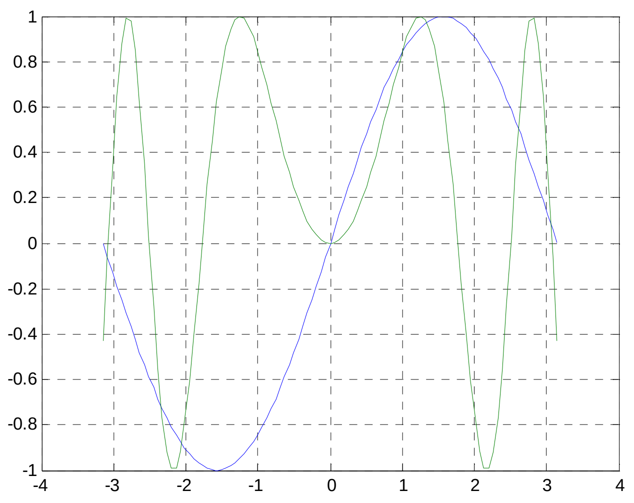

Figure 4: Two curves graphed together

Learning More

The following is a list of Matlab function calls often used:

- General: help, lookfor, 1:N, rand, zeros, A', reshape, size (N and A are variables here)
- Math: max, min, cov, mean, norm, inv, pinv, det, sort, eye
- Control: if, for, while, end, %, function, return, clear
- Display: figure, clf, axis, close, plot, subplot, hold on, fprintf
- Input/Output: load, save, ginput, print,

Two ways to find what you need:

1. type "help *function_name*" in the command window
2. online help documentation and tutorials such as http://www.mathworks.com/help/techdoc/ref/