

# CSE/Math 451

## Matlab Demo

### Fall 2007

#### The Demo

First, log on to your account in 218 IST. If you are unable to do that, please get hold of me. On your first UNIX prompt type

```
% mkdir matlab (or whatever you wish to name this directory)
```

```
% cd matlab
```

Then type

```
% matlab
```

You should get a brief graphic display and then the prompt

```
>>
```

and a big GUI. If you do not want the GUI type

```
% matlab -nojvm
```

Type the command

```
>> A = randn(5,5)
```

This generates a random  $5 \times 5$  matrix according to the normal distribution.

Open a second X-window and also type

```
% cd matlab
```

You will use this second X-window to download and use m-files.

In the matlab window type

```
>> Ainv=inv(A)
```

Type

```
>> I= eye(5,5)
```

What do **inv** and **eye** do? Use the **help** command if this is not obvious.

Compute

```
>> R= I-Ainv*A
```

Is the answer what you expected? Now type

```
>> B=ones(5,5)
```

and type

```
>> inv(B)
```

Type

```
>> help sum
```

Use the sum function to sum all of the elements in A.

Type

```
>> ii=[1 4 5]
>> jj=[2 3 5]
>> C=A(ii,jj)
```

What did you just do? Do you now see how MATLAB indexes arrays? What is  $A(:, 1:3)$ ,  $A(2:4, :)$  and  $A(2:4, 1:3)$ ?

Notice MATLAB automatically prints the results of any computation on the screen. Under some circumstances, that is not a good idea. If we type

```
>> ii=[1 4 5];
>> jj=[2 3 5];
>> C=A(ii,jj);
```

then MATLAB does its work “silently.” But everything is still there. Type  
>> C

to display the answer.

Now a brief introduction to plotting.

Type

```
>> inc=pi/25
>> x=0:inc:2*pi
>> y=sin(x)
>> plot(x,y)
```

To label the graph type

```
>> title('Plot of sin x for one period')
>> xlabel('Domain – From zero to 2*pi')
>> ylabel('Range – From -1 to 1')
```

You can print the plot clicking on “File” and then on “Print” and then click on “Print.” DO NOT DO THIS NOW. For now, click “Cancel.” Print it at your convenience some other time. If you click on “Save As.” in the file menu, you can save the graph as .ps (PostScript) file which can be viewed with ghostview.

Open a browser, say Firefox or Mozilla. Go to <http://www.cse.psu.edu/~barlow/cse455/quadroots0.m> You are now to download this into your matlab directory. To do that, click “File” and the “Save As.”. It should give you a list of your directories, click “matlab.” Then type the file name quadroots0.m and click OK.

Repeat the procedure for <http://www.cse.psu.edu/~barlow/cse455/quadroots.m>.

Type

```
>> diary
>> a=1
>> b=1e10
>> c=1
>> [x1,x2,ier]=quadroots0(a,b,c)
>> [y1,y2,ier]=quadroots(a,b,c)
>> z=roots([a b c])
>> diary off
```

This is like an example we did earlier in class.

In your other window, look in your diary file. This is one way to output the results of a MATLAB session. MATLAB also supports the C input/output routines **fscan** and **fprintf**. They work like the C versions except that they have the ability to input and output matrices.

If you want to know what “roots” does, type “help roots.”

It also support many special operations that will become useful later.

So far, I have given a small portion of MATLAB. Throughout the semester, I will give you more pieces of it. MATLAB is set up so that a computer literate person can learn much on his/her own.

After you are done, type

```
>> demo
    and choose whichever demo you wish. When you are finished type
>> quit
    or
>> exit
```

**Some Tutorial Information on MATLAB** Matrices are the main data element. They can be introduced in the following four ways.

1. As an explicit list of elements.
2. Generated by built-in statements or functions.
3. Created by m-files.
4. Loaded from external data files.

For instance consider the MATLAB command

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

It immediately outputs

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

(no parentheses or brackets). Or if you create a file called gena.m and type

`A = [ 1 2 3 ; 4 5 6 ; 7 8 9]`

in that file.

Then type

`>> gena`

it will also produce  $A$  exactly as above.

It is important to remember the distinction between row and column vectors in MATLAB. If you type

`>> x = [-1.3 sqrt(3) (1+2+3)*4/5]`

you get

$$x = -1.3 \quad 1.7321 \quad 4.8000$$

Row vectors are default.

On the other hand, if you type

`>> x = [-1.3; sqrt(3); (1+2+3)*4/5]`

you get

$$x = \begin{bmatrix} -1.3 \\ 1.7321 \\ 4.8000 \end{bmatrix}$$

The most important MATLAB command is “help.”

`>> help`

This lists all help topics

`>> help inv`

Gives the help information for the function “inv” for inverting a matrix.

`>> help help`

Explains the “help” command.

MATLAB m-files contain two types of items

1. scripts

## 2. user-defined functions

The `gena.m` file described above is an example of a script. It just lines of text consisting of MATLAB commands. Invoking the script (done when typing “`gena`” or the name of the script file without the “.m”) is equivalent to typing the commands directly into MATLAB.

Functions have “function” statements as their first line. Two examples are

```
function A=fun1(x,y,z)
function [A,B,C]=fun2(x,y,z)
```

These should be placed in files called “`fun1.m`” and “`fun2.m`” respectively. Here “`x,y,z`” are input arguments, and “`A,B,C`” are output arguments. Any of these can be simple variables, vectors, or matrices. For examples of functions look at the files “`cosx.m`” and “`bisect2.m`” on the class web page.

MATLAB supports the usual programming structures “if”, “for”, “while” and “switch.” The syntax is a bit different from C syntax, but the use is much the same. The function “`bisect2.m`” contains “if” and “while” statements. The following “for” statement sums the first  $n$  integers.

```
sum=0;
for k=1:n
    sum=sum+k;
end;
```

Two useful MATLAB command are “save” and “load.” MATLAB creates a workspace for all of your variables. When you are done with a MATLAB session, you type

```
>> quit
```

and you are out of MATLAB. However, your workspace and all of the variables that you are using vanish, they will not be there when you log on to MATLAB again.

If, before you quit, you type 

```
>> save mysession
```

the workspace will be saved in the file “`mysession.mat`.” If you type

```
>> save mysession X Y Z
```

then only the variables  $X, Y$ , and  $Z$  will be saved. “save” is a completely destructive operation, “mysession.mat” is completely overwritten.

If you want to retrieve your workspace from the previous session type  
» load mysession

To know what variables you have type  
» who

Most matrix operations in MATLAB are one line.

The matrix transpose operation is

»  $B = A'$  ;

Thus if  $A$  has the contents

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{pmatrix}$$

then  $B$  will have the contents

$$B = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 0 \end{pmatrix}$$

Here are some other functions

»  $C = A+B$

just adds  $A$  and  $B$ .

»  $C = A*B$

just multiplies  $A$  and  $B$ .

»  $C = X'*Y$

does the operation  $C = X^T Y$ .

»  $b = A*x$

is a matrix–vector multiply. If the dimensions are wrong in any of these operations, you will get an error message.

There are two important functions associated with matrices, “size” and “length.” The statement

»  $[m,n]=size(A)$

returns the dimensions of the array  $A$ . If  $m = 1$ ,  $A$  is just a row vector, if  $n = 1$  then  $A$  is column vector, and if  $m = n = 1$ , then  $A$  is scalar.

The statement

»  $p=length(A)$

returns the larger of the two outputs of “size.”

There are a number of ways to generate special vectors and matrices.

Here are some examples

```
>> x = 1:5
```

yields

$$x = 1 \ 2 \ 3 \ 4 \ 5 .$$

```
>> y = 0:pi/4:pi
```

yields

$$y = 0 \ \pi/4 \ \pi/2 \ 3\pi/4 \ \pi .$$

```
>> z = 6:-1:1
```

yields

$$z = 6 \ 5 \ 4 \ 3 \ 2 \ 1 .$$

The statement

```
>> y =exp(z)
```

yields

$$y = e^6 \ e^5 \ e^4 \ e^3 \ e^2 \ e^1 .$$

Actually, it produces the appropriate floating point numbers. Of course, “exp” is exponential function.

MATLAB supports lots of math functions, such as exp, log, log10, sin, cos, tan, asin, atan, and some you have probably never heard of.

Functions can also be defined inline. An example is

```
>> f= inline('1/(x+1)')
```

They can be passed as arguments in other functions in two ways. The first is through the use of function handles. Let’s say we want to compute

$$diff(f, x, h) = [f(x + h/2) - f(x - h/2)]/h$$

the centered difference approximation to the first derivative. If we have the function “fun1” defined either in an m-file or inline, we call **diff** by

```
yprime=diff(fun1,x,h)
```

where fun1 is called a function handle. The m-file for **diff** could look like this.

```
function yprime=diff(f,x,h)
yprime=(feval(f,x+h/2)-feval(f,x-h/2))/h;
```

Alternatively, this can be done with strings. If your version of MATLAB is older than version 6, you have to do it this way. Octave does it similarly, but with slightly different syntax.

```
yprime=diff('fun1',x,h);
function yprime=diff(f,x,h)
yprime=(eval([f,'(x+h/2)'])-eval([f,'(x-h/2)']))/h;
```

MATLAB has toolboxes for a variety of purposes. In this course, we will have cause to use the spline toolbox. There are also toolboxes for symbolic processing (based on MAPLE), image processing, control, optimization, and many others. They have standard routines for linear equations, roots of non-linear equations, numerical integration, and ordinary differential equations. In other words, MATLAB already does most of the things this course teaches (and many things we do not have time to teach).