Introduction to MATLAB 1

Matlab is organized as a collection of windows that can be arranged in multiple ways:

- Command window (*)
- Command history
- Current directory
- Workspace
- Editor window (*) - multiple documents
- Help window(*)

(*) recommended to keep open while you are programming

The single-line examples in this document can be entered in the MATLAB command window at the prompt. Multi-line commands and programs can be written using MATLAB source files (called m files) which will be introduced later. The command window allows the user to interactively execute MATLAB commands while maintaining the values of the variables between commands.

Keep the help window open to refer to the particular syntax and available options of the commands you'll be using.

Evaluating arithmetic expressions

```
> (4+5) * 15
> 9 – 15 * 3 / 4
```

Order of operations

As in other programming languages MATLAB has an order in which it evaluates expressions. This order is called precedence. Order is left to right when the precedence is the same.

Precedence	Operations
Highest	The contents of all parentheses are evaluated, starting from the innermost parentheses and working outward.
	Exponentials are evaluated
	Multiplications and divisions are evaluated
	Additions and subtractions are evaluated
	Colon operator
	Comparison operators (e.g. <)
Lowest	AND and OR operators

e.g. 4 * 6 + 5 is equal to 29 4 * (6+5) is equal to 44

4^2 + 3 is equal to 19 4^(2+3) is equal to 1024

Comments start with single %

All text until the end of the line will be disregarded by Matlab. Note that a double % means the start of a cell that can be evaluated separately.

Suppressing output
To suppress the output of a line put a semicolon at the end of the line. Without a semicolon the result of an expression or assignment will be displayed in the command window.

Variables
A variable can be defined by assigning a value to an identifier. **Variables need not be declared** before being used. Matlab will assign them types as they are encountered. Once a variable has been defined it will be kept in the workspace until it is cleared using the ‘clear’ command.

```
> a = 5;              %  double (default for numeric)
> s = ‘ABC’;          %  char array
> b = 3 = = 5;        %  logical (3 = = 5 has a value of 0 (false)
                      %  which is then assigned to b)
```

Workspace
The workspace contains all currently defined variables. The variables and their sizes are listed in the workspace window or can be observed by using the 'whos' command in the command window.

```
> whos
> clear
```

Numbers are stored as arrays by default

```
> a = 10;             % array with single element
> b = [ 5  7 14 ];    % 1-D array (row vector) with 3 elements
```
% $b = \begin{bmatrix} 5 & 7 & 14 \end{bmatrix}$

```
> b(1)                % array access - MATLAB uses 1 indexed convention
> bt = b';            % bt is the transpose of b;
```
$bt = \begin{bmatrix} 5 \\ 7 \\ 14 \end{bmatrix}$ (**column vector**)

```
> c = 1 : 10          % numeric sequence c(1) = 1, c(2) = 2, c(3) = 3 ... c(10) = 10
> d = 1 : 0.25 : 2    % d = [ 1  1.25  1.50  1.75  2.0 ]
> e = [ 1 2 3 4 ];    % row vector (1-D array)
> f = [ 1; 2; 3; 4 ]; % column vector (1-D array) - alternatively f=e';
```

Array Constructions

```
> a = zeros( 5, 1)    % single dimensional array with 5 elements
> b = ones( 3, 3)     % 3 x 3 two-dimensional array
> c = ones( 3)        % 3 x 3 two-dimensional array
> d = rand( 4, 2)     % array of random numbers between 0 and 1
                      % 4 rows and 2 columns
```

Size

```
> length( a )         % gives the number of elements in the 1D array
                      % returns the size of the longest dimension of a
> size( b )           % gives the dimensions of the matrix b
```

Two-dimensional arrays

```
> a = [ 2 6 4; 9 7 2; 3 4 8 ]      % 2-D array with  3 rows and 3 columns
```

$$\% \; a = \begin{bmatrix} 2 & 6 & 4 \\ 9 & 7 & 2 \\ 3 & 4 & 8 \end{bmatrix}$$

```
> a(1, 2)               % array access: row 1 column 2
> b = zeros( 3, 5);     % array with 3 rows and 5 columns initialized with zeros
```

Extracting slices and submatrices

```
> c = [ 5 2 6 3 4 8; 6 1 4 2 3 6 ];
> c( 1, : )             % displays elements in the top row, : alone means all elements
> c( : , 3)             % third column
> c( 1:2, 1:3 )         % extract a 2x3 matrix from c
```

Transpose of a matrix

```
> c'                    % will swap the rows and columns of c 2x6->6x2
> [ 4 5 6 8 2 ] '       % will result in a column vector
```

Addition and Subtraction of arrays

```
> a = [ 1 2 3];
> b = [ 4 5 6];
> a + b                 % perform addition to yield [ 5 7 9 ]
                        % a and b need to be the same size
```

Concatenation

```
> [ a b ]               % concatenates a and b horizontally
> [ a; b ]              % concatenates a and b vertically
```

Multiplication and division with a scalar

```
> a * 2                 % if a = [ 1 2 3] this will yield [ 2 4 6 ]
```

Elementwise multiplication and division

```
> a .* b                % multiply element by element -> [ 4 10 18]
> a ./ b
```

Matrix multiplication

```
> a = [ 1 2 3; 3 4 5; 4 5 6; 6 3 4];      % 4 x 3
> b = [ 3 1; 5 2; 7 4];                   % 3 x 2
> a * b                                   % 4 x 2
```

Raising to a power

```
>  5 ^ 2;
> a .^ 3;                                 % elementwise
```

Logical operators

```
> 5 = = 7
> 5 ~ = 7                          % ~ means NOT
> 5 ~ = 8 && ~( 5 = = 9)           % && logical AND
> 5 ~ = 8 || ~( 5 = = 9)           % || logical OR
```

Vectorized operations

```
> i = 1:10;
> y = 2 * i + 3;

> j = 11:20;
> z = 3*i + 5*j;         % z is the same size as i and j
> w = (i+5) .* (j*2)     % w is the same size as i and j because .* is an
                         % elementwise operation
```

e.g. the following two code segments are equivalent in terms of the value of s.

Non-vectorized	Vectorized

s=0;
for i =1:30
    s= s + 2*i;
end
	s = sum(2:2:60);

The function called ‘sum’ will add all elements of its input vector. If the input is a matrix, then it adds the columns and returns a vector.

The function called ‘mean’ operates similarly but takes the average instead of the sum.

Data types

logical	Logical (0 or 1) true and false values
char	Character
int8	8-bit signed integer
uint8	8-bit unsigned integer
int16	16-bit signed integer
uint16	16-bit unsigned integer
int32	32-bit signed integer
uint32	32-bit unsigned integer
int64	64-bit signed integer
uint64	64-bit unsigned integer
single	Single-precision floating-point number
double	Double-precision floating-point number

Casting of types

```
> a = 130;                 % a is of type double
> b = uint8( a );          % b is of type uint8
> c = int8( a );           % c will be limited to the maximum of int8
```

Plotting

```
> plot( y )             % plots y against its index with lines joining the points
> plot( t, y)           % plots elements of y against elements of t
> plot( t, y, ‘r’)      % uses red
> plot( t, y, ‘b+’)     % uses blue ‘+’ markers – no lines between points
> plot( t, y, 'k-+')    % uses black ‘+’ markers and lines
```

Displaying images

```
> imagesc( img )        % draws a color graph with the value represented by color
                        % where img is a two-dimensional array
                        % sc refers to the automatic scaling

> A = imread('autumn.tif');         % read MATLAB built-in color image
> imshow(A);                        % display image (or use image(A))

> A = imread('cameraman.tif');      % read MATLAB built-in grayscale image
> imshow(A);                        % display image
```