INTRODUCTION TO MATLAB

Padmanabhan Seshaiyer

First Steps

You should start MATLAB by simply typing `matlab` if you are working on a Unix system or just by double clicking the MATLAB icon if you are using a Windows based system. If all goes well you will see a MATLAB prompt

```
>>
```

inviting you to initiate a calculation. In what follows, any line beginning with >> indicates `typed input` to MATLAB. You are expected to type what follows by not the >> prompt itself. MATLAB supplies that automatically.

Arithmetic with MATLAB

MATLAB understands the basic arithmetic operations: +, -, *, /. Powers are indicated with ^, thus typing

```
>> 5*4 + 3^2
```

and pressing enter, results in

```
ans =
      29
```

The laws of precedence are built in but if in doubt, you should put parentheses appropriately. For example,

```
>> 7+3*(2/(8-2))
ans =
       8
```

The sort of elementary functions familiar on hand calculators are also available. For example,

```
>> sqrt(3^2+4*4)
ans =
       5
>> exp(log(3.2))
ans =
       3.2
```

Using Variables

You can assign numerical values to *variables* for use in subsequent calculations. For example the *volume of a sphere* can be calculated via the following sequence:

```
>> radius = 3;
>> volume=(4/3)*pi*radius^3
volume =
     113.0973
```

Note that the first line did not seem to produce a result in screen. When MATLAB encounters an instruction followed by a semi-colon ; it suppresses any visual confirmation. It really does obey the instruction and records the value of the variable in the memory. This is useful if you want to avoid cluttering up the screen with intermediate results. Each variable must somehow be assigned before you make use of it in further calculations. For example if you have followed the above example with,

```
>> x=4*pi*radius*h
```

you should get the result,

```
??? Undefined function or variable 'h'
```

This is self-explanatory. If you now type

```
>> h=3;
>> x=4*pi*radius*h
```

you should have more success. Incidentally, a quick way of repeating a previous MATLAB instruction is to press the ‘up-arrow’ key until you recover the command you need. You may also use the sideways arrows to modify any of the previous commands.

At any point typing the command,

```
>> who
```

tells about all the variables that are in the workspace of the current MATLAB session.

Vectors

A vector can be input in several ways. For example,

```
>> u=[1 3 5];
>> v=[1,3,5];
>> w=1:2:5;
```

All three commands above yield the same vector. One can perform vector operations such as,

```
>> z=u+v-w;
```

Note that the vectors described so far are *row* vectors. To get a corresponding *column* vector, we invoke the transpose of a vector. For example,

```
>> u'
ans =
       1
       3
       5
```

The difference between the vectors u and u' can be understood by typing the commands

```
>> size(u)
>> size(u')
```

which yields the row size followed by the column size for each vector. One can now multiply vectors appropriately to either yield an *innerproduct*

```
>> z=u*u'
```

or a matrix,

```
>> z=u'*u
```

Multiplication of vectors only happens if the inner matrix dimensions agree. For example,

```
>> u*u
```

would yield,

```
??? Error using ==> *
Inner matrix dimensions must agree.
```

Suppose we want a set of values `z` given by $z = u^2$ then we want

```
>> z = u.*u;
```

where the `.` is inserted before the * symbol which forces an element-by-element operation. Similarly u./v and u.^3 can be understood to be the corresponding element-by-element operations.

Example: Suppose we want to find the distance between two points A and B whose position vectors are given by **a** = (1, 2) and **b** = (5, 5) respectively. The following sequence of commands can be executed in MATLAB to obtain the desired result:

```
>> a=[1 2];
>> b=[5 5];
>> d = b – a;
>> dd = d*d';
>> dist_ab = sqrt(dd)
```

Script Files

Now suppose we want to modify the position vectors and find the distance between the new coordinates, it will not be worthwhile to type all the commands again. Instead one can create a *script* file, which will contain all the necessary information. To create a new script file, one can open their favorite text editor and type the following commands,

```
% dist_ab.m
% Calculates distance between any two position vectors a and b
d=b-a;
dd=d*d';
dist_ab=sqrt(dd)
```

Save these contents in a file named *dist_ab.m* and note that the script file has a '.m' extension which denotes that it is a MATLAB file. Now one can pick values for the vectors **a** and **b** by simply typing say,

```
>> a=[1 2 3];
>> b=[5 5 3];
```

Then find the distance by simply typing

```
>> dist_ab
dist_ab =
     5
```

This program can be called repeatedly to find the distance between any two position vectors that are entered by the user.

Function Files

It was tedious to have to assign the two vectors each time before using the above script file. One can combine the assignment of input values with the actual instruction, which invokes the script file by using a *function m-file*. Not only that, but also one can at the same time assign the answer to an output variable.

To create a new function file, one can open their favorite text editor and type the following commands,

```
% distfn.m
% Calculates the distance between two vectors a and b
% Input: a, b (position vectors)
% Output: dist_ab is the distance between a and b
function dist_ab = distfn(a , b)
  d= b – a;
  dd = d*d’;
  dist_ab = sqrt(dd);
```

Save these contents in a file named *distfn.m* and we should now be able to run,

```
>> dist_ab=distfn([1 2 3], [ 5 5 3])
```

or

```
>> a=[1 2 3];
>> b=[5 5 3];
>> dist_ab=distfn(a , b);
```

To save a certain part of the work in the MATLAB session, one can type,

```
>> diary work1
>> ….
>> ….
>> diary off
```

All the work between the diary commands will be saved into a file called *work1*.

Homework: Create a function file to deduce the maximum length of the side of a triangle ABC whose position vectors are given by **a**, **b**, **c**.

One of the most useful commands in MATLAB is the `help` command. For e.g., you can type,

```
>> help sin
>> help sqrt
```

and so on. Also note that,

```
>> help distfn
```

would return the comments that were entered at the beginning of the program written in the session.

Matrices

To **define a matrix** you start with its name. Square brackets are used to begin and end each matrix. You can separate elements with a space or a comma, and end each row with a semi-colon like this:

>> A = [1, 2, -6; 7, 5, 2; -2, 1, 0]

or this:

>> A = [1 2 -6; 7 5 2; -2 1 0]

In either case you have defined the same 3 by 3 matrix named A and you can use it at any time in a calculation or to define another matrix by using its name (A).

Examples:

>> B = [2 -4 7]

B is a 1 by 3 row matrix $\begin{bmatrix} 2 & -4 & 7 \end{bmatrix}$

>> C = [2; -4; 7;]

C is a 3 by 1 column matrix $\begin{bmatrix} 2 \\ -4 \\ 7 \end{bmatrix}$

>>D = [5, 8,-2; 3,-4, 5]

D is a 2 by 3 matrix $\begin{bmatrix} 5 & 8 & -2 \\ 3 & -4 & 5 \end{bmatrix}$

You can **edit** individual elements of a matrix as follows:

>> A(1,1) = -5 changes the element in row1 column1 of matrix A to -5

>> D(2,3) = 0 changes the element in row2 column3 of matrix D to 0

To perform **operations** on matrices you use the symbols (**+, -, ***) from the number keypad or above the numbers across the top of the keyboard.

Examples:

>> A + A adds the matrix A to itself.

>> B * C multiplies B and C

>> C*B - A A is subtracted from the product of C and B

The symbol ^ (above the number 6) is used to raise a matrix to an exponent as follows:

>> A^3 cubes the matrix A (you might also use A*A*A for the same calculation)

>> C*D an error message is displayed Matlab will give an error message when the calculation cannot be done because of a dimension mismatch.

To solve the system of equations for (x, y, z) using Gaussian Elimination:

$$a\,x + b\,y + c\,z = u$$
$$e\,x + f\,y + g\,z = v$$
$$p\,x + q\,y + r\,z = w$$

we perform the following steps in matlab.

```
>> A = [a b c; e f g; p q r];

>> b = [u; v; w];

>> M = [A b];

>> R = rref(M);

>> X = R(4, 1:3);
```

More helpful MATLAB commands:

quit *or* **exit** either of these closes the program and ends your matlab session.

save *filename* this command will save all variables (and ONLY the variables - NOT the whole session) that you have defined in the current session. This is helpful when you enter large matrices that you want to work with at a later date.

load *filename* this command loads the variables that you saved in a previous session.

lpr this is the command used to print your diary file. EXIT THE MATLAB PROGRAM. At the osf1 prompt, type: **lpr -Pst220 *filename*** (note: st220 is the printer in ST I room 220. At other locations you will use another printer name after the **-P**)

who displays variables you have defined in your current session

why matlab answers the question.(again and again)

clear clears all variables from your current session. Only use this command if you want to lose everything.

% this is used for comments. Matlab ignores any line that begins with %

Matlab has many built in matrix functions and operators. I have listed some here that you may find useful:

>> **size**(A) gives the dimension of the matrix A

>> **inv**(A) calculates the inverse of the matrix A , if it exists.

>> **det**(A) calculates the determinant of the matrix A

>> **rref**(A) calculates the row reduced echelon form of the matrix A

>> A**'** forms the transpose of the matrix A.

>> **eye** (2,2) this is the 2 by 2 identity

>> **zeros** (3,3) builds the zero matrix of any dimension

>> **ones** (3,2) fills a matrix of any size with ones.

>> **rats**(A) displays elements of the matrix A as fractions

>> **format long** displays all numbers with 15 digits instead of the usual 4 digits

Padmanabhan Seshaiyer

Some useful built-in functions:

Elementary trigonometric functions and their inverses
sin, cos, tan, sec, csc, cot, asin, acos, atan, asec, acsc, acot

Elementary hyperbolic functions and their inverses
sinh, cosh, tanh, sech, csch, coth, asinh, acosh, atanh, asech, acsch, acoth

Basic logarithmic and exponentiation functions
log, log2, log10, exp, sqrt, pow

Basic Statistical functions
max, mean, min, median, std, var, sum

Basic complex number functions
imag, real, i, j, abs, angle, cart2pol

Basic data analysis functions
fft, ifft, interpn, spline, diff, del2, gradient

Basic logical functions
and, or, xor, not, any, all, isempty, is*

Basic polynomial operations
poly, roots, residue, polyfit, polyval,conv

Function functions that allows users to manipulate mathematical expressions
feval, fminbnd, fzero, quad, ode23, ode45, vectorize, inline, fplot, explot

Basic matrix functions
zeros, ones, det, trace, norm, eig

Try the following:

```
>> f='2+cos(x)';
>> f=inline(f);
>> fplot(f,[0,2*pi])
>> x=fzero(inline('cos(x)','x'),1)
```

Plotting Basics

Suppose we wish to plot the graph $y=x^2$ in the interval [-1, 1], then just type,

```
>> x = -1:0.1:1
>> y=x.*x
>> plot(x,y)
```

Note that the axes are automatically chosen to suit the range of variables used. One can add more features to the plot using the following commands:

```
>> title('Graph of y=x^2')
>> xlabel('x')
>> ylabel('y')
```

Suppose now we want to plot the graphs $y_1=x^2$ and $y_2=2x$ on the same figure, we need,

```
>> y1=x.*x;
>> y2=2*x;
>> plot(x,y1)
>> hold on
>> plot(x,y2,'ro')
```

Note that the `hold on` command tells MATLAB to retain the most recent graph so that a new graph can be plotted. Note that axes are adjusted and the second curve is plotted with a red circle. More options on the choice for the color and symbol can be found by typing,

```
>> help plot
```

Suppose we want to plot the graphs $y_1=x^2$ and $y_2=2x$ on the same figure but partitioned into two subplots then we say,

```
>> subplot(2,1,1)
>> plot(x,y1)
>> subplot(2,1,2)
>> plot(x,y2)
```

One must now be able to employ the various plotting commands in various combinations to get the desired figure.

Conditional Statements:

`for` Loops:

This allows a group of commands to be repeated a fixed, predetermined number of times. The general form of a `for` loop is:

```
for x = array
     Commands …
End
```

For example,

```
>> for n=1:10
        x(n)=sin(n*pi/10);
   end
```

yields the vector x given by,

```
>> x
x =
Columns 1 through 7
0.3090   0.5878   0.8090   0.9511   1.0000  0.9511  0.8090
Columns 8 through 10
0.5878   0.3090   0.0000
```

Let us now use the for loop to generate the first 15 Fibonacci numbers 1,1,2,3,5,8,…

```
>> f=[1 1];
>> for k=1:15
     f(k+2) = f(k+1) + f(k);
   end
>> f
```

`while` Loops

This evaluates a group of commands an infinite number of times unlike the `for` loop that evaluates a group of commands a fixed number of times. The general form for while loop is

```
while expression
  Commands …
end
```

For example to generate all the Fibonacci numbers less than 1000, we can do the following:

```
>> f=[1 1];
>> k=1;
>> while f(k) < 1000
     f(k+1) = f(k+1) + f(k);
     k = k +1;
    end
>> f
```

if-else-end Construct

There are times when a sequence of commands must be conditionally evaluated based on a relational test. This is done by the if-else-end construct whose general form is,

```
if expression
   Commands …
end
```

For example if we want to give 20% discount for larger purchases of oranges, we say,

```
>> oranges=10;              % number of oranges
>> cost = oranges*25        % Cost of oranges
cost =
     250
>> if oranges > 5
        cost = (1-20/100)*cost;
   end
>> cost
cost =
   200
```

If there are more conditions to be evaluated then one uses the more general if-else-end construct given by,

```
   if expression
    Commands evaluated if True
   else
    Commands evaluated if False
  end
```

switch-case Construct

This is used when sequences of commands must be conditionally evaluated based on repeated use of an equality test with one common argument. In general the form is,

```
switch expression
     case test_expression1
           Commands_1 …
     case test_expression2
           Commands_2 …
      …..
     otherwise
           Commands_n ….
end
```

Let us now consider the problem of converting entry in given units to centimeters.

```
% centimeter.m
% This program converts a given measument into the equivalent in cms
% It illustrates the use of SWITCH-CASE control flow
function y=centimeter(A,units)
switch units   % convert A to cms
    case {'inch','in'}
      y = A*2.54;
    case {'feet','ft'}
      y = A*2.54*12;
    case {'meter','m'}
      y = A*100;
    case {'centimeter','cm'}
      y = A;
    otherwise
      disp(['Unknown units: ' units])
       y = nan;   %% stands for not a number
end
```

Character Strings:

In MATLAB, these are arrays of ASCII values that are displayed as their character string representation. For example:

```
>> t = ‘Hello’
t =
Hello
>> size(t)
ans =
     1     5
```

A character string is simply text surrounded by single quotes. Each character in a string is one element in the array. To see the underlying ASCII representation of a character string, you can type,

```
>> double(t)
ans =
   104   101   108   108   111
```

The function char provides the reverse transformation:

```
>> char(t)
ans =
hello
```

Since strings are numerical arrays with special attributes, they can be manipulated just like vectors or matrices. For example,

```
>> u=t(2:4)
u =
ell
>> u=t(5:-1:1)
u =
olleh
>> u=t(2:4)’
u =
e
l
l
```

One can also concatenate strings directly. For instance,

```
>> u=’My name is ’;
>> v=’Mr. MATLAB’;
>> w=[u v]
w =
My name is Mr. MATLAB
```

The function `disp` allows you to display a string without printing its variable name. For example:

```
>> disp(w)
My name is Mr. MATLAB
```

In many situations, it is desirable to embed a numerical result within a string. The following string conversion performs this task.

```
>> radius=10; volume=(4/3)*pi*radius^3;
>> t=[‘A sphere of radius ‘ num2str(radius) ‘ has volume… of ‘ num2str(volume) ’.’];
>> disp(t)
```

It may sometimes be required to find a certain part of a longer string. For example,

```
>>a=’Texas Tech University’;
>>findstr(a,’ ‘)   %% Finds spaces
ans =
  6  6  11
>>findstr(a,’Tech’)  %% Finds the string Tech
ans
  7
>>findstr(a,'Te')  %% Finds the string starting with Te
ans =
     1     7
>>findstr(a,’tech’)  %% This command is case-sensitive
ans =
   [  ]
```

If it is desired to replace all the case on Tech to TECH then one can do this by using,

```
>>strrep(a,’Tech’,’TECH’)
ans =
Texas TECH University
```

Relational Operators:

These include the operators < <= > >= == ~=

These operators can be used to compare two arrays of the same size, or to compare an array to a scalar. For example:

```
>> A=1:5, B=5-A
A =
     1     2     3     4     5
B =
     4     3     2     1     0
>> g=A>3
g =
     0     0     0     1     1
```

finds elements of A that are greater than 3. Zeros appear in the result where this condition is not satisfied and ones appear where it is.

```
>> for n=2:6
        if rem(n,2)~=0
          n
        end
end
n =
     3
n =
  5
```

Logical Operators:

The three main logical operators are: AND, OR, NOT which are represented by the symbols &, |, ~ respectively in MATLAB. Often one can combine these operators with relational expressions. For example:

```
>> g=(A>2) & (A<5)
g =
     0     0     1     1     0
```

returns ones where A is greater than 2 AND less than 5.

```
>> for n=2:6
        if (rem(n,2)~=0) & (rem(n,3)~=0)
          n
        end
end
n =
     5
```

Finally, the above capabilities make it easy to generate arrays representing discontinuous functions, which are very useful in understanding signals etc. The basic idea is to multiply those values in an array that you wish to keep with ones, and multiply all other values with zeros. For example,

```
>> x = -2:0.1:2;   % Creates Data
>> y = ones(size(x));
>> y = (x>-1)&(x<1);
>> plot(x,y)
```

This should plot a function which resembles part of a square wave that is discontinuous at `x=-1` and `x=1`.