Lab 1: Introduction to MATLAB

1 Getting Ready

2 Help *(Is Your MATLAB Friend)*

Using Command line (start with the following prompt >>)

1. >> help
2. >> doc

Tryout: help + , help * , help plot

2.1 MATLAB as Calculator

The basic arithmetic operators are + - * / ^ combined with brackets( ).

```
>>  2 + 3/4*5
ans =
5.7500
```

MATLAB auto generates a variable called (**ans**), which save the most recent operation answer.

2.2 Numbers & Formats

Type	Examples
Integer	1362 , -224863
Real	1.25 , -10.37
Complex	3.21 - 4.3i (i=$\sqrt{-1}$)
Inf	Infinity (result of dividing by 0)
NaN	Not a Number, 0/0

N.B. The "e" notation is used for very large or very small numbers:
$-1.3412e+03 = -1.3412 \times 10^{3} = -1341.2$
$-1.3412e-01 = -1.3412 \times 10^{-1} = -0.13412$

2.3 Variables (Assignment Statements)

Assignment Statements form: Left variable = assigned value + Right variable

```
>> x = 3 - 2^4;
>> y = x*5;
>> x, y
x =   -13
y =   -65
```

We used (;) to suppress output printing on the screen. Each variable must be assigned a value before it may be used on the right of an assignment statement.

2.4 Variable Names

Legal names consist of any combination of letters and digits, starting with a letter BUT it is recommended to use names that reflect the values they represent.

These **are not** allowable : Net-Cost, 2pay, %x, @sign
These are allowable : NetCost, Left2Pay, x3, X3, z25c5

Special names *(reserved words)* should not be used as variable names otherwise you will get incorrect results.

Special names	Reserved MATLAB function
eps	Floating-point relative accuracy
i	Imaginary unit
inf	Infinity
j	Imaginary unit
NaN	Not-a-Number
pi	Ratio of circle's circumference to its diameter

Further Commands

Managing commands and functions.	
`help`	On-line documentation.
`doc`	Load hypertext documentation.
`what`	Directory listing of M-, MAT- and MEX-files.
`type`	List M-file.
`lookfor`	Keyword search through the HELP entries.
`which`	Locate functions and files.
`demo`	Run demos.

Controlling the command window.	
`cedit`	Set command line edit/recall facility parameters.
`clc`	Clear command window.
`home`	Send cursor home.
`format`	Set output format.
`echo`	Echo commands inside script files.
`more`	Control paged output in command window.

Managing variables and the workspace.	
`who`	List current variables.
`whos`	List current variables, long form.
`load`	Retrieve variables from disk.
`save`	Save workspace variables to disk.
`clear`	Clear variables and functions from memory.
`size`	Size of matrix.
`length`	Length of vector.
`disp`	Display matrix or text.

Exercise 1

Write a Matlab script that calculates x=2^4+1+3/4*5 and y=2^(4+1)+3/(4*5) and name a variable z, which is equal to x-y. Make sure that the script outputs all three variables underneath each other. The testing program is capital sensitive, so make sure to use the small letters given in the template.

Exercise 2

Write a MATLAB script that calculates all the following equations and assign the answers to these equations to the corresponding letters. Do not suppress the output. Again, use the template provided.

a = -2^3+9

b = 2/3*3

c = 3*2/3

d = 3*4-5^2*2-3

e = (2/3^2*5)*(3-4^3)^2

f = 3*(3*4-2*5^2-3)

3 Functions

3.1 Built in Functions

You can find a complete list of all built in function under the following topics ***"Help → MATLAB → Functions → Mathematics → Elementary Math "***

Trigonometric Functions	Exponential	Complex	Rounding & Remainder
sin	exp	abs	ceil
cos	log	angle	fix
tan	log10	imag	floor
asin	pow2	real	mod
acos	log2	conj	rem
atan	sqrt	complex(a,b)	round

Tryout (1)

(a) >> x = 5*cos(pi/6)

(b) >> y = 5*sin(pi/6)

(c) >> acos(x/5)

(d) >> asin(y/5)

(e) >> pi/6

Tryout (2)

(a) >> x = 9;

(b) >> sqrt(x)

(c) >> exp(x)

(d) >> log(sqrt(x))

(e) >> log10(x^2+6)

3.2 Functions m-files

You can use MATLAB to build your own function and use it in a similar way you use the built in functions. Here we will give an example of creating a function that will calculate the area, A as output, of a triangle with sides of length a, b and c as inputs.

$$A = \sqrt{s(s-a)(s-b)(s-c)},$$

where $s = (a+b+c)/2$.
The main steps to follow when defining a MATLAB function are:

1. Decide on a name for the function, making sure that it does not conflict with a name that is already used by MATLAB. In this example the name of the function is to be TRIarea, so its definition will be saved in a file called TRIarea.m
2. The first line of the file must have the format:

   function [list of outputs] = function name(list of inputs)

   For our example, the output (A) is a function of the three variables (inputs) a, b and c so the first line should read

   function [A] = TRIarea(a,b,c)
3. Document the function. That is, describe briefly the purpose of the function and how it can be used. These lines should be preceded by % which signify that they are comment lines that will be ignored when the function is evaluated.
4. Finally include the code that defines the function. This should be interspersed with sufficient comments to enable another user to understand the processes involved.

The complete file might look like:

```
function [A] = TRIarea(a,b,c)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Compute the area of a triangle whose
% sides have length a, b and c.
% Inputs:
% a,b,c: Lengths of sides
% Output:
% A: area of triangle
% Usage:
% Area = TRIarea(2,3,4);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
s = (a+b+c)/2;
A = sqrt(s*(s-a)*(s-b)*(s-c));
%%%%%%%%% end of area %%%%%%%%%%%%%%
```

Tryout: >> help TRIarea
This will produce the leading comments you have just inserted from the file
Tryout:

```
>> Area = TRIarea(10,15,20)
Area =
72.6184
```

where the result of the computation is assigned to the variable Area. The variable s used in the definition of the function above is *(a local variable)*; its value is local to the function and cannot be used outside

```
>> s
??? Undefined function or variable s.
```

If we were to be interested in the value of s as well as A, then the first line of the file should be changed to

function [A,s] = TRIarea(a,b,c)

where there are two output variables. This function can be called in several different ways:

1. No outputs assigned

   ```
   >> TRIarea(10,15,20)
   ans =
   72.6184
   ```

gives only the area (first of the output variables from the file) assigned to ans; the second output is ignored.

2. One output assigned

```
>> Area = TRIarea(10,15,20)
Area =
72.6184
```

again the second output is ignored.

3. Two outputs assigned

```
>> [Area, hlen] = TRIarea(10,15,20)
Area =
72.6184
hlen =
22.5000
```

4 Loops & Conditional statements

4.1 For Loop

Is used to execute statements specified number of times

```
>> Sumation=0;
>> for index=1:1:10
Sumation=Sumation+index
end
```

4.2 While Loop

There are some occasions when we want to repeat a section of MATLAB code until some logical condition is satisfied, but we cannot tell in advance how many times we have to go around the loop. This we can do with a *while ... end* construct.
Ex. What is the greatest value of n that can be used in the sum $1^2 + 2^2 + \ldots + n^2$ and get a value of less than 100?

```
>> S = 1; n = 1;
>> while S+(n+1)^2 < 100
n = n+1; S = S + n^2;
end
>> [n, S]
ans = 6      91
```

The lines of code between while and end will only be executed if the condition S+(n+1)ˆ 2 ¡ 100 is true.

4.3 If ... then ... else ... end

This allows us to execute different commands depending on the truth or falsity of some logical tests.
Ex. To test whether or not Π^e is greater than or equal to, e^{Π} :

```
>> N1=piˆexp(1); N2=exp(pi);
>> if N1 >= N2
disp('The condition is True')
else
disp('The condition is False')
end
```

Exercise 3

The triangle inequality theorem states that the sum of any two sides length must be greater than the length of the third side. Otherwise, you cannot create a triangle from the three sides. The function TRIarea we have build before does not check to see if this condition is fulfilled (**Tryout:** area(1,2,4)). So, try to modify the file so that it computes the area only if satisfy the triangle inequality theorem and assigns the value 0 to the area if this theorem is not satisfied. **N.B.** You have to check this theorem for every pair of sides.

5 Vectors

Vectors are list of numbers that can be represented using the following two types of notation

1. Column vector : Vc= [1; 3; 5]
2. Row vector : Vr= [1, 3, 5] or Vr=[1 3 5]

The number of entries is known as the "length" of the vector and the entries are often referred to as "elements" or "components" of the vector. The entries must be enclosed in square brackets.

```
>> Vr = [1, 3, 5]
>> Vc = [2; 4; 6]
>> L1=length(Vr),L2=length(Vc)
```

5.1 The Colon Notation

This is a shortcut for producing row vectors

```
>> V1=1:4
V1 =
    1     2     3     4
```

More generally a: b: c produces a vector of entries starting with the value a, incrementing by the value b until it gets to c (it will not produce a value beyond c).

```
>> V2=1:2:10
V2=
     1     3     5     7     9
```

N.B. If we define Vx as a complex vector, then Vx' gives the complex conjugate transpose of Vx

```
>> Vx = [1+3i, 2-2i]
>> Vx'
>> Vx.'
```

(give comment on the results)

5.2 Transposing

We can convert a row vector into a column vector (and vice versa) by a process called transposing donated by (').

```
>> Vr'
>> Vc'
```

(give comment on the results)

5.3 Arithmetic operations

We can do certain arithmetic operations (+,-,*) with vectors of the same length and same type (i.e. row or column vectors).

```
>> V1=[2, 4, 6]; V2=[10, 20, 30]; V3=[100, 10, 31, -31 , 24];
```

Tryout (1) V1+V2, 3*V1, V2/10, sort (V3).
Tryout (2) V1*V2, V1.*V2, V2./V1 (give comment on the results)

5.4 Scalar product (*)

Suppose that u and v are two vectors of the same length n. The scalar product is defined by multiplying the corresponding elements together and adding the results to give a single number *(scalar)*.

$$u = [u_1, u_2, \ldots, u_n] \quad , \quad v = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix},$$

$$uv \quad = \quad \sum_{i=1}^{n} u_i v_i$$

For example, if $u = [10, -11, 12]^T$, and $v = [20, -21, -22]$, then $n = 3$ and

$$uv = 10 \times 20 + (-11) \times (-21) + 12 \times (-22)$$

We can perform this product in MATLAB using scalar product operator(*)

```
>> u = [ 10, -11, 12], v = [20; -21; -22]
>> prod = u*v
```

Norm

We shall refer to the Euclidean length of a vector as the norm of a vector; it is denoted by the symbol $||u||$ and defined by

$$\|u\| = \sqrt{\sum_{i=1}^{n} |u_i|^2},$$

where n is its dimension. This can be computed in MATLAB in one of two ways:

```
>> [sqrt(u*u'), norm(u)]
ans =
19.1050  19.1050
```

Where norm is a built in MATLAB function that accepts a vector as input and delivers a scalar as output. It can also be used to compute other norms for more information *help norm*.

5.5 Dot product (.*)

Suppose that u and v are two vectors of the same type (both row vectors or both column vectors), the mathematical definition of the dot product is the vector having the components

$$uv = [u_1v_1, u_2v_2, \ldots, u_nv_n]$$

The result is a vector of the same length and type as u and v. Thus, we simply multiply the corresponding elements of two vectors. In MATLAB, the product is computed with the dot product operator (.*)

```
>> u .* v
```

5.6 Dot Power (.^)

To square each of the elements of a vector we could, for example, do (u.*u). However, a neater way is to use the dot power operator (.^):

```
>> u.^2
>> u.*u
>> u.^3
```

5.7 Dot Division (./)

There is no mathematical definition for the division of one vector by another. However, in MATLAB, the operator (./) is defined to give element by element division. It is therefore only defined for vectors of the same size and type.

```
>> a = 1:5, b = 6:10, a./b
a =
     1     2     3     4     5
b =
     6     7     8     9    10
ans =
    0.1667    0.2857    0.3750    0.4444    0.5000
>> a./a
ans =
     1     1     1     1     1
>> c = -2:2, a./c
c =
    -2    -1     0     1     2
ans =
   -0.5000   -2.0000       Inf    4.0000    2.5000
```

The previous calculation required division by 0, notice the Inf (infinity) in the answer.

```
>> a.*b -24, ans./c
ans =
   -18   -10     0    12    26
ans =
     9    10   NaN    12    13
```

Here we are warned about 0/0, giving a NaN (Not a Number).

Exercise 4

Enter the vectors $U = [6, 2, 4], V = [3, 2, 3, 0]$, W $= \begin{bmatrix} 3 \\ -4 \\ 2 \\ 6 \end{bmatrix}$, Z $= \begin{bmatrix} 3 \\ 2 \\ 2 \\ 7 \end{bmatrix}$ into Matlab.

Which of the products U*V, V*W, U*V', V*W', W*Z', U.*V , U'*V, V'*W, W'*Z, U.*W, W.*Z, V.*W is legal?
Mark the illegal products by setting the values of the variables corresponding to these products to 0. **It should be noted here that Matlab versions of 2016 and higher have some "intelligence" that even gives results for "impossible" calculations. E.g. U.*W is the dot product of a rowvector with a columnvector – to make things worse, with different lengths n. This should not give a result, but as you will see, Matlab gives some output. Bottomline, always use you own intelligence over that of Matlab.**

6 Plotting

6.1 Plotting Elementary Functions

Suppose we wish to plot a graph of $y = sin3\pi x$ for $0 \leq x \leq 1$. We do this by sampling the function at a sufficiently large number of points and then joining up the points (x, y) by straight lines. Suppose we take $N+1$ points equally spaced a distance h apart:

```
>> N = 10; h = 1/N; x = 0:h:1;
```

defines the set of points $x = 0, h, \ldots, 2h, , 1-h, 1$. Alternately, we may use the command ***linspace:*** The general form of the command is ***linspace (a,b,n)*** which generates n equispaced points between a and b, inclusive. So, in this case we would use the command

```
>> x = linspace (0,1,11);
```

The corresponding y values are computed by

```
>> y = sin(3*pi*x);
```

and finally, we can plot the points with

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

```
The result is shown in Figure 1, where it is clear that the value of N is too small.
```

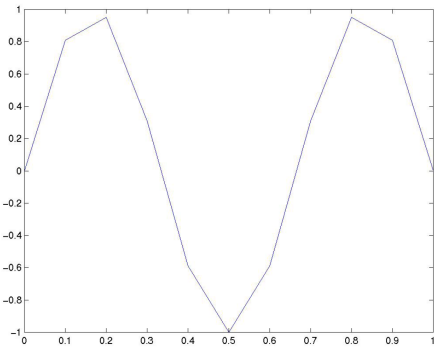


Figure 1: Graph of $y = sin3\pi x$ for $0 \leq x \leq 1$ using $h = 0.1$

On changing the value of N to 100:

```
>> N = 100; h = 1/N; x = 0:h:1;
>> y = sin(3*pi*x); plot(x,y)
```

we get the picture shown in Figure 2.

6.2 Plotting (Titles & Labels)

To put a title and label the axes, we use

```
>> title('Graph of y = sin(3pi x)')
>> xlabel('x axis')
>> ylabel('y-axis')
```

The strings enclosed in single quotes, can be anything of our choosing. See also *help* ***ezplot*** "the Easy to use function plotter".

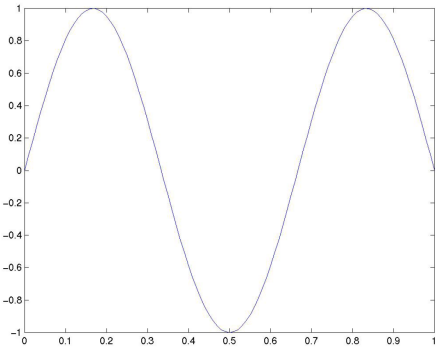


Figure 2: Graph of $y = sin3\pi x$ for $0 \leq x \leq 1$ using $h = 0.01$

6.3 Grids & axes display range

A dotted grid may be added by

```
>> grid
```

This can be removed using either grid again, or grid off. Also you can control the axes display range by using xlim([xmin xmax]) and ylim([ymin ymax])

```
>> xlim([-2 2])
>> ylim([-2 2])
```

6.4 Line-Styles & Colors

The default is to plot solid lines. A solid white line is produced by

```
>> plot(x,y,'w-')
```

Colours		Line Styles	
y	yellow	.	point
m	magenta	o	circle
c	cyan	x	x-mark
r	red	+	plus
g	green	-	solid
b	blue	*	star
w	white	:	dotted
k	black	-.	dashdot
		--	dashed

The third argument is a string whose first character specifies the color(optional) and the second is the line style. The options for colors and styles are: The number of available plot symbols is wider than shown in this table. Use *help* ***plot*** to obtain a full list. See also *help* ***shapes***.

6.5 Multi-plots

Several graphs may be drawn on the same figure as in

```
>> plot(x,y,'w-',x,cos(3*pi*x),'g--')
```

A descriptive legend may be included with

```
>> legend('Sin curve','Cos curve')
```

which will give a list of line-styles, as they appeared in the plot command, followed by a brief description. MATLAB fits the legend in a suitable position, so as not to conceal the graphs whenever possible. For further information do *help* ***plot***.
The result of the commands

```
>> plot(x,y,'w-',x,cos(3*pi*x),'g--')
>> legend('Sin curve','Cos curve')
>> title('Multi-plot ')
>> xlabel('x axis'), ylabel('y axis')
>> grid
```

is shown in Figure 3. The legend may be moved manually by dragging it with the mouse.

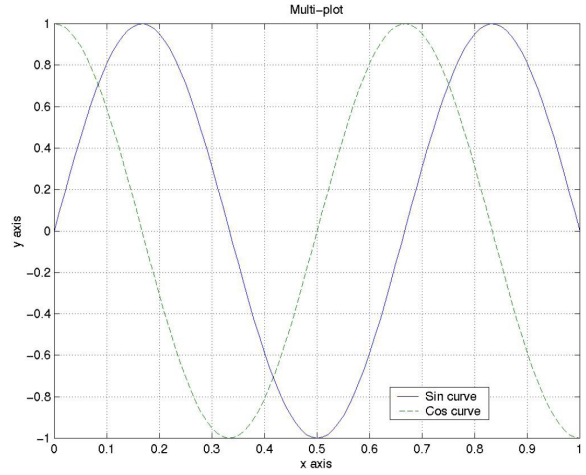


Figure 3: Graph of $y = sin3\pi x$ and $y = cos3\pi x$ for $0 \leq x \leq 1$ $h = 0.01$

6.6 Hold

A call to plot clears the graphics window before plotting the current graph. This is not convenient if we wish to add further graphics to the figure at some later stage. To stop the window being cleared:

```
>> plot(x,y,'w-'), hold on
>> plot(x,y,'gx'), hold off
```

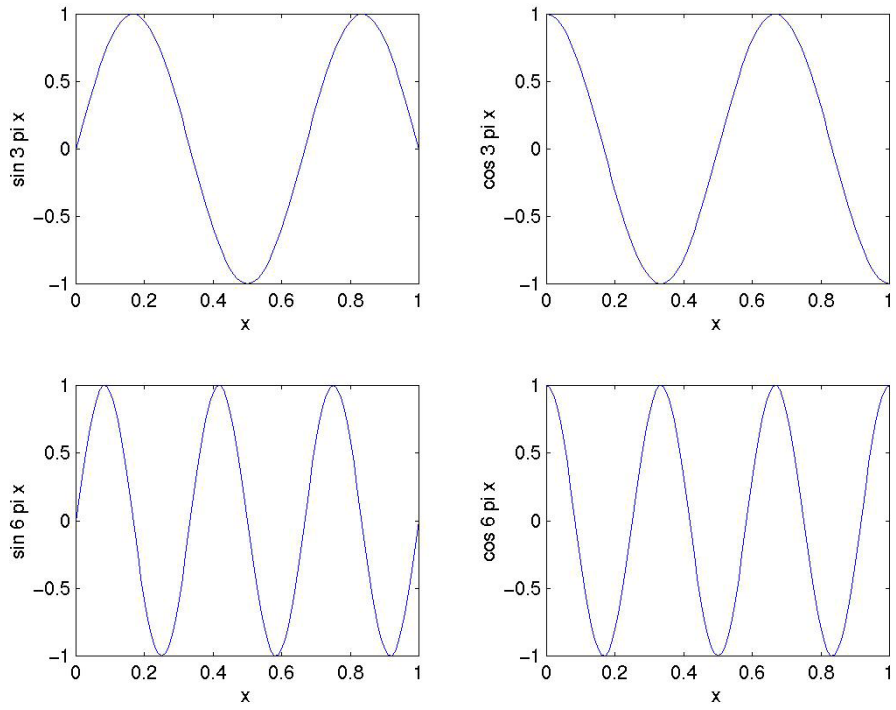
"hold on" holds the current picture; **"hold off"** releases it (but does not clear the window, which can be done with *clf*). **"hold"** on its own toggles the hold state.

6.7 Subplot

The graphics window may be split into an $m \times n$ array of smaller windows into which we may plot one or more graphs. The windows are counted 1 to mn row-wise starting from the top left. Both hold and grid work on the current subplot.

```
>> subplot(221), plot(x,y)
>> xlabel('x'),ylabel('sin 3 pi x')
>> subplot(222), plot(x,cos(3*pi*x))
>> xlabel('x'),ylabel('cos 3 pi x')
>> subplot(223), plot(x,sin(6*pi*x))
>> xlabel('x'),ylabel('sin 6 pi x')
>> subplot(224), plot(x,cos(6*pi*x))
>> xlabel('x'),ylabel('cos 6 pi x')
```

subplot(221) (or subplot(2,2,1)) specifies that the window should be split into a 2×2 array and we select the first subwindow.



Further Commands

Graphics & plotting.	
`figure`	Create Figure (graph window).
`clf`	Clear current figure.
`close`	Close figure.
`subplot`	Create axes in tiled positions.
`axis`	Control axis scaling and appearance.
`hold`	Hold current graph.
`figure`	Create figure window.
`text`	Create text.
`print`	Save graph to file.
`plot`	Linear plot.
`loglog`	Log-log scale plot.
`semilogx`	Semi-log scale plot.
`semilogy`	Semi-log scale plot.

Graph annotation.	
`title`	Graph title.
`xlabel`	X-axis label.
`ylabel`	Y-axis label.
`text`	Text annotation.
`gtext`	Mouse placement of text.
`grid`	Grid lines.
`contour`	Contour plot.
`mesh`	3-D mesh surface.
`surf`	3-D shaded surface.
`waterfall`	Waterfall plot.
`view`	3-D graph viewpoint specification.
`zlabel`	Z-axis label for 3-D plots.
`gtext`	Mouse placement of text.
`grid`	Grid lines.

Exercise 5

Write a script that makes a subplot in a 2x2 array with the functions given below. The subplot array should look like: [1, 2; 3, 4], with 1, 2, 3 and 4 corresponding to the different plots. All plots should be made with the normal 'plot' function (with a linear axis scale). The domain is specified as [0, 10] containing 101 data points. The subplots should fulfill the following requirements:

Plot	Function	Y-Label	Grid	LineColor	LineStyle
1	y	y(x)	on	magenta	:
2	u	u(x)	off	blue	--
3	v	v(x)	off	red	-.
4	w	w(x)	on	green	-

If performed correctly, the figure should look exactly like the figure on the next page.

$$y = \frac{\sin x}{x} \tag{1}$$

$$u = \frac{1}{(x-1)^2} + x \tag{2}$$

$$v = \frac{x^2+1}{x^2-4} \tag{3}$$

$$w = \frac{(10-x)^{1/3} - 1}{(4-x^2)^{1/2}} \tag{4}$$