Intro to programming with MATLAB

* Smartphone - features - with help of MATLAB.
* NASA - matlab

Book: Computer programming with MATLAB - J. michael Fitzpatrick & AKOS Ledeczi

---

week: 1: The MATLAB environment

Introduction

* first - high-level-language: Fortran (1954): Easy to solve numerical problems.
* "Shrinkwrapped product" - used by nonprogrammers - word, spreadsheet, OS, Games etc... → C, C++, Java, C#, etc.
* MATLAB: numerical problem easy than FORTRAN.

Invented: cleve moler

"Concentrate on problem not Coding"

Vocabulary & Idea: Took more than 50 years to come up

* Stable (Today won't work tomorrow)
* New features
* NERD - Naturally Endowed for Research & Development.

https://matlab.mathworks.com

---

matlab desktop

* GUI - Graphical User Interface.
* prompt - Symol(s) - Indicate that it is ready for i/p from the user.

" >> "

format compact

```
>> x = 2+1
x = 3

>> format compact
>> x = 1+2
>> x = 3
```

No vertical/useless space (Compact o/p)

"command history" → set as dock (For seperate window)

* Layout → Command history → docked

* clc - clear command window

* Formats

* See via search documentation

* Regain control - when too much time for calculating.

Stop Ctrl + c

quit → matlab will be closed

Variable: Varying value. [Location in memory with name & value]

---

* Function: (math: Any operation produces an o/p)

*

>> grand

ans = 0.8147 → Automatically created one.

workspace order

↓

Alphabetical.

Delete the workspace

>> clear ans (or) right click & delete.

>> clear (clears all variables)

MATLAB has autocompletion & description

(Function hint) - what i/p is

Root - Topmost Directory

Up → Towards parent (up to one level)

.m → Extension

% → Comments

* Save before running in Command window.

my files

https://drive.matlab.com

```
>> speed = (3.8*10^8) * 365*24*60*60
speed = 9.4608e+12 → Trillion (10^12)
>> earth_to_sun = 150e6
>> min = speed/earth_to_sun
500
```

earth_moon = 384400Km

speed of light = 300000 Km/sec

Load

>> load [From local directory matlab file]

Save

>> save

---

* 'variable - Start with alphabet or underscore'
* Digits/ underscore / alphabets
* Length ≤ 63

>> x = 12 ; → don't show in command window [semicolon]

* multiple commands on a single line → semicolon.

```
>> x = 15; y = 12;
>> x = 15 , y = 12;
x = 15
```
└→ No semicolon.

one command on multiple lines

(3 fullstops)

```
>> hello = 33 * ...
4325;
```

Syntax: set of rules

```
>> x = 1
```

Target must be a variable

Semantics: meaning → Not intented code.

x = 1, y = 2

```
>> x = y; y = x;
```

x = 2, y = 2

```
>> x = 1; y = 2;
>> temp = x;
>> x = y;
>> y = temp;
```

'x value lost'

x = 2, y = 1 → swapped.

'Semantics error is the worst: No way of knowing it is an error.

'Help'

» help format » format loose (more line spaces)

» format long (15 digits - decimal - accuracy)

» doc format [documentation]

click f_x for seeing function (below help)

Plotting

```
» x = [1, 3, 10];
» y = [2, -4.2, 12.3];
» plot (x, y);
```

```
» length (x)
ans = 3
```

plot (x, y, '*') ; → stars instead of line.

'-' → default → single dashed line '--', ':', '-.' → line style o, +, *, ., x, s, d → pointer shape

```
» plot (x, y, 'rs');
```
r → red, s → square

```
grid on
xlabel ('selection')
ylabel ('change')
title ('changes in selections')
axis ([0, 12, -10, 20])
      x     y
```

» bar (x, y) → bar graph

» figure → open figure window (Any new plot will be there)

» pie ([4 2 7 4 7])

Close figure 1

» close (1);

» close all → all will be closed.

open image

```
>> imread('img_name.format');
```

'palace.JPG'

```
>> picture = imread('img_name.format');
```

↳ 3456 x 4608 x 3 uint8

See an image

```
>> axis off   → No axes
>> image(pretty_image)
>> quit.
```

$$speed = \frac{distance}{time}$$

2018 → Usain Bolt → 100m → 9.58 seconds

2018 → Kipchoge → 42.195 Km → 2:01:39

```
hundred = (100*(1/1000))/(9.58*(1/3600))
marathon = 42.195/((2*3600)+(1*60)+39)*(1/3600))
```

Matrices & operators

matrix: 2d array

Array: set of numbers in a rectangular pattern

3d: stack (2 pages: Row & column each page)

6rows, 4col, 3pages

vector: 1d array.

matlab - Matrix laboratory

```
>> a = [1,2,3]        → row vector
>> b = [1,2,3;4,5,6]  → 2x3 matrix
```

functions

```
>> sqrt(2)    → 1.4142
>> sin(30)    → -0.9880 (radians)
>> sind(30)   → 0.5000 (degrees)
>> size(a)    → 3
>> size(b)    → 2 3
```

```
>> x = 5;
>> size(x)
 1   1
```

'matlab looks everything as matrices'

Custom

Small letters → vectors
Capital → matrices

>> a = [1,2,3]

'Row vector'

(or)

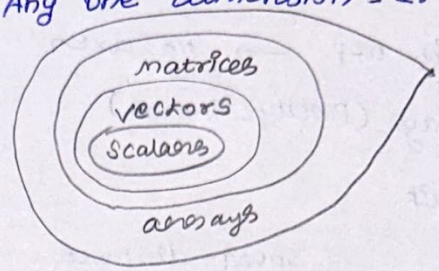
>> a = [1 2 3]

'b = [1;2;3]

'Column vector'

Scalar - vector with both dimensions = 1

vector - Any one dimension = 1.



Colon operation

>> x = 1 : 3 : 7 → End.
→ Start
→ Interval

x = 1 4 7

(operands)

'seperated by operation'

>> plus (2, 3) — function (Arguments)

ans = 5

>> Colon (1, 7)

1 2 3 4 5 6 7

Inside parenthesis

>> Colon (1, 3, 10)

1 4 7 10

odd numbers

1 : 2 : 1000

1, 3, 5, ... , 999

Even numbers

2 : 2 : 1000

Downwards

>> 7 : -3 : 1

7 4 1

Empty matrix

>> x = []

>> x = 7 : 3 : 1 → 'Never achieved'

Size

0 0

Size

1 0

'xyz not exist'

>> XYZ(2,2) = 2

>> XYZ

0 0
0 2

'matrix will be created'

>> odds = 1:2:100

>> evens = 100:-2:2

Access matrix

x = [1:4 ; 5:8 ; 9:12];

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$$

>> x(2,3)

7

updatable

>> x(2,3) = 97

>> x =

1 2 3 4
5 6 97 8
9 10 11 12

>> x = 1) matrix no longer exists.

x = 1

$$y = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

>> y(3,3) = 5

1 2 0
3 4 0
0 0 5

'Extented'

Subarray

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

>> x(2, [1 3 4])

ans : 5 7 8 → column 1, 3, 4

>> x([2,1], [3,4,2])

6 4 4 5
3 1 1 2

2nd row then 1st row

3 col, 1 col, 1 col, 2 col

x = 1 2 3
4 5 6

>> x([1,2], 4)

ans : 4
8

>> x([2,1], 3)

ans : 7
3

condition: Index must exist in the matrix

>> x(2, 1:3)

↳ up to 1 to 3 column

>> x(2:-1:1, 3:-1:1)

↓ 2, 1 ; ↓ 3, 2, 1 column.

>> x(end, 2) [end = 2 (here)]

5

'end' → Reserved word

>> X(end-1, [end, end-1])

↓ 1 row ↓ 3 Col ↓ 2 Col

>> X(1:end, 2:3) = [10 20; 30 40; 50 60]

2	10	20
2	30	40
2	50	60

All rows / columns

: → 1:end

>> X(:, 1)

All rows of 1 column

>> X(1:end, 1) = 2

↓

All rows, 1st column will be 2.

No mismatch b/w indices

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \end{bmatrix}$$

V = A(:,2) → 2nd Column of A

A(end,:) = 0 → Set last row of A to 0

Combine matrices

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

>> A = [1 1 1; 1 1 1]

>> B = [2 2 2; 2 2 2]

>> C = [3 3 3; 3 3 3]

>> D = [A B C]

A	B	C	
1 1 1	2 2 2	3 3 3	→ Same rows.
1 1 1	2 2 2	3 3 3	

>> E = [A; B; C]

1 1 1	
1 1 1	→ Same columns.
2 2 2	
2 2 2	
3 3 3	
3 3 3	

>> A1 = [1 1]

>> A2 = [1 1 1]

) No Same Columns

>> [A1; A2] → Error

'vertcat' – vertical catenation

'stack matrices'

) Same columns.

else error.

Transposition:

$$H = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, \quad H' = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

2×3 $\qquad$ 3×2

'Rows 9n to columns', 'columns 9n to Rows'

>> H' → matlab code

>> 1:2:5' ∴ 5' = 5

1 3 5

>> a = 1:2:5]
>> a = a'] column vector.

Array addition

'Same dimension'

$$x = \begin{bmatrix} 1 & 4 \\ 7 & 0 \\ 5 & 5 \end{bmatrix}, \quad y = \begin{bmatrix} 2 & -4 \\ 6 & 2 \\ 0 & 3 \end{bmatrix}, \quad z = x + y \Rightarrow z = \begin{bmatrix} 3 & 0 \\ 13 & 2 \\ 5 & 8 \end{bmatrix}$$

>> z = x + y

we can't add 2x3 matrix & 1x6 matrix

>> z = x - y

Array multiplication

z = x .* y

.* element wise

$$z = \begin{bmatrix} 2 & -16 \\ 42 & 0 \\ 0 & 15 \end{bmatrix}$$

$$\text{>> meas} = \begin{bmatrix} 71.0001 & 52.4010 & 78.1818 \\ 83.6957 & 78.6214 & 59.8462 \end{bmatrix}, \quad \text{>> Caleb} = \begin{bmatrix} 1.1 & 1.5 & 0.99 \\ 0.92 & 1.00 & 1.33 \end{bmatrix}$$

'mul each meas w.r.t their Caleb'

>> meas .* caleb

$$\begin{bmatrix} 78.1001 & 78.6015 & 77.4000 \\ 77.0837 & 78.7000 & 77.8001 \end{bmatrix}$$

matrix multiplication

'shape : compatible' – uses both mul & add.

$(L \times M)(M \times N) = (L \times N)$

'Inner dimensions must be equal'

>> [size(A), size(B)]

4 3 3 2 → compatible.

>> C = A * B

Array division

X ./ Y & X .\ Y

over under

$$X = \begin{bmatrix} 1 & 4 \\ 7 & 1 \\ 5 & 5 \end{bmatrix}, \quad Y = \begin{bmatrix} 2 & -4 \\ 6 & 2 \\ 1 & 3 \end{bmatrix}$$

Z = X ./ Y

$$= \begin{bmatrix} 1/2 & 4/-4 \\ 7/6 & 1/2 \\ 5/1 & 5/3 \end{bmatrix} = \begin{bmatrix} 0.5 & -1 \\ 1.1667 & 0.5 \\ 5 & 1.667 \end{bmatrix}$$

Z = X .\ Y

$$Z = \begin{bmatrix} 1 \backslash 2 & 4 \backslash (-4) \\ 7 \backslash 6 & 1 \backslash 2 \\ 5 \backslash 1 & 5 \backslash 3 \end{bmatrix}$$

$$= \begin{bmatrix} 2/1 & -4/4 \\ 6/7 & 2/1 \\ 1/5 & 3/5 \end{bmatrix} = \begin{bmatrix} 2 & -1 \\ 0.8571 & 2 \\ 0.2 & 0.6 \end{bmatrix}$$

Z = X ./ Y | Z = X .\ Y

Z = X / Y | Z = Y ./ X

Array exponentiation

>> 2 ^ 3

8

$$X = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad N = \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}$$

= X ^ 3 = $\begin{bmatrix} 1 & 8 \\ 27 & 64 \end{bmatrix}$

X .^ N = $\begin{bmatrix} 1 & 8 \\ 9 & 16 \end{bmatrix}$

= X * X * X

>> 2 .^ A (valid) → Piecewise)

>> 2 ./ A → piecewise valid.

>> 2 * A
>> 2 .* A
) same

>> 3 + A (Add 3 to all ele)

>> A / 2 → OK

>> 2 / A or >> A \ 2 → error (dimension)

1 → Scalar & a square matrix.

A = [1:5 ; 6:10 ; 11:15 ; 16:20];

row = ones(1,4) → Same no. as rows
col = ones(5,1) → Same no. as columns
) vectors

result = row * A * col → 210

(n,1) * (1,n) = (n,n)

Functions

>> 1+rand(3,4)*9 → range (1 to 10)

(or)

>> randi([1,100],5)
↓ ↓
Range 5×5 matrix

open editor

>> edit

```
function myRand
  a = 1+rand(3,4)*9
end
```

→ save then run → Type name in command window → It will run

myRand.m

Functions

"function has a private workspace, what happens in that workspace remains there." (Local)

↓

Don't reflected in Command Window workspace

```
function a = myRand
  a = 1+rand(3,4)*9
end
```

→ a = 3×4 matrix
ans = 3×4 matrix
) same. [won't be reflected in Command Window workspace]

∴ "First for a then for the function" printed twice.

```
function a = myRand
a = 1+rand(3,4)*9; → Don't print a
end
```

o/p : >> myRand

ans =

3×4 matrix

>> b = myRand

b =

3×4 matrix

b/w 2 and 5

>> 2 + rand (3,4) * 3

Take Arguments

function a = myRand(low,high) → must open assigning in command wind.

a = low + rand (3,4) * (high - low);

end

↓

user defined range

'Two scalars as i/p & a matrix as o/p'

Two o/p s

function [a, s] = myRand (low, high)

a = low + rand (3,4) * (high - low);

v = a(:);

s = sum(v);

end

a(:)

: → 1:end.

when only one : is given, matlab considers all the columns are stacked up into a single one.

>> myRand (2,5)

ans =

3×4 matrix

→ No s

∴ only one o/p is taken

If we need both o/p

>> [a, s] = myRand (2,5)

a =

3×4 matrix

s =

30.2078

using array we can get more o/ps

↓

'Here : 2 o/p s'

```
function    area = tri_area(b,h)
    area = 1/2 * b * h;
end
```

→ Area of a Δle

Formal definition

function [out_Arg1, ..., out_Argn] = function_name(in_arg1, ... in_argm)

end

in_argm → optional.

[out_Arg1, ..., out_Argn] → single/multiple o/p s

Function names

"Use meaning full names that tells you something about what your function does" : Stay away from builtin function names

sqrt, plot, sin, etc...

>> help exist

Sub functions

```
function [a,s] = myRand(low,high)
a = low + rand(3,4) * (high-low);
s = Sum_array(a);
end
function S = Sum_array(a)
v = a(:);
S = Sum(v);
end
```

(or) we can write function inside a function.

Scope

"we can't access any variable of functions in functions"

↓

"Local scope" – only valid for a function (Inside)

Scope: set of statements that can access a variable.

(only one function)

Global variable

```
global V;
V = M(:);
```
) we can't combine like `global V = M(:);`

↓

Now we can access from anywhere.

Any change in anywhere affects V : It is global : accessible from anywhere.

"Global variables can cause errors" – Difficult to find Ghost moving around the furniture.

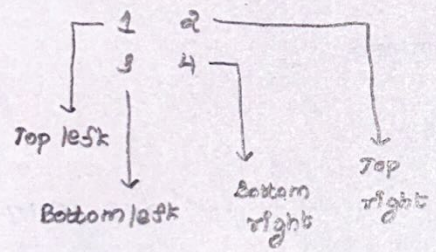
Advantage of Functions

* Break down large - complex problems – manageable piece
* Decompose – Reuse.

"Libraries" – lot are available

↓

"opensource"




```
function [a,b,c,d] = corners(mat)
a = mat(1,1);
b = mat(1,end);
c = mat(end,1);
d = mat(end,end);
end
```

```
A = randi(100,4,5);
[a,b,c,d] = corners(A)
B = [1;2]
[a,b,c,d] = corners(B)
```

```
>> x = 5;
```

script

example.m

```
y = x+3;
z = 2*y;
```

```
>> example
>> y
y = 8
```

Shares same workspace.

Scripts : collection of matlab commands.

↓

* Never returns / i/p s anything
* scope of script = scope of Command window
* >> edit example_script

`pause(5);` → pause for 5 sec before further execution.

1st Km → $5
Next → $2
1 minute waiting time → $0.25

] use ceil (Take as whole numbers)

```
function fare = taxi_fare (d,t)
d = ceil(d);
t = ceil(t);
fare = (5+(d-1)*2) + (t*0.25);
end
```

] No use of if/else or loop.

Suppose 0.75 Km, No waiting time

d = 1 → t = 0

5 + 0 + 0 = 5 dollars

Lesson : 4 : Programmer's Toolbox

```
>> sqrt(9)
>> sqrt([1 4; 9 16; 25 36])
>>
```

Polymorphism

If the type of an i/p argument to a function can vary from one call to another, it is called as polymorphic function.

'Multiple forms' - saves huge amount of work.

* many functions : empty - returning o/p that is shaped like the i/p
* one function : handles huge variety of i/p s

Polymorphism
- Type
- No. of arguments

eg: 2×3 matrix → 2×3 matrix [sqrt( )]

'Shape shifting'

eg:

```
>> V = [1 2 3 -4 7]
   Sum(V) → 9
>> A = [1 2; 3 4]
>> Sum(A)
    4   6
```

] Not polymorphic but o/p has different shape!

$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$

$A(:) \longrightarrow \begin{matrix} 1 \\ 3 \\ 2 \\ 4 \end{matrix}$

$A(1) = 1$
$A(2) = 3$
$A(3) = 2$
$A(4) = 4$

: → 1:end

A(1:end)

Auto varying

1 1
1 2
2 1
2 2

[a b] = max([1 2 -4 8])

a = 8 → value
b = 4 → Index

```
>> size([1 2; 9 8; 0 -2])
>> size =
     3     2
```

```
>> [row col] = size([1 2; 9 8; 0 -2])
row = 3
col = 2
```

---

matrix building

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

```
>> zeros(5,6)
     0   0   0   0   0   0
     0   0   0   0   0   0
     0   0   0   0   0   0
     0   0   0   0   0   0
     0   0   0   0   0   0
```

```
>> ones(4,2)
     1   1
     1   1
     1   1
     1   1
```

```
>> 5*ones(4,2)
     5   5
     5   5
     5   5
     5   5
```

```
>> zeros(2)
     0   0
     0   0
```

```
>> diag([7 3 9 1])
     7   0   0   0
     0   3   0   0
     0   0   9   0
     0   0   0   1
```

Polymorphism
'works with two as well as one argument'

```
>> rand
ans =
    0.8147
```

↓

uniformly distributed
(same probability)

```
>> rand(5)
```

5×5 matrix

1 + rand(5,4) * 10 → (1 - 11 range)

```
>> a = [ 9.1472   1.9754   2.5761
        10.0579   3.7850   5.2176
         9.4913   7.3236   9.0028 ]
```

```
>> fix(a)
     9     1     2
    10     3     5
     9     7     9
```

(or)

>> grandi(10, 5, 4) → polymorphic

↓ RxC

Range

>> randi([5,10], 2, 3)

Distributions

Gaussian or bell curve

>> randn(5)

-1.1564	-0.0200	-0.7145	-0.8479	-1.2571
⋮	⋮	⋮	⋮	⋮

$-\infty$ to ∞

↓

S.D = 1

↓

Clustered about 0

>> randn(1, 10000) → row vector.

>> hist(randn(1, 10000), 100)

↳ Bends.

when we start matlab – call rand – Always gives 0.8147

(Random sequence initialization)

Pseudo random: Computer: deterministic: By Algorithm: uses fancy math to produce pseudo random numbers.

(No discernible way to predict): Unpredictability.

Same i/p : change program

reinitialize that number without restart

rng → useful in debugging

(I need to give same i/p)

rng - random number generator

```
>> rand(1,3)
0.8147   0.9058   0.1270
>> quit
```

```
>> rand(1,3)
0.8147   0.9058   0.1270
```

Same.

```
>> rng(0);
>> rand        → 'Same seed' → same number → 1st time
ans = 0.8147
```

more random – Input a string

```
>> rng('shuffle');
>> rand
  0.5099
```

```
>> rng('shuffle')
>> rand
  0.1168
```

Same string but random o/p

'uses current reading of system clock to generate truly random numbers'
(microsecond accuracy: Guarantees the randomness)

---

minimax takes M, return

mmr → row vector → max − min of a row
mms → max − min element of matrix M

$$A = \begin{bmatrix} 10 & 5 & 6 \\ 2 & 3 & 7 \end{bmatrix}$$

mmr = [5 5]

mms = 8

```
function [mmr, mms] = minimax(M);
mmr = max(M') - min(M');
mms = M(:);
mms = max(mms) - min(mms);
end.
```

Take Input

```
function a = onemore
x = input('Give a number: ');
a = x + 1;
```

↓ String.

```
>> one more
>> Give a number: 2
ans =
   3
```

↓ matlab legal (must)

'matlab evaluates then gives o/p'

```
>> fprintf('Hello')
Hello
```

↓ format string

\n → newline

>> fprintf('This is the end')

This is the end >> ⟶ 'No new line' – So continues.

% escape characters + format specifiers

- % .2f → precision (2 points)
- % 5.2f →
- % d ⟶ Integer

f – conversion characters (fixed point notation) – normal instead of scientific.

>> a=3
>> fprintf('%.3f', a)
3.000
'3 digits after point'

\n , % .2f ⟶ escape characters

% 5.2f → print using atleast 5 spaces

>> fprintf('Total = $ %5.2f', total)

Total = $ 8.13

8.13 — 4 spaces, 5 spaces, 1 Blank

5 spaces (no space given by fprintf)

Print percentage sign

>> fprintf('12.5 % % of 1234 equals %.3f \n', 0.125*1234)

12.5% of 1234 equals 154.250

>>

\\n → 'Backslash' char with new line → \

'' ⟶ single quote ⟶ '

>> fprintf('%d %d %d %d \n', 1,2,3,4)

1 2 3 4

>> fprintf('%d %d %d %d \n', 1,2,3)

1 2 3 >>

'Not enough arguments for completion'

>> fprintf('%d %d %d %d', 2,3,4,5,6).

2 3 4 5 6 → Extra arguments skipped from first

>> fprintf('%d h %d e', 1,2,3)

1 h 2 e 3 h

↳ start over again.

```
>> fprintf('%4.1f\n', [1 2 3 4 5 6]);
 1.0
 2.0
 3.0
 4.0
 5.0
 6.0
```

↓ 4 Spaces, 1 digit precesion.

Plotting

```
>> a = (1:10).^2;
>> plot(a)
>> b = (-10:10).^2
```

"picks a range automatically"

(y axis → gn)

New figure

```
>> figure(2)
>> plot(b)
```

```
>> t = -10:10
>> plot(t, b)
```

more plots

```
>> x1 = 0:0.1:2*pi ; y1 = sin(x1)
>> x2 = pi/2 : 0.1 : 3*pi ; y2 = cos(x2)
```

```
>> hold on
>> plot(x1, y1)
>> plot(x2, y2)
>> hold off
```

(or)

```
plot(x1, y1, x2, y2)
```

Default: different colors.

Colors

```
plot(x1, y1, 'r', x2, y2, 'K:')
```

'r' ↓ red. Solid line (Default)

'K:' ↓ Black dotted

```
>> help plot
```

---

b - blue
g - green
r - red
c - cyan
m - magenta
y - yellow
K - black
w - white

. point
o circle
x (x-mark)
* star
s square
d diamond
v down triangle
^ up triangle
>) Right, left Δle
<

p - pentagram
h - hexagram

-	Solid
:	dotted
-.	dashdot
--	dashed
(none)	no line.

'm--o' → magenta, double dashed, circle pointers

Additional plotting options

>> grid (##) appears
>> grid (turns it off)
>> title (' ')
>> xlabel (' ')
>> ylabel (' ')

>> legend ('sine', 'cosine')
↓ ↓
x_1, y_1 x_2, y_2
give with comma

>> help legend

>> axis ([-2 12 -1.5 1.5]) → Affects both axes
>> close (1) → closes figure 1.
>> close all → all will be closed.

Debugging

* Syntax error: MATLAB catches these
* Semantic error: may/may not wrong result { occassional problems, Hard to notice & find }

MATLAB has inbuilt debugger. — Tool to find bugs

Some mistakes:
>> 1 = x → error
>> rand (2) → error
'Index exceeds'

>> y(6) → Runtime error (while execution)
Semantics error.

'During debugging: we can see function workspace'
Stops at break point!

>> 2 x = randi(n,m);
↓ ↓
2nd line Statement

K >> m
m = 2 → Function is still running. Accessing function workspace

K >> m = pi → we can also change.

```
function x = rand_int(n,m)
x=randi(n,m);
fprint('The last element in the last row is %d.\n', x(n,m));
end
```

```
>>rand_int(3,3)
. . .
. . .
. . .
```

```
>> rand_int(3,2)
. .
. .
. .
```
→ 2,2

(3,2) → Index error

why

randi(3,2)
↓ └→ 2×2
Range
(max)

>>db quit

"problem is here with Randi"

Fix:

x = randi(10,n,m)
↓ ⌣
max └→ indices

→ grey colon break point (Inactive)

"Debugger: most useful one"

trio → function → takes n & m [+ve integers]

returns 3n by m matrix called T

Top third → 1s
middle → 2s
Bottom → 3s

```
>> M = trio(2,4)
M =
```

1	1	1	1
1	1	1	1
2	2	2	2
2	2	2	2
3	3	3	3
3	3	3	3

```
function T=trio(m,n)
T= [ones(n,m); 2*ones(n,m); 3*ones(n,m)];
end
```

Lesson : 5 : Selection

* Sequential flow : until now : Default.
* Selection / Branching. ⟶ If Statement

```
if ____
   :
end
```

```
if ____
   ____
else
   ____
end
```

```
if ____
   ____
else if ____
   ____
end
```

```
function guess(n)
if x==2
    fprintf('congrats \n');
else
    fprintf('Keep trying \n');
end
```

```
function guess(n)
if x==42
    fprintf('congrats! \n');
else if x<42
    fprintf('Try bigger');
else
    fprintf('Too big \n');
end
```

return ⟶ quits without further execution

what day - Is it weekend ? → Github/matlabonline

Relational operators

* produces o/p based on the relation b/w its two operands.

>= greater than or equal to	== ⟶ equal to	> greater than
<= less than or equal to	~= ⟶ Not equal to	< less than

(16 * 64 > 1000) ⟶ True ⟶ 1

>> x = 16 * 64 > 1000 + 9

>> x

x = 1

∴ (16 * 64 > 1009)

```
function if_test(x)
if x
    fprintf('True \n');
else
    fprintf('False \n');
```

» if_test(0)

False

» if_test(1)

True

» if_test(-2)

True

Anything except 0 is True

0 → false.

» if_true(0.00000001)

True

>> [4 -1 7 5 3] .* [5 -9 6 5 -3]

ans → "multiplication : element wise"

20 9 42 25 -9

>> [4 -1 7 5 3] > [5 -9 6 5 -3]

→ Applies to each element

ans

0 1 1 0 1

>> [4 -1 7 5 3] <= 4

→ Compares 4 to each element

ans =

1 1 0 0 1

>> Sum([14 9 3 14 8 3] == 14) → Sum([1 0 0 1 0 0])

→ 2

Non zero: True

zero : FALSE

Combine logical operators

&& → and

|| → or

~ → not

---

x <= y && y <= z → z greatest

x >= y && y >= z → x greatest

else → y greatest

>> ~ [1 pi 0 -2]

0 0 1 0

&&, || → Not arrays — must be scalars. [convertible to scalars]

Array version — Single | and single &

>> [1 -3 0 9 0] & [pi 0 0 2 3]

1 0 0 1 0

>> [1 -3 0 9 0] | [pi 0 0 1 3]

1 1 0 1 1

```
>> a | [0 1 ; 2 3]
```

→ works with scalars & arrays

```
ans =
     1     1
     1     1
```

```
>> 1.4 < sqrt(2) & [pi > 3-1>1]
```

↓

```
1 & [1 . 0]
```

↓

```
ans =
     1     0
```

---

picker function → i/p s → condition, in1, in2
↳ o/p → out

if condition → True
out = in 1
else
out = in 2

```
function out = picker (condition, in1, in2)
if condition == 1
    out = in1;
else
    out = in2;
end
end.
```

---

eligible → v, a (i/p)
→ admit (logical o/p)

eligible → average atleast 92%
and
Individual % over 88%

] Returns logical true or false values

```
function admit = eligible (v, a)
if (((v+a)/2) >= 92 && ((v > 88) && (a > 88)))
    admit = true;
else
    admit = false;
end
end
```

function ____
if ___
else ___
    if ___
    else
    end
end

Nested - if - statements

'MATLAB' - Doesn't care about Intentation

```
if
    if
    else
    end
else
    ____
end
```

Polymorphic functions

* Behave differently based on
    * Type of i/p or o/p
    * No. of i/p or o/p arguments

Number of arguments

Two builtin functions:

nargin: returns the no. of actual i/p arguments that the functions are called with.

nargout: returns the no. of o/p arguments that the function is requested.

'called from inside a function to make our function polymorphic'

eg: function [table summa] = multable(n,m)

returns n x m multiplication table → table
Sum of all elements → summa

If m is not provided → n x n matrix.

Making a polymorphic function

```
function [table summa] = multable(n,m)
if nargin < 2
    m = n
end
if nargout == 2
    summa = sum(table(:));
end
```

table = (1:n)' * (1:m)
= (n x m)

→ Avoids execution time.
→ only when user requested.

'See Github' - for program

```
function [table summa] = multable(n,m)
if nargin<2
    m=n;
end
table = (1:n)' * (1:m);
if nargout==2
   summa = sum(table(:));
end
end
```

→ one argument must.

multable(-3,-5)

ans = []

Bug

1:-3 → [ ]

update - required.

[ ] * [ ] = [ ]

1:-3 → [ ] ∴ +1 is the default ↑.

```
function too_young = under_age(age, limit)
if nargin<2
    limit = 21;
end
if age < limit
    too_young = true;
else
    too_young = false;
end
end
```

```
>> under_age(18,18)
ans =
     0
```

Robustness

A function declaration specifies

- ★ Name
- ★ No. of i/p arguments
- ★ No. of o/p arguments

Function code & documentation specify

- ★ Use
- ★ Type of arg
- ★ What the arg represent.

Robustness

★ A function is robust if it handles erroneous i/p & o/p arguments and provides a meaningful error message.

★ Provides meaningful error messages.

↓

All matlab-built in functions are robust

Function Called Correctly - regards with o/p & i/p argument.

eg:

```
function [table summa] = multable(n,m)
if nargin < 1
        error('must have atleast one input argument');
end.
if nargin < 2
        m = n;
else if ~isscalar(m) || m < 1 || m ~= fix(m)
        error('m needs to be a positive integer');
end.
```

→ m = fix(m) → only if it is an integer.
→ when m ≠ fix(m) → show error message.

```
if ~isscalar(n) || n < 1 || n ~= fix(n)
        error('n needs to be a positive integer');
end
table = (1:n)' * (1:m);
if nargout == 2
        summa = sum(table(:));
end.
```

* check m & n
* give o/p as per wish
* Take one/two arg.

'error checking' - Though painful - worth the pain - Robust
'Avoid the pain later'

Comments

% → Extra text → human readers
→ Document
→ Explain

Before code - after function declaration.
↓
help uses this comment.

multiplication table - See Github

$(n \times 1) * (1 \times m) = (n \times m)$

```
function [table summa] = multable(n,m)

% multable multiplication table
% T = multable(N) returns an N by N matrix
% containing the multiplication tables for the integers 1 through N
% multable(n,m) returns nxm matrix
% Both inputs must be positive integers.
% [T SM] = multable returns the matrix having multable in T
% and the sum of its elements in SM.
```

⋮

See Github/matlab drive

Persistent variable

* It's a local variable, but its value persists from one call of the function to the next.
* Relatively: rarely used - No side effects eg global variable.

eg: How many times a function is called?

↙ Global variable (side effects)

↘ Persistant variable (No side effects)

```
function total = accumulate(n)
persistent summa;
if isempty(summa)
summa = n;
else
summa = summa + n
end
total = summa
```

```
>> accumulate(3)
ans = 3
>> accumulate(5)
ans = 8
```

Persistent.

first time → Initialize

Error to declare a variable persistent that already exists in a current workspace. That's why we couldn't use the o/p argument total to store the value inside the function accumulate'

clear → we couldn't accomplish what accumulate does with local variables
∴ deleted everytime

Re initialize the persistant variable:

* Resave the function (editor)
* clear workspace
* Restart matlab

---

* 'Don't use persistent variable as o/p'
* Don't put the o/p variable inside function.

'Normally o/p variable get the value after executing the function'
'Since persistent variable already has a value' – Don't use them as o/p variable.

Count error → exceeds 3 → display ('Game over')

↳ 'persistent variable is not o/p' – Just count
once exceeds
display.

---

valid_date → year, month, day
↳ valid-data → True
→ False
Valid (o/p)

* Any of the i/p is not an +ve integer
scalar → valid = false

* Leap year → 29-2-.... → valid, else not valid.

Jan – 31	① validate +ve scalar integers
Feb – 28/29	② $1 \le Day \le 31$, $1 \le Month \le 12$
mar – 31	③ 1, 3, 5, 7, 8, 10, 12 → 31 days
Apr – 30	④ 4, 6, 9, 11 → 30 days
may – 31	⑤ 2 → 28/29 days
Jun – 30	
Jul – 31	Leap year
Aug – 31	divisible by 100
Sep – 30	divi
Oct – 31	
Nov – 30	
Dec – 31	