MATLAB Tutorial

The following tutorial has been compiled from several resources including the online Help menu of MATLAB. It contains a list of commands that will be directly helpful for understanding the FEM Matlab programs to be used in the course homework.

1. The MATLAB Windows

Upon opening MATLAB you should see three windows: the workspace window, the command window, and the command history window. If you do not see these three windows, or see more than three windows you can change the layout by clicking on the following menu selections: Desktop → desktop layout → default.

2. The Command Windows

If you click in the command window a cursor will appear for you to type and enter various commands. The cursor is indicated by two greater than symbols (>>).

3. Simple scalar or number operations

After clicking in the command window you can enter commands you wish MATLAB to execute. Try entering the following: 8+4, you will see that MATLAB will then return:

```
>> 8+4
ans =
     12
```

Operations	Symbol	Example
Addition	+	2+3
Subtraction	-	2-3
Multiplication	*	2*3
Division	/	2/3
Exponentiation, a^b	^	2^3

Figure 1: scalar or number operations

4. Creating Variables

Just as commands are entered in MATLAB, variables are created as well. The general format for entering variables is: variable = expression. For example, enter y = 1 in the command window. MATLAB returns: y = 1. A variable y has been created and assigned a value of 1. This variable can be used instead of the number 1 in future math operations. For example, typing y*y at the command prompt returns: ans = 1. MATLAB is case sensitive, so y = 1 and Y = 5 will create two separate variables. Note MATLAB does not prompt output on the screen when an operation ends with the semicolon (;).

5. Getting Help and Finding Functions

MATLAB has many standard mathematical functions such as sine (sin(x)) and cosine (cos(x)). The ability to find and implement MATLAB's functions and tools is the most important skill a beginner needs to develop. MATLAB contains many built-in functions besides those described below that may be useful.

There are two different ways to get help:

- Click on the little question mark icon at the top of the screen. This will open up the help window that has several tabs useful for finding information.
- Type "help" in the command line: MATLAB returns a list of topics for which it has functions. At the bottom of the list it tells you how to get more information about a topic. As an example, if you type "help sqrt" and MATLAB will return a list of functions available for the square root.

```
>> help sqrt
 SQRT   Square root.
    SQRT(X) is the square root of the elements of X. Complex
    results are produced if X is not positive.
    See also sqrtm.
    Overloaded functions or methods (ones with the same name in other directories)
       help sym/sqrt.m
    Reference page in Help browser
       doc sqrt
```

6. Introduction to Vectors in MATLAB

MATLAB is a software package that makes it easier for you to enter matrices and vector, and manipulate them. The interface follows a language that is designed to look a lot like the notation use in linear algebra. In the following section, we will discuss some of the basics of working with vectors.

A vector is defined by placing a sequence of numbers within square braces:

```
>> v = [1 2 3]
v =
   1    2    3
```

The length of a vector is checked with

```
>> length(v)
ans =
   3
```

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

```
>> v(1)
ans =
    1
```

This command prints out entry 1 in the vector. 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.

To simplify the creation of large vectors, you can define a vector by specifying the first entry, an increment and the last entry. MATLAB will automatically figure out how many entries you need and their values. For example, to create a vector whose entries are 0, 2, 4, 6, and 8, you can type in the following line:

```
>> v = 0:2:8
v =
    0    2    4    6    8
```

Alternately, you can also use the function v = linspace(a,b,n) to generate a row vector v of n points linearly spaced between and including a and b:

```
>> v = linspace(0,8,5)
```

```
v =

    0    2    4    6    8
```

7. Introduction to Matrices in MATLAB

Defining a matrix is similar to defining a vector. To define a matrix, you can treat it like a column of row vectors:

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

Thus, a matrix is entered row by row, and each row is separated by the semicolon (;). Within each row, elements are separated by a space or a comma (,). You can also treat a matrix like a row of column vectors:

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

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
  ans       1x1                         8  double array
  v         1x5                        40  double array
Grand total is 15 elements using 120 bytes
```

The size of the matrix is checked with

```
>> size(A)
ans =
```

```
    3     3
```

Creating special matrix:

· zeros(m,n) creats an $m \times n$ matrix of zeros:

```
zeros(3,3)
ans =
    0    0    0
    0    0    0
    0    0    0
```

· ones(m,n) creates an $m \times n$ matrix of ones:

```
>> ones(3,3)
ans =
    1    1    1
    1    1    1
    1    1    1
```

· eye(n) creates an $n \times n$ identity matrix:

```
>> eye(2)
ans =
    1    0
    0    1
```

Note: If you know the size of the matrix in advance, it is always better to preallocate the memory of the matrix using the above three functions to increase the efficiency of the code, although MATLAB can dynamically change the size of the matrix.

Transpose of a matrix: In order to find the transpose of matrix A, we type

```
A'
ans =
    1    4    7
    2    5    8
    3    6    9
```

Accessing elements within a matrix: You can access the element at ith row and jth column with

```
>> A(2,3)
ans =
    6
```

MATLAB also provides columnwise or rowwise operation of a matrix. The following expression yields

```
>> A(:,3)
ans =
    3
    6
    9
```

which is the the third column of matrix A. In addition

>> A(1,:) represents the first row of A as

```
ans =
    1    2    3
```

We can also try

```
>> A(1,:)+A(3,:)
```

as addition of the first and third rows of A with the result

```
ans =
    8    10    12
```

Now let us introduce another matrix B as

```
>> B = [3 4 5;6 7 2;8 1 0];
```

Remember MATLAB does not prompt output on the screen when an operation ends with the semicolon (;).

Matrix addition: Adding two matrices is straightforward like:

```
>> C = A + B
C =
    4    6    8
   10   12    8
```

```
   15     9     9
```

Matrix subtraction: In order to subtract matrix B from matrix A, we type

```
>> C = A - B
C =
   -2    -2    -2
   -2    -2     4
   -1     7     9
```

Note that C is now a new matrix, not the summation of A and B anymore

Matrix multiplication: Similarly, matrix multiplication can be done as

```
>> C = A*B
C =
    39    21     9
    90    57    30
   141    93    51
```

As mentioned before, the notation used by MATLAB is the standard linear algebra notation you should have seen before. You have to be careful, though -- your matrices and vectors need to have the right sizes!

Matrix inverse: The inverse of a matrix is as simple as inv(A).

Determinant of a matrix: det(A) produces the determinant of the matrix A.

```
>> A = [1 3 6;2 7 8;0 3 9];
>> inv(A)
ans =
    1.8571   -0.4286   -0.8571
   -0.8571    0.4286    0.1905
    0.2857   -0.1429    0.0476
>> det(A)
ans =
    21
```

Solution of system of linear equations: The solution of a linear system of equations is frequently needed in the finite element method. The typical form of a linear system of algebraic equations is

$$\mathbf{K}d = \mathbf{f}$$

and the solution is obtained by d = K\f. For example,

```
>> K = [1 3 4;5 7 8;2 3 5];
>> f = [10;9;8];
>> d = K\f
d =
  -4.2500
   1.7500
   2.2500
```

Sparse matrix: In the finite element method, matrices are often sparse, i.e, they contain many zeros. MATLAB has the ability to store and manipulate sparse matrices, which greatly increases its usefulness for realistic problems. Creating a sparse matrix can be rather difficult, but manipulating them is easy, since the same operators apply to both sparse and dense matrices. In particular, the backslash (\) operator works with sparse matrices, so sparse systems can be solved in the same fashion as dense systems.

The command K = sparse (I, J, X, m, n, nzmax) uses vectors I, J and X to generate an m-by-n sparse matrix such that K(I(k), J(k)) = X(k). The optional argument nzmax caused MATLAB to pre-allocate storage for nzmax nonzero entries, which can increase efficiency in the case when more nonzeros will be added later to K. Any elements of K that are zero are ignored, along with the corresponding values of I and J. Any elements of X that have duplicate values of I and J are added together, which is exactly what we want when assembling a finite element matrix (you will learn about this shortly in one or two lectures).

```
>> I = 1:3;
>> J = 3:5;
>> X = 2:4;
>> K = sparse(I,J,X,5,5)
```

```
K =
   (1,3)        2
   (2,4)        3
   (3,5)        4
```

You can convert it back to full matrix with

```
>> full(K)
ans =
     0     0     2     0     0
     0     0     0     3     0
     0     0     0     0     4
     0     0     0     0     0
     0     0     0     0     0
```

Another useful command is spy, which creates a graphic displaying the sparsity pattern of a matrix (this can be useful when you want to see if the node/element numbering in a finite element grid leads to low or high bandwidth).

8. Strings in MATLAB

MATLAB variables can also be defined as string variables. A string character is a text surrounded by single quotes. For example

```
>> str = 'hello world'
str =
hello world
```

Special built-in string manipulation functions are available in MATLAB that allow you to work with strings. For example, the function *strcmpi* compares to strings

```
>> str = 'print output';
>> strcmpi(str,'PRINT OUTPUT')
ans =
     1
```

A true statement results in 1 and a false statement is 0.

Another function used frequently is *fprintf*. This function allows the user to print to the screen (or to a file) strings and numeric information in a tabulated fashion. For example

```
>> fprintf(1,'The number of nodes in the mesh is %d\n',10);
The number of nodes in the mesh is 10
```

The first argument to the function tells MATLAB to print the message to the screen. The second argument is a string variable, where %d defines a decimal character with the value of 10 and the \n defines a new line.

9. Structure in MATLAB

A structure is a data type which contains several values, possibly of different types, referenced by name. The simplest way to create a structure is by simple assignment.

```
patient.name = 'John Doe';
patient.billing = 127.00;
patient.test = [79 75 73; 180 178 177.5; 220 210 205];
```

Now entering

```
>> patient
patient =
    name: 'John Doe'
    billing: 127
    test: [3x3 double]
```

patient is an array containing a structure with three fields. To access data in structure, we type

```
>> patient.test
ans =
   79.0000   75.0000   73.0000
  180.0000  178.0000  177.5000
  220.0000  210.0000  205.0000
```

To expand the structure array, add subscripts after the structure name.

```
patient(2).name = 'Ann Lane';
patient(2).billing = 28.50;
patient(2).test = [68 70 68; 118 118 119; 172 170 169];
```

The patient structure array now has size [1 2]. Note that once a structure array contains more than a single element, MATLAB does not display individual field contents when

you type the array name. Instead, it shows a summary of the kind of information the structure contains:

```
>> patient
patient =
1x2 struct array with fields:
    name
    billing
    test
```

10. Programming with MATLAB

MATLAB offers four decision-making or control flow structures: for loops, if-else-end constructions, and switch-case constructions. Because there constructions often encompass numerous MATLAB commands, they often appear in M-files rather than being typed directly at the MATLAB prompt.

10.1 For loops

For Loops allow a group of commands to be repeated a fixed, predetermined number of times. The general form is

```
for x = array
        (commands)
end
```

Example:

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

For Loops can be nested as desired:

```
for m = 1:10
   for n = 1:10
       A(m,n) = m^2+n^2;
   end
end
```

10.2 While loops

As opposed to a For Loop that evaluates a group of commands a fixed number of times, a While Loop evaluates a group of statements an indefinite number of times given a conditions is satisfied. The general form of a While Loop is:

```
while expression
      (commands)
end
```

Example:
```
num = 0; EPS = 1;
while (1+EPS) > 1
      EPS = EPS/2;
   num = num + 1;
end
```

10.3 If-Else-End Constructions

Many times, sequences of commands must be conditionally evaluated based on a relational test. In programming language this logic is provided by some variation of a If-Else-End construction. The simplest form is

```
if expression
  (commands)
end
```

In cased where there are two alternatives, it has the form:

```
if expression
  (commands)
else
  (commands)
end
```

When there are more alternatives, it has the form:

```
if expression 1
  (commands evaluated if expression 1 is true)
elseif expression2
  (commands evaluated if expression 2 is true)
elseif ..
...
```

else

(commands evaluated if no other expression is true)

end

Exampel:
```
EPS = 1;
for num = 1: 1000
    EPS = EPS / 2;
    if (1+EPS) <= 1
        EPS = EPS*2;
    break;
    end
end
```

10.4 Switch-Case Constructions

The switch statement syntax is a means of conditionally executing code. In particular, switch executes one set of statements selected from an arbitrary number of alternatives. It has the form:

```
switch expression
  case test_expression1
        (commands)
  case test_expression2
        (commands)
  case …
   ….
  otherwise
      (commands)
end
```

In its basic syntax, switch executes the statements associated e.g. with the first case when expression == test_expression1. Here expression must be either a scalar or a character string. `switch' executes only the first matching case; subsequent matching cases do not execute. Therefore, break statements are not used.

Table Loop and Logical statements

Symbol	Explanations
= =	Two conditions are equal
~=	Two conditions are not equal
<=(>=)	One is less(greater) than or equal to the other
<(>)	One is less (greater) than the other
&	*and* operator – two conditions are met
~	*not* operator
|	*or* operator – either condition is met

11. M-files

11.1 Script M-files

A script is simply a collection of MATLAB commands in an m-file (a text 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.

Open a new m-file, and type the following commands

```
% script m-file to find the solution to the equation
A = [20 1 2; 3 20 4; 5 6 30];
b = [1 2 3]';
x = A\b
```

Save this m-file as 'solve_matrix.m" in the 'work' subdirectory. Then at the MATLAB prompt, type 'solve_matrix' and hit the enter key.

```
>> solve_matrix
x =
   0.0383
   0.0787
   0.0779
```

11.2 Function M-files

Functions 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 values of the outputs and performing other desired actions.

The function m-file name and the function name that appears in the first line of the file should be identical. In reality, MATLAB ignores the function name in the first line and executes functions based on the file name stored on disk. Function m-file names can have up to 31 characters and must begin with a letter. Any combination of letters, numbers and underscores can appear after the first character. This name rule is identical to that for variables.

Below is a simple example of a function that computes the quadratic function $f(x) = x^2 - 3x - 1$. The following commands should be stored in the file *fcn.m:*

```
function y = fcn(x)
y = x^2-3*x-1;
```

Then type command:

```
>>fcn(0.1)

ans =

  -1.2900
```

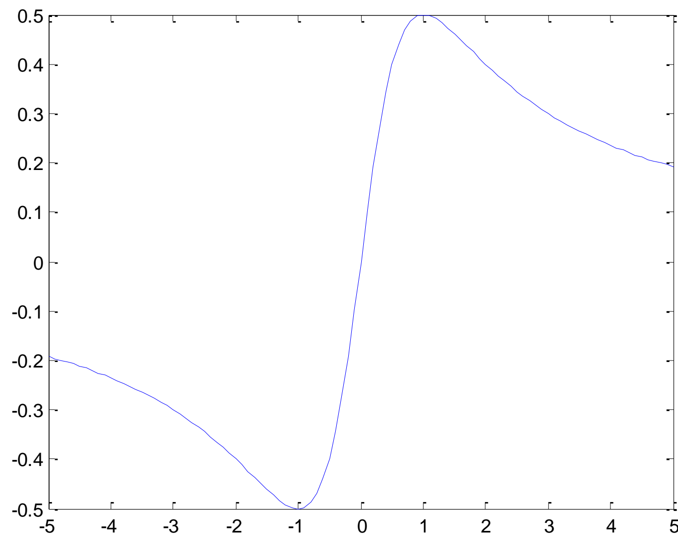
A function m-file is similar to a script file in that it is a text file having a .m extension. As with script m-file, function files are not entered in the *Command* window, but rather are external text files created with a text editor. A function m-file is different from a script m-file in that a function communicates with the MATLAB workspace only through the variables passed to it and through the output variables it creates. Intermediate variables within the function do not appear in or interact with the MATLAB workspace. In practice, your code executes more quickly if it is implemented in a function rather than a script. Therefore, it is better to use `function' even if it takes no input or output arguments.

12. Basic graphics

MATLAB is an excellent tool for visualizing and plotting results. To plot a graph the user specifies the x coordinate vector and y coordinate vector using the following syntax

```
>> x = [-5:.1:5];
>> y = x./(1+x.^2);
>> plot(x,y);
```

MATLAB also provides vectorized elementwise arithmetic operators, which are the same as the ordinary operators, preceded by “.”. Thus x.^2 squares each component of x, and x./z divides each component of x by the corresponding component of z. Addition and subtraction are performed component-wise by definition, so there are no “.+” or “._” operators. Note the difference between A^2 and A.^2. The first is only defined if A is a square matrix, while the second is defined for any n-dimensional array A.



In the above examples, MATLAB chose the solid linestyle and the colors blue for the plot. You can specify your own colors, marker, and linestyles by giving plot a third argument after each pair of data arrays. This optional argument is a character string consisting of one or more characters from the following table.

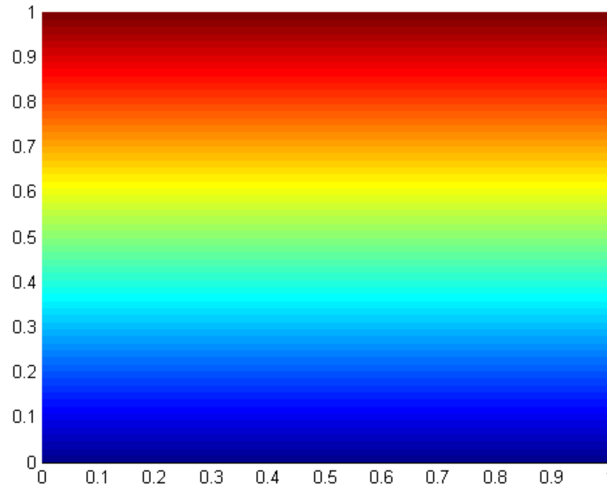
symbol	color	symbol	Marker	Symbol	linestyle
b	blue	.	point	-	solid line
g	green	O	circle	:	dotted line
r	red	x	cross	-.	dash-dot line
c	cyan	+	plus sign	--	dashed line
m	magenta	*	asterisk		
y	yellow	s	square		

k	black	d	diamond		
w	white	v	triangle(down)		
		^	triangle(up)		
		<	triangle(left)		
		>	triangle(right)		
		p	pentagram		
		h	hexagram		

- *title ('string')* add a title to the current plot.
- *xlabel('string'), ylabel('string')* label the horizontal and vertical axes, respectively, in the current plot;
- *axis([a b c d])* change the window on the current graph to $a \le x \le b, c \le y \le d$;
- *grid* adds a rectangular grid to the current plot;
- *hold on* freezes the current plot so that subsequent graphs will be displayed with the current plot;
- *subplot* put multiple plots in one graphics window
- *text (x,y,'string')* adds the string in quotes to the location specified by the point (x,y).

In MATLAB based finite element codes, we also use two specialized plots. The first plot is the *patch* function. This function is used to visualize 2D polygons with colors. The colors are interpolated from nodes of the polygon to create a colored surface. The following example generates a filled square. The colors along the x axis are the same while the colors along the y axis are interpolated between the values [0,1].

```
>> x = [0 1 1 0];  % x coordinate of the vértices of the polygon
>> y = [0 0 1 1];  % y coordinate of the vértices of the polygon
>> c = [0 0 1 1];  % Each number in c corresponds to one color of the vertices. The color
                   % in matlab is represented as a number. The Matlab will automatically
                   % translate this number to a color when using the function patch.
>> patch(x,y,c);
```



We will use the *patch* function to visualize temperatures, stresses and other variables obtained at the finite element solutions.

Another specialized plot function is the *quiver*. This function is used to visualize gradients of functions as an arrow plot. The following example demonstrate the use of *quiver* function for plotting the gradients to the function $xe^{-x^2-y^2}$.

```
% Plot the gradient field of the function xexp(-x^2-y^2):
%
[X,Y] = meshgrid(-2:.2:2);
Z = X.*exp(-X.^2 - Y.^2);
[DX,DY] = gradient(Z,.2,.2);
contour(X,Y,Z)
hold on
quiver(X,Y,DX,DY)
colormap hsv
hold off
```