22/8/21

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 - Symbol(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)

*

>> rand

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 &

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

Matlab as calculator

```
>> speed = (3.8x10^8) x 365 x 24 x 60 x 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 eg 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 = 2, y = 2

```
>> x = y; y = x;
>> 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 its 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])
```
(0, 12 → x ; -10, 20 → 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.

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

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)

6 rows, 4 col, 3 pages

vector: 1d array.

Matlab - Matrix laboratory

>> a = [1, 2, 3] → row vector

>> b = [1, 2, 3; 4, 5, 6] → 2 x 3 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 vectors'

(or)

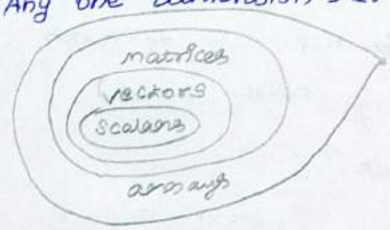
```
>> a = [1 2 3]
```

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

'Column vectors'

Scalars - vectors with both dimensions = 1

vectors - Any one dimension = 1.




```
>> x = 1:3:7
```

1 → Start, 3 → Interval, 7 → End.

```
x = 1   4   7
```

(operands)

'seperated by operators'

Colon operator

```
>> plus(2,3)
ans = 5
```

Function (Arguments) → Inside parenthesis

```
>> Colon(1,7)
1   2   3   4   5   6   7
>> 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 = []
```

Size

0 0

```
>> x = 7:3:1
```

→ 'Never achieved'

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];
>> x(2,3)
7
```

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

Updatable

```
>> x(2,3) = 97
>> x =
     1     2     3     4
     5     6    97     8
     9    10    11    12
```

```
>> x = 1
x = 1
```

matrix no longer exists.

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

```
>> y(3,3) = 5
     1     2     0
     3     4     0
     0     0     5
```

'Extented'

Subarray

```
>> x = [1 2 3 4
        5 6 7 8]
>> 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([1,2,3,4])
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]

1	10	20
2	30	40
3	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
2 2 2
2 2 2
3 3 3
3 3 3

→ Same columns.

>> A1 = [1 1]

>> A2 = [1 1 1]

No same columns

>> [A1; A2] → Error

'vertcat' - Vertical Catenation
'stack matrices'
} Same columns

else error.

Transposition:

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

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] column vector.

>> a = a'

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 2×3 matrix & 1×6 matrix

>> z = x - y

Array multiplication

z = x .* y

.* element wise

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

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

"mul each meas w.r.t their calib"

>> meas .* calib

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

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

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

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

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

= 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)

^ → Scalar & a square matrix.

```
A = [1:5 ; 6:10 ; 11:15 ; 16:20];
row = ones(1,4) → Same no. as rows   ) vectors
Col = ones(5,1) → Same no. as columns
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 (myRand.m) → Type name in command window → It will run

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   →   a =
end                          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 done assigning in command wind.

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

end

↓

User defined range

"Two scalar 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 gn, 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

out_Arg1, ..., out_Argn → single/multiple o/p ✗

in_argm → optional.

Function names

"Use meaningful names that tells you something about what your function does" : stay away from builtin function names

sort, 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.

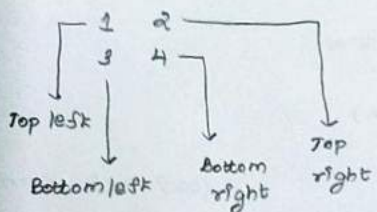
Advantage of Functions

* 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 walking 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 walking 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)

>> randi (10, 5, 4) → polymorphic
↓ Range; (5, 4) → R×C

>> 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 number.

(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
ans = 0.8147
```

→ 'Same Seed' → same numbers → 1st time

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 = [8 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 character + format specifier

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

f – conversion character (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 (5 spaces)

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 argument for completion'

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

2 3 4 5 6 → Extra arguments skipped from start

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

1 h 2 e 3h

↳ 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 precision.

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 side
<

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 errors: MATLAB catches these
* Semantic errors: 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!

>> dbstop 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 exit

'problem is here with Randi'

Eg:

```
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 : uptil 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 biggeer');
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(0.0000001) → 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]

ans → Applies to each element

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 2 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 = in2

```
function out = picker (condition, in1, in2)
if condition == 1
    out = in 1;
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
```

Nested - if - statements

```
function ____
if ___
else ___
    if ___
    else
    end
end
```

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

'MATLAB' - Doesn't care about Intentation

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 == 0
    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 - responds 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 ~is scalar (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 ~is scalar (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 mul table 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)

↘ persistent 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:

★ Remove 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 got 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
Feb - 28/29
mar - 31
Apr - 30
may - 31
Jun - 30
Jul - 31
Aug - 31
Sep - 30
Oct - 31
Nov - 30
Dec - 31

① variable +ve scalar integer
② $1 \le Day \le 31$, $1 \le Month \le 12$
③ 1, 3, 5, 7, 8, 10, 12 → 31 days
④ 4, 6, 9, 11 → 30 days
⑤ 2 → 28/29 days

Leap year

divisible by 100
divi

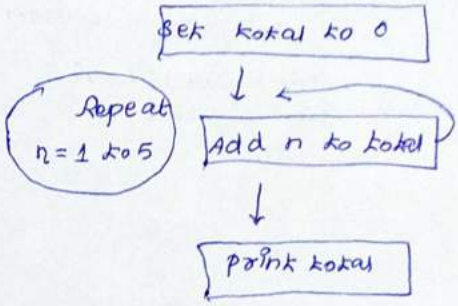
① Scalar, positive, integer
② 0 < day < 32 , 0 < month < 13 → return
③ leap year, feb, → ≤ 29 → valid
> 29 → not valid → return

④ Non leap year → a = [31 30 31 30 31 30 31 31 30 31 30 31]
a [month] ≤ day → valid → return
↳ else → not valid.

Loops

n = 1:5 ; → matlab uses internally loop

sum( ) → loop is being used by matlab



[1 2 3 4 5]

total = 0
n = 1 total = 1
2 total = 3
3 total = 6
4 total = 10
5 total = 15
loop ens

MATLAB for loop

```
total = 0;
for n = 1:5
    total = total + n;
end
fprintf ('total is %d \n', total);
```

Debugging : click the number

1
[2] → Ready to debug
3 'placed a break'
4

Points

loop
for n = 1:5 → control statement
total = total + n; → body
end

using list

```
list = rand(1,5);
for x = list
    if x > 0.5
        fprintf ('Large\n');
```

```
else
    fprintf ('Small \n');
end
end
```

* Values assigned to the loop index don't have to be integers, regularly spaced or assigned in increasing order.

* They don't have to be scalars either (columns of array)

* Any other control statements can be used inside the body of the for loop

```
lst = rand(1, N)
for x = lst
    :
    :
```

lst = [0.8147, 0.9058, 0.1270, 0.9134, 0.6324]

x = 0.8147
x = 0.9058
x = 0.1270
x = 0.9134
x = 0.6324

x takes each element in each loop

>> rng(0) → we get same numbers for rand(1,5)

```
>> u = [5 4 8 8 2]
>> v = [5 5 7 8 8]
>> u - v  → matlab uses loop.
>> for x = 1 : length(u)
       w(x) = u(x) - v(x);
   end
>> w
   w = [0  -1  -1  0  -6]
```

```
>> length(u) = 5
>> u(1) = 5
>> u(2) = 4
>> u(3) = 8
>> u(4) = 8
>> u(5) = 2
```

fibonacci

```
function f = fibo(n)
    if (~isscalar(n) || n < 1 || n ~= fix(n))
        fprintf ('n must be a +ve scalar integer \n');
    end
    f(1) = 0; f(2) = 1;
    for x = 3 : n
        f(x) = f(x-2) + f(x-1);
    end
end.
```

$P = A .* A$ → same as $A .* A$

```
[row col] = size(A)
for r = 1 : row
    for c = 1 : col
        P(r,c) = A(r,c) * A(r,c)
    end
end
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} .* \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$\begin{array}{cc|cc} A_{11} * A_{11} & & A_{21} * A_{21} \\ A_{12} * A_{12} & & A_{22} * A_{22} \end{array}$$

Row 1 → c1, c2
Row 2 → c1, c2
) 2 for loop one inside other.

$$A = \begin{bmatrix} 9 & 10 & 3 & 10 \\ 10 & 7 & 6 & 2 \\ 2 & 1 & 10 & 10 \end{bmatrix}$$

```
*
* *
* * *
```

```
for x = 1 : 3
    for y = 1 : x
        fprintf('*');
    end
    fprintf('\n');
end.
```

```
* * *
  * *
    *
```

→ 3 2 1

```
* * *
* *
*
```

```
function Inv_tree(n)
for x = 1 : n
    for y = 0 : x-1
        fprintf(' ');
    end
    for z = n : -1 : x
        fprintf('*');
    end
    fprintf('\n');
end
end.
```

```
* * *
  * *
    *
```

function halfSum → input matrix - A
→ o/p (summa)

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

1+2+3+5+6+9

1 → 1
2 → 2
3 → 3
(Col starts with)

1, 2, 3, 4
2, 3, 4
3, 4
4

```
function summa = halfSum(A)
[row col] = size(A);
summa = 0;
for x = 1 : row
    for y = x : col
        summa = summa + A(x,y);
    end
end
end
```

while Loops

"when we don't know no. of iterations"

eg: +ve integers add upto sum = 50.

"while loop"

```
while condition
    .
    .
    .
end
```

Postsum.m

```
function y = approx_sqrt(x)
y = x;
while abs(y^2 - x) > 0.001 * x
    y = (x/y + y) / 2;
end
end
```

```
root = 0.5 * (x + (N/x))
```

"Newton's method"

Busy: still working (may be ∞ loop)

"Try to avoid using i as variable"
↓
used as complex part

Next prime number?

```
function pprime = next_pprime(n)
n = n+1;
while (~isprime(n))
    n = n+1;
end
pprime = n;
end
```

★ %% → section

★ Script has no arguments — uses workspace.

Examples

%% Example 1

% skipping is accomplished in the while condition.

```
ii = 1;
while ii < length(readings) &&
                readings(ii) <= 100
    readings(ii) = 0;
    ii = ii + 1;
end
```

'replace all elements of an array - until you see a 100+' value.

Name ← 'array_0_upto100'
(Script with sections) 'array_0_upto100_for'
'array_0_upto100_index'

%% Example 2

```
for ii = 1 : length(readings)
    if readings(ii) > 100
        break
    else
        readings(ii) = 0;
    end
end
```

%% Example 3

```
for ii = 1 : length(readings)
    if readings(ii) > 100
        break;
    end
end
fprintf('first reading above 100 is
        at index %d.\n', ii);
```

'Publish' – web version will be there' – html file to publish.

$$A = \begin{bmatrix} 81 & 10 & 50 & 15 \\ 90 & 28 & 97 & 12 \\ 91 & 2 & 4 & 10 \end{bmatrix}$$

```
>> size(a, 1)
ans = 1
>> size(a, 2)
ans = 20
```

a = 1 × 20 matrix

Run script

```
for ii = 1:size(A,1)        → zero in
    for jj = 1:size(A,2)
        if A(ii,jj) <= 90
            A(ii,jj) = 0;
        else
            break;
        end
    end
end
```

```
>> A
```

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 97 & 12 \\ 91 & 2 & 4 & 10 \end{bmatrix}$$

→ when it sees 90+ skips & next step (on) turn to 0.

Logical Indexing

* Given a vector v, of scalars, create a second vector w that has only non-negative elements of v.

*

'Elegant Solution'

```
>> w = [] ; ff = 1;
>> a = [1, 2, 3]
>> w = [w, a(ff)]
   w = [1]
```

Elegant

```
w = [];
for ff = 1 : length (v)
 if v(ff) >= 0
     w = [w  v(ff)] ;
 end
end
```

```
w = [] ; ff = 0;
for ff = 1 : length (v)
    if v(ff) >= 0
        ff = ff + 1;
        w(ff) = v(ff);
    end
end
```

```
>> w = []
>> b = [1  2  3]
>> w = [w  b]
   w = [1  2  3]   → appending.
```

much elegant – Ultimate Solution

'Logical Indexing'

```
w = v(v >= 0);   [logical indexing]
```

↳ index based on logic.

```
function numfreeze = freezing(a)
numfreeze = length (a(a<32));
end.
```

```
>> [4  -1  7  5  3] > [5  -9  6  5  -3]
ans =
    0   1   1   0   1
```

```
>> c = [a>1 , a<1 , (3>2 && 4>5)]
c =
    1   0   1
```

```
>> holmes = logical ([1  -2  0  0.9  0])
holmes =
    1   1   0   1   0
```

```
r = rand (10, 1, 6);
holmes = [1 1 0 0 1 1];
r(holmes)  → banned
'Index must be +ve integers / logic values'
```

```
>> a = [1 2 3 4];
>> a = a >= 0
   1   1   1   1
```

```
>> a = [1 0 1 0]
>> b = [2 3 4 5]
>> b(a) → can't be used
∴ a(0) → Invalid.
>> a(b) → Exceeds index
"can't be used"
```

valid

```
>> a = [1 0 1 0]
>> b = [2 2 2 2]
>> a(b)
   0   0   0   0
```

```
>> a = [1 0 1 0];
>> b = [2 2 2 2];
>> a(b>2)
   ↳ [ ]
>> 1x0 empty double row vector.
```

```
>> b = [2 3 4 5];
>> a(b>2)
   ↓
  [0 1 1 1]
>> 0 1 0 (last 3 columns)
```

Note

"we can use logical arrays as i/p

```
>> a = [1 0 1 0]
a = [1 0 1 0]
```
→ Not logical
"Even though it has 1's & 0's"

```
>> a = logical(a)
a =
   1x4 logical array
    1   0   1   0
```

```
>> b = [1 2 3 4];
>> b(a)
   1   3   → 1st column & 3rd column
```

Index 1 → Taken into account
Index 0 → eliminate

V(V > V0) → we get rid of elements that are not greater than the simultaneous element in V0.

```
V  = [1 2 3 4 5]
V0 = [2 1 0 1 5]

V(V > V0)
   ↳ [0 1 1 1 0]

V([0 1 1 1 0]) → [2 3 4] → o/p
```

"logical arrays can be given as index" - size (length) - must suit.

V = [1 2 -1 -2 3 4 5]

>> V(V<0) = 0

V = [1 2 0 0 3 4 5]

>> A = [1 2 3 ; 4 5 6]

A =

1 2 3

4 5 6

>> B = A(A>2)

B =

4
5
3
6

→ why not 2×3 a column vector

[3 ; 4 5 6] → Not a matrix

"stability" - returns as column array.

"Matlab gives up keeping a 2×3 array it saves as a column array"

A = [1 2 3 ; 4 5 6]

A = A(:) → 1 2 3 4 5 6

"Exactly what happens in left side"

A(A>2) → 4 5 3 6

"Column vectors in column major order".

"1st Column - 1st"

>> stem(x,y)

t = 0 : 0.001 : 1;

y = 2 * cos(2 * pi * 1000 * t);

plot(t,y);

f = 1000

t = 0 : (0.001/1000) : (1/1000)

t = 0 : (0.001/100) : (1/100);

A = [1 2 3 ; 4 5 6]

A(A>0) → 1 4 2 5 3 6

Column major order.

Set 5×5 array to 0

>> zeros(5); A = randn(5)

A(A<0) = 0

A =

0.5377 0 . . . .

. . . . .

. . . .

5×5 → continues

```
>> a = randn(5);
>> b = a(a<0)
b =
    :
```

Column vector

```
>> a(a<0) = 0
```

5×5 matrix

When a matrix: logic indexed right side: assigned to some other- major column wise transisition happens'

Left side: a(a<0) = 0 = Assignment (Inner): No transition happens'

∴ Since size always remains as matrix which the previous case is not'

Right of '=' sign → Column transition

Left of '=' sign → No Column transition.

'Both sides' – Same matrix – No transition.

```
>> A(A>0.1) = sqrt(A(A>0.1))
```

5×5 matrix

```
>> A(A>B) = A(A>B) - B(A>B)
```

No transition — Transition happens

$$A = \begin{bmatrix} 89 & 82 & 11 & 53 \\ 33 & 5 & 59 & 42 \end{bmatrix}$$

$$B = \begin{bmatrix} 34 & 44 & 52 & 64 \\ 62 & 73 & 58 & 99 \end{bmatrix}$$

$$A>B = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$B>A = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 55 & 38 & 11 & 53 \\ 33 & 5 & 1 & 42 \end{bmatrix}$$

55 = 89 − 34
38 = 82 − 44
1 = 59 − 58

A(A>B) = 89, 82, 59

B(A>B) = 34, 44, 58

A(A>B)

↳ logical array as index (same size).

Pre allocation

'Time a function'

Function : tic, toc [Stop watch]

```
>> tic;
>> sum(1:1000);
>> toc;
```

```
>> tic; sum(1,1000); toc
```

$$\begin{bmatrix} 1 & 2 & 3 & 4 & \cdots \\ 2 & 4 & 6 & 8 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \\ \vdots & \vdots & \vdots & \vdots & \end{bmatrix}$$

```
>> function nopreallocc
N = 5000;
for ii = 1:N
   for jj = 1:N
      A(ii,jj) = ii * jj;
   end
end
```

```
>> tic; nopreallocc; toc
Elapsed time is 46.861254 seconds.
```

Small alteration

```
function preallocc
N = 5000;
A = zeros(N,N);
for i = 1:N
    for j = 1:N
        A(i,j) = i * j;
    end
end
```

```
>> tic; preallocc; toc
Elapsed time is 0.596797 seconds.
```

'78 times faster'

Why: In nopre allocation: matlab has no idea how big the matrix is!
Need to update every time in memory allocation

Case:2 - Already allocated

Just assign (update the value)

* 'Memory allocation - over & over' - Take time.
* Most of the time - unnecessary (Just Initialize).

'Allocating before using (need) - Pre allocation.

Pre allocate every time - using large array - whose dimensions you know

'matlab gives the same advice' - Note the script carefully.

A → size change on every loop
(consider pre allocating)

loops - everywhere - without it - takes only billion years to solve problems.

max_sum → V (row vector), n (+ve integer)

→ n consecutive elements eg v (whose sum is the possible largest sum)

V = [1 2 3 4 5 4 3 2 1], n = 3

∴ 4, 5, 4 = 13 (largest sum)

2
↓
1, 2, 3, 4, 5, 6, 7, 8 → length(v) - 1

9, (n=2)	9 (n=3), upto
↓	↓
8	7

'length(v) - n + 1'

9-2+1	9-3+1
8	7

n=2	n=3
x(1) + x(2)	x(1) + x(2) + x(3)
x(2) + x(3)	x(2) + x(3) + x(4)

∴ sum(1 : 1+n-1) | 1 : 1+3

1 : 2 | 1 : 3

1 : n

Key: * Initial → 1 : n = summa

* In loop → (1 : length(v) - n + 1)

then

* V(1 : 1+n-1)

```
function [summa, index] = max_sum (v,n)
if n> length (v)
        index = -1; summa = 0; return;
end
        index = 1; summa = sum (v(1:n));
        for i = 1 : (length (v) + 1 - n)
            if  sum (v(i : i+n -1)) > summa
                summa = sum (v(i: i+n-1));
                index = i;
            end
        end.
```

```
>> [summa , index] = max_sum ([1  2  3  4  5  4  3  2  1], 9)
summa = 25
index = 1
```

'Consecutive sum' – maximum sum

---

week-8 – Data types

Limitation of Computers:

Real numbers in mathematics:

* Can be infinitely large
* Have infinitely fine resolution – difference b/w two num can be only small

* 'Computers – finite memory' – limit.
* memory can't store irrational number.
* π → appr of π
* $\sqrt{2}$ → appr of $\sqrt{2}$

no. of values represented by variable in matlab → limited 'True for any language'

Data types

*Set of values/operations that can be performed to those values.

* No. of dimensions
* Size in each dimensions
* Elementary type

→ matlab allows a fn function to be called with different data types – 'strict rule': All elements of a fn array must be same type'

```
>> a=[1  2.2  3  4]
a = [1.0000  2.2000  3.0000  4.0000]
```

>> a=2 >> class(a) double >> class(sqrt(-1)) double.	**data type** double - default data type for all numerical values (matlab) >> whos 'prints all the variables in the current workspace' >>class(a) ans = double ↓ Name Size bytes Class ans 1x1 12 Char

★ All scalars → 1×1 matrix

★ String: row vectors of 'char' data type

```
>> class('sup?')
ans =
    char
>> class(4<3)
ans =
    logical.
```

double → 64 bits (8 bytes)
(Default)

$12.34 = 1234 \times 10^{-2}$

↳ 'stores 2 integers' → 1234, −2 to represent as non-integer 12.34

1234 → mantissa

−2 → exponent

★ 'matlab uses floating point representation for all doubles'
'uses 8 bytes = 64 bits'

single:

'other floating point type supported by MATLAB'

32-bit floating point

length → only 32 bits, double → 64 bits
(That's name)

Double: Extra space allows for more digits: double precession.

Integers:

* signed & unsigned.
* Sensor values: Used in computer: Typical int, Color eg Pixels.

8, 16, 32, 64 bit long (supports)

Variants

more bits - wider range.

---

double → Inf, NaN [out of range values - also double type]

Inf → $\frac{1}{0}$ (or) any value greater than the largest value that can fit in to double.

NAN → Not a number. [Invalid value $\frac{0}{0}$]

Useful functions

>> class(1)
double

>> isa(x, 'double') → True/false.
↓
Particular type.

Range check

intmax, intmin — maximum & minimum eg integer types

realmax, realmin
max & min for the floating point type.

```
>> intmax
2147483647
>> intmin
-2147483648
>> realmin
2.2251e-308
>> realmax
1.7977e+308
```

Conversion

Name of function = name eg desired data type

* int8(x) * uint32(x) * double(x) etc..

operators

* Arithmetic → Same data - no problem
  → mixed - mode → many rules & restrictions

```
>> a = uint32(2);
>> a - 1.54
ans = 0
uint32

>> a - 1.34
ans = uint32
1
```

Relational data type: → Any data type

Result: Always logical

x = single(98.76)	n = int8(-16)	m = uint16(1234)	K = uint8(500)
x = 98.7600	n = -16	m = 1234	K = 255
			↓ max value = 255

K = uint8(256)	>> K = uint8(-1)	class(K)	'Matlab converts within its fixed bound'
K = 255	K = 0	→ uint8	↓
↓ max value	↓ unsigned		Fits.

Nearby uint value – Fitted

```
>> K = uint8(2)
>> class(K)
uint8
```
→
```
K = 2
class(K)
double
```
→ The very moment we assign K, data type changed.

```
>> K = 'It makes sense'
>> K =
   It makes sense
>> class(K)
'char'
```

intmax('uint8') → 255

intmin('uint8') → 0

uint32 → 0 → −2147483648

double → $1.7977e^{+308}$

single → $1.1755e^{-38}$ (min)

```
>> a = 'hello';
>> uint32(a)
104  101  108  108  111
>> uint32('Hello')
 72  101  108  108  111
```
→ ASCII

Strings

→ fprintf : String-vectors of char-s

→ char: numerical data type → Encoding, decoding scheme

→ ASCII scheme.

* ASCII — 1960 (7 bit long — 128 characters)

* Four more schemes are developed.

Point_ascii.m — points all ascii values

```
function Char_Codes
for 9 = 33:126
    fprintf('%s', Char(9));
end
fprintf('\n');
```

Same result

```
fprintf('%s', 9);
```

↳ Converted implicitly '%s'

'matlab-gn a character & Produces a pixel pattern - that symbolizes - something to us'

```
» a = 'MATLAB for Smarties'
» a(1)        » a(1:6)        » a(end-8:end)
M             MATLAB          ans = Smarties.
```

Compare arrays

```
» a = 'Kernel'
» b = 'Colonel'
» a == b
```

matrix dimensions must agree' — Not same size.

```
» c = 'Matlab'
» b = c(end:-1:1)
'Baltam'
```

```
» c == b
[0 1 0 0 1 0] → logical.
```

'There must be an other way'

```
» a = 'hello';
» b = double(a);
```

encrypt →

```
» b = Char(b+3)
Khoor
```

↓ Decrypt

Char(b-3)

↓

Implicit Conversion then again to char

```
» line1 = '1 2 3 4 5'
» line2 = 'hello'
» txt = [line1; line2]
```

```
1 2 3 4 5
h e l l o
```

```
» txt'
1 h
2 e
3 l
4 l
5 o
```

>> poem = ['Tiger, tiger, burning bright'; 'In the forest of the night']

Error using vertcat

Dimensions of arrays being concatenated are not consistent.

↓

we are trying to create a matrix of different lengths.

'vertically - catinate'

28 chars ; 27 chars

Not equal dimension.

Built in functions - String

* Searching for a substring → findstr
* converts number to string → num2str
  vice versa → str2num
* strcmp → compares strings
* lower, upper) changes to caps/lower case.
* sprintf → fancy format → (into string)

>> findstr ('matlab', 'lab')

ans = 4.

>> findstr ('matlab', 'Lab')

ans = [ ]

'case sensitive'

>> mooc = 'MATLAB for smartkies'

>> findstr (mooc, 'r')

ans =

[10 15]

'Ignore case sensitivity'

>> strcmpi (mooc (1:6), 'matlab')

ans = 1

>> ppp = 3.1416;
>> ppp = '3.1416') Not same

ppp = 3.1416 → 'MATLAB' doesn't use quotes.

'Numbers are intended'

'Strings are not'

>> size (ppp) → number

1

>> size (ppp) → char.

1 × 6

>> ppp + 1 → 4.1416 (Implicit conversion)

>> strcmp (mooc, lang)) why?

ans = 0

>> strcmp (mooc (1:6), lang)

↳ 'MATLAB'

ans = 1

↓

If Exact → true

else → 0

» pi + 1 (Ascii then +1)

52 47 50 53 50 55

→ Implicit Conversion

3 → 51 + 1 = 52.

» str2num (pi)

3.1416

» ans + 1

52 47 50 53 50 55

» fprintf ('The area of a circle with radius %.2f is %.2f !\n', r, pi*r^2);

The area of a circle with radius 12.00 is 452.39!

» sprintf ('. . . . . ', r, pi*r^2);

→ Nothing in command window

» str = sprintf ('. . . ', r, pi*r^2);

» str

str =

The area of a circle with radius 12.00 is 452.39!

sprintf → can be taken as o/p argument (saved to a variable).

» class (str)

ans =

char

Caesars cypher → Encryption algorithm

caesar → (vector, shift amount)

→ coded

space to ~ → 32 through 126

If we shift from ~ to space (1)

-1 (space to ~)

ascii = 32:126 → 95

'ABCD' → 65, 66, 67, 68

str = str + 0; → [65 66 67 68]

str(x) - 31 + n > 95

(or)

str = str - 31;

① n → must be smaller than 95

1 - 2 → 94

95 + x + n

'Bound to 32 to 126' → 0 to 95 then add 31.

① Bound n to 1 to 95 → mod (n, 95)

② Bound Str(x)+n to 1 to 95 → 0 to 95 Str(x) = Str(x) + n

→ greater than 95

Str(x) = mod((Str(x)+n), 95)

→ less than 0

Str(x) = Str(x) + n + 95.

③ Str = Str + 31

Str = char(Str)

```
function coded = caesar(Str, n)
Str = Str - 31;
for x = 1 : length (Str)
    while abs(n) > 95
        n = mod (n, 95);
    end
    if Str(x) + n > 95
        Str(x) = mod ((Str(x) + n), 95);
    elseif ((Str(x) + n) >= 1 && (Str(x) + n) <= 95)
        Str(x) = Str(x) + n;
    else
        Str(x) = Str(x) + n + 95;
    end
end
coded = char(Str + 31);
end
```

→ caesar.m

caesar_sol.m

---

back = caesar(wrap, -96) → wrap = '2345' → '1234'

[50 51 52 53]

[49 50 51 52]

50 - 96 = [-46 -45 -44 -43]

mod(wrap - first, last - first + 1) + first

-46 - 32, 126 - 32 + 1 → 49

$[50 \quad 51 \quad 52 \quad 53]$, -96

```
first = double (' ');
last = first ('~');
txt = double (txt) + key;
txt = char (mod (txt - first, last - first + 1)
            + first);
```

* txt = (50 − 96) = −46
* txt − 32 = −78 → bound
* mod (−78, 95) → −78
* Again + 32

---

circshift – shifts the positions of elements circularly.

* ' ' : '~' → shift 96 times backwards circularly.
* we can have our element.

Take index of gn vector.

```
function caesar = CircShif(v, n)
v1 = ' ' : '~';
[~, v] = ismember(v, v1); → returns position matrix of v in v1
v1 = circshift(v1, -n);
                ↳ forward
caesar = v1(v)
end.
```

```
>> mod(8,2)
0
>> mod(2,8)
2
```

mod

∴ Bound to 95 (∴ gn is 32 to 126)

→ ∴ gn − 32

→ mod (within 95, 95)

↳ Ans

→ mod (above 95, 95)

Some answer b/w 1 to 95.

∴

→ ans + 32 → conv to char.

no need to shoot n within 95.

mod → rem

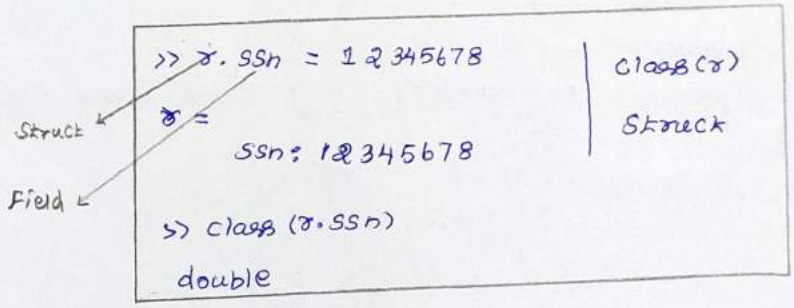
```
function Codeen = caesaen(V,n)
Codeen = V - 31 + n;
Codeen = char(mod(Codeen,95) + 31);
end.
```

"caesaen_obj_myversion.m"

Struck

* homogeneous — array
* heterogeneous — Struck → fields (not element)

Access by field names, not indices
fields in the same struck can have diff types.
field of a struck can contain another struct




```
>> r.name = 'Homeen Simpson'
>> r =
    SSn: 12345678
    name: 'Homeen Simpson'
```

```
r.address.street = '742 Evergreen Teerace'
>> r =
    SSn: 12345678
    name: 'Homeen Simpson'
    Address: [1x1 struct]
```

'struck inside struck'

arrays can hold structs
structs can hold arrays

* field of a struck can be arrays (homogeneity)

---

isfield → Is it has contain field
setfield → assigns a field (dynamic — while running)
rmfield → remove a field
struck → create a struck with contain fields.

'Bank account'

```
>> account.number = 1234;
>> account.balance = 6000;
>> account.owneen.name = 'Hasi'
```

```
>> account.owneen.email = 'NIL'
>> class(account)
ans = struct.
```

>> account . owner

ans

name = 'Hari'

email = 'N91'

Array of Struct

>> account (2) . number = 123456

account =

1×2 Struct array with fields

number

balance

owner.

Struct array: same fields.

>> account (2) . owner

ans = → Not assigned yet.

[]

>> account (2)

number = ——

balance = ——

owner = [1×1 Struct]

>> account (1) . owner . name

↓

can have different values

'But same fields'

>> length (account (1) . owner . name)

ans = 9

>> account (1) . owner . age = 23

>> account =

1×2 Struct with fields

number

balance

owner

→ only fields (enough)

one equal

↓

value can be different

∴ That's why owner (acc(1)) has age

>> account (2) . owner . age

'Not found'

acc(2) → has no age.

'field inside field need not to be homogeneous'

>> account (1:2) . owner

ans =

name: 'Joe Smith'

email: 'joe@mat'

age : 23

ans =

name: 'Jane Doe'

owner — Is the field need to be common.

'Not Values'

↓

homogeneity.

Struck array: fields must be homogeneous
values can be different

>> isfield (account(1).owner, 'age')

ans = 1

>> isfield (account(2).owner, 'age')

ans = 0

>> rmfield (account(1).owner, 'age');

↳ can't change just changes
'Need to assign'

>> account(1).owner

ans = name: 'Joe'
email: 'Joe'
age: 23

→ Not change (Just returns version alone)

>> account(1).owner = rmfield (account(1).owner, 'age')

↳ Now age will be gone.

>> Course = struct ('Area', 'CS', 'number', 103, 'title', 'Intro to matlab');

Course =
Area: 'CS'
number: 103
title: Intro to matlab

] i/p arguments come in pairs

'Setting values for fields by processing a list values': (field: value pairs)

↓

Come from fortran (LISP)

↓

List processing
(AI - still used).

Cells -

'Store a page of text'

* 'Each line - seperate string' :

* Array → Each row should be of same length
(Not a good way)

* Storing in a single string: not appealing

* Each line in seperate string: link each strings

'Special objects' - pointers

★ Idea: Vectors of special objects linking to each lines.

Pointers

★ each variable (anything) - stored in memory - Address (index)

★ Consecutive address - neighbour cells

★ MATLAB calls a pointer a "cell"

★ Pointer (variable) - stores address

Cells

★ MATLAB doesn't allow - to treat a cell as if it were a number

★ Strict rule - hard coo to make mistakes.

★ cell arrays - like structs - group heterogeneous data together (Index with numbers) - so used more frequently than structs.

Access Index

{}

```
>> page{1} = 'You could find'
>> whos
        workspace
>> class(page)
ans = cell.
```

```
>> class(page{1})
char
>> size(page{1})
ans =
     1    47
```

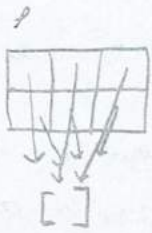
Pages

```
page{1} = 'Hi';
page{2} = 'Hello';
page{3} = 'Fine';
fprintf('\n');
for ii = 1:length(page)
    fprintf('%s\n', page{ii});
end
fprintf('\n');
```

→ display page (cell)

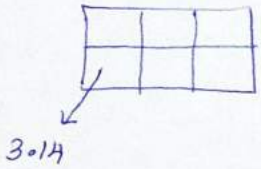
Cells are not just strings

```
>> P = Cell(2,3)

P =
    []    []    []
    []    []    []
```



"Blank" - Each element has an address of the empty array.

```
>> P{2,1} = pi

P =
    []      []    []
  [3.14]    []    []
```



* address is assigned - not - the value 3.14.
* 3.14 stored to some more - address stored in P{2,1}

"Still P is homogeneous"

↓

All elements are cell type

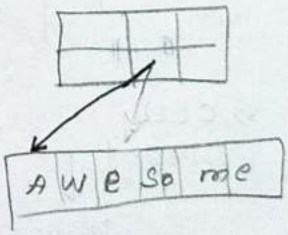
(even though it contains [] & scalars)

∴ It stores address alone : cell type

matlab cell array model allows individual elements to point at different types - but elements themselves are same data type (cell type)

"Address alone copied - not object itself"

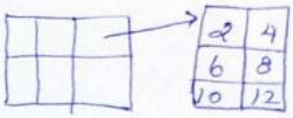
```
>> P{2,2} = 'Awesome'
```



* Pointed at first character of Awesome because the address used for vectors / may / anything - is that its first element.

» P{2,3} = [2 4 ; 6 8 ; 10 12]

"Again points first element"



Infact: makes no difference what element it points at, because MATLAB doesn't allow to look at

* numerical address
* can change it. → "strict control"

» P{1,3} = sum(P{2,3});

↓ Store pointer.

↳ Right side: object need to be retrieved instead of address.

sum(P{2,3}); → [18 24] → ⇒ Location is saved in P{1,3}

1/0 → Inf

» class(P{1,2}) → ans = double

» class(P(1,2)) → ans = cell

P{1,2} → cell object

P(1,2) → cell itself → matlab doesn't allow to see/change cell values.

» P{2,3}(3,2)

ans = 12

{2,3} → retrieves 2,3 object

$\begin{pmatrix} 2 & 4 \\ 6 & 8 \\ 10 & 12 \end{pmatrix}$

then looks (3,2) ↓ 12.

C{1} = 1.4 , C{2} = 'Hello'

» C{1} → ans = 1.4

» C{2} → ans = Hello

» C → 1×2 cell array {[12]} {'hello'}

» C(1) → [3.1456] ↓ got brackets – looking at a pointer

» C(2) → 'Hello' → with quotes

→ look at pointers.

No quotes – string

quote – Not a string (cell)

* "Matlab doesn't allow two cells to point an object" (Same)

```
>> C1 = { [1  2], [10, 20] }
>> c1 = [1x2double] [1x2 double]
>> C2 = C1
  C2 =
      [1 x 2 double]  [1x2 double]
```

```
>> c1{1,1}
ans =
     1   2
>> c2{1,1}
     1   2
>> c1{1,2}
ans =
     10   20
>> c2{1,2}
ans =
     10   20
```

If same object: Change at one position change others.

```
>> c1{1,1} = 'Straw'
C1 =
     'Straw'      [1x2 double]
>> c2{1,1}
ans =                    → No change.
     1   2
```

"It takes a copy in other cell - then assigns that copied values address in cell" — Not change the original.

"Cell pointers model: make copy - don't allow two cells point a single address."

C2 → points copy
C1 → points original
) not same address — don't change simultaneously.

↓

Not enforced by most languages C, C++, Java.

↓

Too more flexible.

"Matlab has two pointers → 1 (don't share two address)
→ 2 (like other languages)

Rarely used: Rarely needed for numeric problem solving.

cell → create cell

Sparse matrix → mostly zeros (matrix with most value zero)
↳ store where not zero alone (save space)

Sparse matrix

* cell vector – first 2 element – size
* second element – default value.

```
cell vec = {[2,3], 0, [1,2,3], [2  2  -3]};
```

```
function matrix = sparse2matrix(v)
row = v{1,1}(1,1);
col = v{1,1}(1,2);
def = v{1,2};
matrix = def * ones(row, col);
for x = 3:length(v)
    r = v{1,x}(1,1);
    c = v{1,x}(1,2);
    matrix(r,c) = v{1,x}(1,3)
end
end
```

```
b = v{1};
def = v{2};
matrix = def * ones(b);
for x = 3:length(v)
    r = v{x};
    matrix(r(1),r(2)) =
                r(3);
end
end.
```

Refer,

```
0   3   0
0  -3   0
```

```
>> a = {[2,3], 0, [1 2 3], [2 2 -3]};
>> b = a{1}
b =
    2   3
>> matrix = ones(b)
matrix =
     1   1   1
     1   1   1
>> c = a{2}
c =
     1   2   3
```

```
>> c(1)
   1
>> c(2)
   2
>> c(3)
   3
>> c = a(3)  ──→ cell (not array)
{[1 2 3]}
```

we need array not cell
use curly braces

The String type - 2017 - intro

Problems: using chars to hold strings - matlab introduced some features.

```
>> fruit = 'Strawberries'
>> class (fruit)
ans = 'char'
>> size (fruit)
ans = 1  12
```

String - Informm way of denoting vectors of characters

'Row vectors'

→ up to 2014 version no single quotes

MATLAB - uses quotes now (2017)

```
>> a = 'Straw'
a =
    'Straw'
```

other String type

```
>> a = "Hello"
a = "Hello"
```

```
>> class(a)
'string'
```

```
>> size (a)
ans =
    1    1   (not 1 x 5)
```

'See'

Any function: handles char & string as same.

```
>> fprintf('Hello');
Hello
>> fprintf("Hello");
Hello
```

→ No difference

'Works similarly'

```
>> a = 'The grain in Spain';
>> strfind (a, 'in')
ans =
    7   10   16   27   32   42
>> b = "Bushy-headed trainee"
>> strfind (b, 'in')
ans =
    17   22   38
```

Every substring starts with a lowercase or followed by more lowercase letters

regexp → regular expression

```
>> regexp (b, 'a[a-z]+r')
```

a[a-z]+

starting with a → lowercases → ending with z

Concept: function working fine with both Char & String.

'polymorphic functions'

---

function out = my_Str_function(In)

out = In(end:-1:1);

→

>> my_Str_function('Straw')

out = 'warts'

>> my_Str_function(''Straw'').

out = 'Straw' → No change

∴ >> size(''Straw'')

ans = 1 × 1

>> my_Str_function(Char(''Straw''))

→ switch back & forth

ans = 'warts'

>> String(ans)

'' '' → '' (pointS)

''warts''

'String doesn't fall into the limitations of Char type'

>> a = [''ace''; ''King'']

a =

2 × 1 String array

''ace''

''King''

] Not same size incase of Char

Size(''ace'') → 1 × 1

Size(''King'') → 1 × 1

>> a = ['ace'; 'King']

dimensions must be consistent

∴ Size('ace') → 1 × 3

Size('King') → 1 × 4

) Columns different

>> a = [1 2 3; 1 2 3 4]

'dimension error'

>> a = [''ace'', ''King'']

>> b = char(a).

2 × 4 Char array

'ace' → empty space

'King'

matlab using char adjusts to max no. of columns (putting space)

>> a = ['King', 'ace'] → not possible

>> a = ["King"; "ace"] → possible

>> a = char(a) = ['King'; 'ace_'] → possible.

>> a = ["Key"; "boaard"; "ding!"] → 1×3 String array

"Key" "boaard" "ding!"

>> char(a)

1 × 5 × 3 char array

>> a = ['Key'; 'boaard'; 'ding!']

ans =

'Keyboaarding!' → Concatenating

>> b = [1, 2, 3]

>> c = [12 0 1 -2]

>> d = [7 7 7 7]

>> e = [b, c, d]

= [1 2 3 12 0 1 -2 7 7 7 7]

Concatenating

Sometimes — we need Concatenation.

Concatenation

>> str1 = "Key"

>> str2 = "boaard"

>> str3 = "ding!"

>> Str1 + Str2 + Str3

"Keyboaarding!"

>> "three" - "one"

'undefined operation'

'+' only arithmetic operation to be used in strings

>> string(17)

"17"

>> string(-1.123456)

"-1.1235"

>> string(3.0e8)

"300000000"

>> b = uint16(477)

uint16

477

>> string(b)

"477"

>> string(pi)

"3.1416"

4 digits - short precision

string(true) → "true"

uses number format (bank exp)

string([34 0 -4 18] > [0 0 0 0])

ans =

1 × 4 string array

"true" "false" "false" "true"

>> logical ("true") → error (not possible)

∴ matlab → don't know meaning of string.

>> double ("17"). → double knows.

ans = 17

Non numeric string.

>> double ("Hello") ← ans = NaN

>> double ('beelo')

104 101 101 108 111

>> double ("pi")

NAN

'matlab doesn't know meaning'

string (PI)

ans = "3.1416" → short format

>> format long

>> double ("-1.123456")

ans = -1.12345600000000

>> format short

>> double ('17')

ans : → ASCII

49 55

>> str2num ('17')

ans =

17

>> str2double ('17')

ans =

17

>> str2double ("17")

17

>> double (string (unknown)) → convert to string then double

>> int8 ("17") → Conversion to int8 from String is not possible.

↓

>> int8 (double ("17"))

ans =

int8

17

>> string ({1776, logical (1776), 'independence'})

1x3 String array

"1776" "true" "independ"

cell to string ←

Access Individual Characters

>> a = 'MATLAB for Smarties';

>> a (1:6)

ans =

'MATLAB'

↓

Not happen with strings.

>> temp = char ("independance")

>> string (temp (3:8))

"depend"

↓

>> extract Between ("independance", "in", "ence")

>> depend

↓

Now b/w

>> For string

s = "independence"

>> extract Between (s, 3, 8)

ans =

"depend"

↳ Not b/w, since 'in' includes also

58 functions - working with strings

>> doc → matlab documentation page
'language fundamentals' ← → Data types → characters & strings.

numerical functions → help page

date time

>> datetime

ans =

datetime

28 - Jul - 2018 10:22:43 → Based on Computer time & date.

↳ pop up window (documentation)

>> class (ans)

ans =

'datetime'

Preferences

Command window → default → time format

(we can get resolution - ms, μs, ns)

>> datetime ("yesterday"), datetime ("today"), datetime ("tomorrow")

27- Jul - 2018 28 - Jul - 2018 29 - Jul - 2018

'computer clock' ↓ Not specific to include time

>> datetime ("now")

open webpage

'function can do automatic documentation with date & time'

[~, weekday_name] = weekday (search_time, 'long');

↓ doesn't use this argument

↓ Full name (no need for abbreviation)

>> open_webpage

Enter the url : cs103.net

At 11-Aug-2018 09:26:50, you opened the webpage at cs103.net

Have a great Saturday!

open webpage

returns 0 if successful ← web('url')

webpage_log.m

>> hours (2.5)

ans =

duration

2.5 hr

>> days (2.5)

ans =

duration

2.5 days

>> datetime

11-Aug-2018 09:27:48

>> three_years_from_now = ans + years (3)

11-Aug-2021 02:55:24

>> three_days_ago = rightnow − days(3)

(datetime)

>> datetime (1989, 11, 9)

09-Nov-1989

>> datetime (1918, 11, 11, 11, 0, 0)

11-Nov-1918 11:00:00

>> timezones → opens webpage (we can search)

'we can do addition & subtraction'

timezones

>> timezones

>>

'Set the time zone property of the datetime variable to the string Europe/London'

↓

dot operators

>> agreement = datetime (1918, 11, 11, 11, 0, 0);

>> agreement.TimeZone = "Europe/London"

>> agreement =

datetime

11-Nov-1918 11:00:00 → No change (∵ London)

make a copy

>> agreement_India = agreement → copies everything

>> agreement_India.TimeZone

ans =

'Europe/London' → Just copied everything.

```
>> agreement = datetime(1947,8,15,1,0,0)

agreement =
            datetime
    15-Aug-1947  01:00:00

>> agreement.TimeZone = 'Asia/calcutta'

15-Aug-1947  01:00:00
```

'changing timezone' ⟶

```
>> ind_ame = agreement

>> ind_ame.TimeZone

ans =
    'Asia/calcutta'

>> ind_ame.TimeZone = 'America/
                       New_York'

ind_ame =
    14-Aug-1947  15:30:00
```

datetime(1947, 8, 15, 11)

↳ 'If you are giving hour - must give sec, min'

else: error message

3 or 6 or 7 columns array i/p

ind_ame.TimeZone ⟶ used as structs.

See field names

```
>> fieldnames(ind_ame)

ans =
    8×1 cell array
        {'Format'}
        {'TimeZone'}
        {'year'}
        {'month'}
        {'Day'}
        {'Hour'}
        {'minute'}
        {'second'}
```

we can't add new fields (fixed) ↓ use struct to properties ↓ So its datetime not Struct. ←

```
>> class(ind_ame)

ans =
    'datetime'
```

(London)

```
>> Treaty_Of_versailles = Armistice_WW1 + day(228)

28-Jun-1919  12:00:00 ⟶ change(11:00:00)
```

(Day light saving time) – In European Countries (From 1916)

US – started in 1919/2007(stopped)

need only date alone

>> Treaty_of_versailles • Format = 'dd-MMMMM-YYYY'

27-June-1919

June

(or)
dd/MM/yyyy
↓
month
mm → minutes

(or)
See properties.

'eeee' – day of the week.

'eeee, MMMM, dd, yyyy' → Friday, June 27, 1919.

'Takes cares of leap years/daylight hours' – etc...

>> Treaty_of_versailles + years(100)

ans =

wednesday, June 26, 2019
↳ not 27

→ Reason: leap years (365.2425 – MATLAB – All years has same days)

→ length of year averaged over four Centuries.

Some variations with calendars

our Calender – 365 – leap year – 366

>> Treaty_of_versailles + calyears(100)

By calendar.

Thursday, June 27, 2019

---

year, hour (Singular version) → very old functions

>> day(Treaty_of_versailles)

ans =

27

>> month(Treaty_of_versailles)

ans = 6

>> hour(")

ans =

12

» yuanteen (Treaty_of_versailles)

2

→ which part of year (which yuanteen)

(Financial functions)

"Financial toolbox" – kolonate – handy (billions of array elements)

They won't fits in comp memories.

» fnd_ame.year

1918

» fnd_ame.day

fieldnames (fnd_ame)

» duration (3,7,43)

03:07:43

→ duration

» half_duration = duration (3,7,43)/2

1:33:51

» 2 * half_duration

3:07:43

» long_duration – half_duration

01:33:51

» long_movie + 1

↳ added as day.

ans =

27:07:43

→ mixed mode arithmetic

hours + double = duration.

(went up by 24 hours)

"Basic time unit of duration = day" – like microsoft Excel Spreadsheet

---

» double (long_movie) → not possible (use seconds, minutes, hours, etc..)

↓

matlab doesn't know format

(seconds or minutes/etc)

» seconds(long_movie)

ans = 11263

» days(long_movie)

ans =

0.1304.

Subtract two datetime

↓

duration will be the answer.

```
>> a = datetime(1756,5,17) - datetime(1763,2,15)
59160:00:00
>> years(a)
6.7489
```

```
>> b = days(datetime(1764,5,17) - datetime(1764,2,15))
b = 92
```

"Takes care of leap years"

days → double (not duration)

```
>> days(2,5)
ans =
   duration
   2.5 days
```

→ polymorphism

why — Sometimes we need double, duration.
- duration - when i/ps are numbers
- number - when i/ps are duration

* double() → doesn't know unit
* hours, minutes, seconds, days, milliseconds, years → (conversions) 6 functions

double ↔ durations → No need for 12 functions.

Points in time

```
>> Start = datetime(2018,9,10)
10-sep-2018
>> End = datetime(2018,11,19)
End = 19-Nov-2018
>> rng(0)
>> dates = Start - days(30) + days(randi(100,5,1))
```

↓ 5 days

dates = 5x1 datetime array

```
01-Nov-2018  00:00:00
10-Nov-2018  "
24-Aug-2018  "
11-Nov-2018  "
14-Oct-2018  "
```

```
>> Start <= dates & dates <= End
```

5x1 logical array

```
1
1
0
1
1.
```

```
>> Start = datetime(2002,5,18)
>> End = datetime(2002,7,18)
>> dates = Start + days(30) + days(randi(100,5,1))
>> Start <= dates & dates <= End
```

```
1
1
1
0
1
```

>> make_calendar (8, 2018)

ans :

11 x 1 String array

	August 2018					
Su	mo	Tu	we	Th	fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

datetime (2018, 3, 27) : days(14) : datetime(2018, 5, 13)

ans :

1x4 datetime array

27-Mar-2018 10-Apr-2018 24-Apr-2018 08-May-2018

Colon operator

>> datetime (2018, 3, 27) : 14 : datetime (2018, 5, 13)

27-Mar-2018 10-Apr-2018 24-Apr-2018 8-May-2018

54:08

>> days (1) : 3 : days (19)

1 day 4 days 7 days 10 days 13 days 16 days 19 days

>> calendar (2021, 9)

Sep 2021

S	M	T	W	Th	F	S
0	0	0	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	0	0
0	0	0	0	0	0	0

→

a =

0	0	0	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	0	0
0	0	0	0	0	0	0

6x7 double array

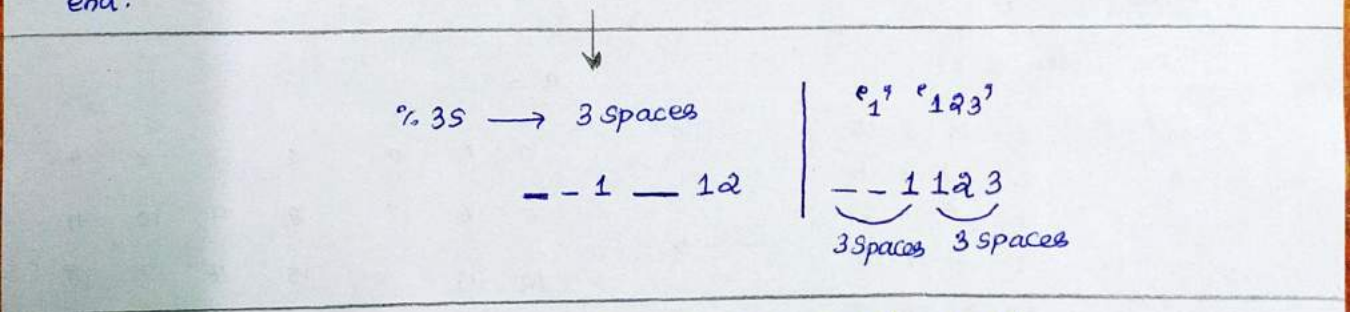
Problems

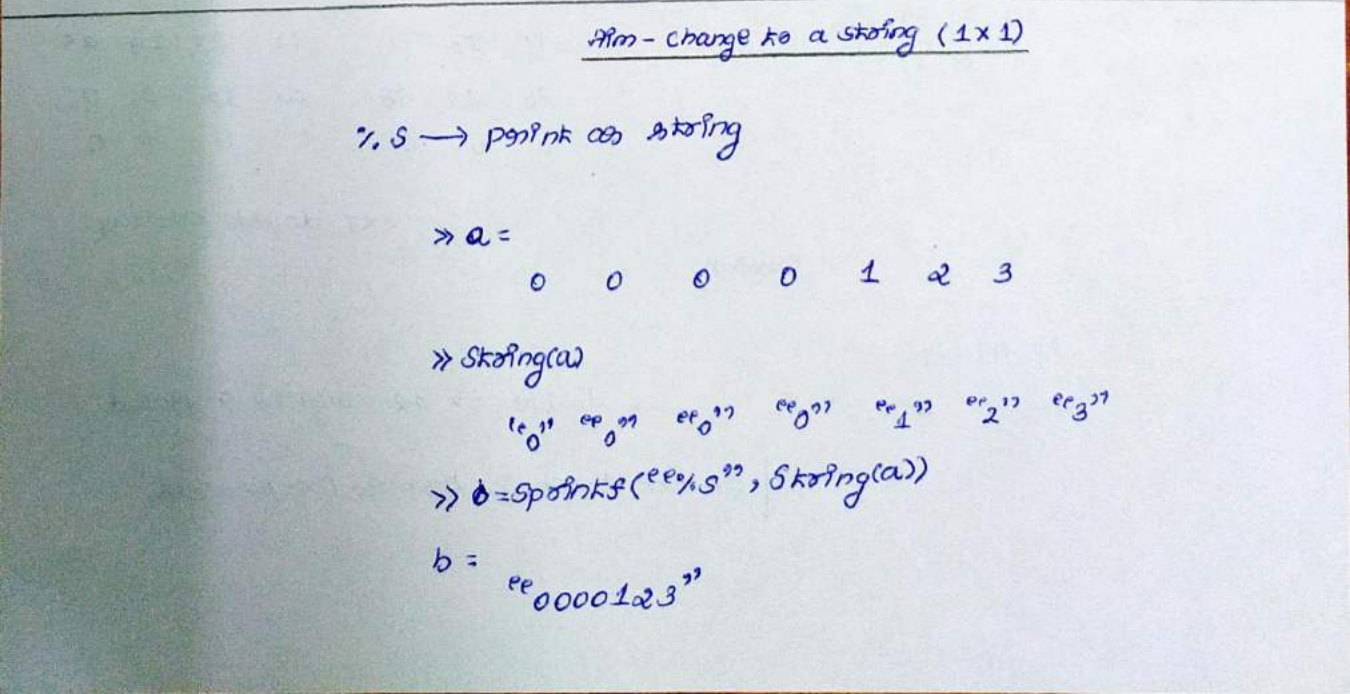
If a(x, y) == 0

a(x, y) = ' ' → double → 32 will be replaced

↓ Convert to string then do the process.

```
function cal_str = make_calendar (n_month, n_year)

dt = datetime (n_year, n_month, 1);   01-May-2002
                                      ----→ 28
dt.format = 'mmmm yyyy'  → May 2002

cal_num = calendar(dt)

title   = string(dt)   → "May 2002"

cal_str = strings(11,1)  → preallocation

left  = blanks (floor(14 - strlength (title_str)/2));

right = blanks (ceil(14 - strlength (title_str)/2));

cal_str([1,3,11]) = "----------";

cal_str(2) = sprintf("%s", left, title_str, right);

cal_str(4) = "Su Mo Tu We Th Fr Sa";

for pp = 1:6

    temp = sprintf("%3s", string(cal_num(pp,:)));

    cal_str(pp+4) = strrep(temp, "0", "");  → replace 0 with ""

end.
```





my-cal.m

① User → month, year (9, 2021) ⟩ conversion.
calendar (2021, 9) ⟨

dat.format = 'MMMM yyyy' ⟩ calendar created
cal_number = Calendar(dat);

title = " September 2021"

» cell(

"September 2021" → 14

26 − len(title) = 12
↙ ↘
6 6

Suppose 13
↙ ↘
6 7

floor(12.6) → 12
ceil(12.6) → 13

» BL = sprintf("%.5", ans, "hello", ans)

n =

" hello "

%.3S → 3spaces must

'Build the function'

" 0 1 2 10 20"

strrep(temp, "0", " ") → " 0 1 2 1_ 2_"
↓
All zeros are replaced

Solution

10 → nogap

"make 0 in single zeros"

↓

temp = sprintf("%3f", (1,:))

temp =

" _0 _0 _0 10 20 30"

∴ strrep(temp, "_0", " ") ✓

Points : "p1" "p2" "p3" → can't combine into single one.

way

%S , temp(1, :)

---

we can use date

5-August-2018 12:49:0 → AS x axis

plot(hide_datetimes, hide_heights)

or

```
>> a = datetime("now");
>> b = a + hours(13);
>> c = a : hours(1) : b; → 1 X 14 matrix
>> d = randi(4, [redacted], 14);
>> plot(c, d)
```

→ plotted.

c → datetime, d → matrix

Lesson:8 : I/O File

Read & write data: managed by OS

* File: exchange data b/w programs & o/p.
* File input/output.
* write a doc - generate pdf (example)
* for permanent storage.

handle : text, binary, excel spreadsheets

---

change current folder

'pwd'

```
>> pwd
```

→ unix OS (directories)

print working directory

```
>> C:\Users\fikzpajm   → windows
   /Users/fikzpajm     → mac
```

* strrep(temp, old, new) → replace string

>> list of files in sub/current folder.

>> ls (list)

data.m
hello.m
→ array of chars. [1×198]

>> whos and → display details.

Change path

>> cd ("path") → change directory.

>> cd ('Lesson08')

↓

> Tutorials > Lesson 08

Lesson 8 in the Tutorials (parent)

Other directory - "other files in some other parent can't be opened"

>>

>>

Back to parent

>> cd ('..') → back to parent.

Tutorials → Lesson 01 to Lesson 08

>> we are in tutorial now

>> cd('Lesson 07') → changed (since we are in its parent)

---

* Functions take strings as an argument
* built in functions - don't need parenthesis - single word - no need quotes

>> ls 'Lesson 07'

>> help sort
>> help ('sort') } Same.

>> cd ('Lesson 08')

» cd ('../..') → to grandparent. .. → parent

» mkdir ('new_folder') → make directory

» rmdir ('new_folder') → Removes only empty directories

↳ if it has something: error

Recycle: send it to trash / permanently delete.

* » whos
  : All variables in workspace

* save → save workspace. (Current) ⟩ preserve variables
* load → load workspace (Current)

Caution: existing variables will be overwritten

we can save particular / load particular variables.

save my_data_file data s a → particular variables

filename | variables

load my_data_file — data
— s
— a

» load my_data_file s a
↓
only s & a

Excel files

* xlsread
* xlswrite

» [num, txt, raw] = xlsread ('Nashville_climate.xlsx');

num → numbers from text
txt → text
raw → everything

num =

46	28	3.98
51	31	3.7
61	39	4.88
:	:	:

→ matlab identifies smallest rectangle to be extracted.

'num - smaller size than spreadsheet'

'Some cells may not have numerical data : NaN'

NaN NaN NaN → See slides

>> txt = all the texts

txt → cell array

txt =

[1×30 char]
[1×40 char]
'Jan' 'High temp'

→ like in cells ondeon

Too long strings to point in window

text → no. eg char

∴ rect array - not possible

(matlab prints type) alone

* Type : txt → cell array.

>> raw

raw =

union to num & txt

>> temps = xlsread ('Nashville_climate_data.xlsx')

temps =

[temps txt]

(op)

'Loaded'

[~ txt]

→ alone

>> num = xlsread ('Nashville_climate_data.xlsx', 1, 'D15')

61

(op)

No. eg sheet (op) Name

Name eg cell

>> num = xlsread ('Nashville_climate_data.xlsx', 1, 'D15:E17')

num =

61.000 ...

... ...

... ...

up to (rectangular)

* xlswrite → write in xls files
* But on MAC, matlab provides a limited alternative
  * xlswrite → writes text files (Instead of excel)
  * only numerical arrays can be written
  * Each line is wrote on a seperate line by comma seperated values (numbers on a row seperated by commas.)

↓

* Comma seperated values (CSV)

caution: we can overwrite: without even getting any warning.
"only modify the cell - that actually you write"
other cells won't be affected

	A	B	C	D	E	F
1		Abilene, TX	Akron, OH	Abbu	Alexa	Allentown, PA
2	Abi	0	1481	724	1615	1730
3	Akron, OH	1481	0	2185	382	552
4	Albuquerque, NM	724	2185	0	2331	2447
5	Alexandria, VA	1615	382	2331	0	195
6	Allentown, PA	1730	552	2447	195	0

Problem

get_distance → Two City names
→ distance (B/w two cities)

one (or) both cities - not in file → returns -1

Analyzing problem

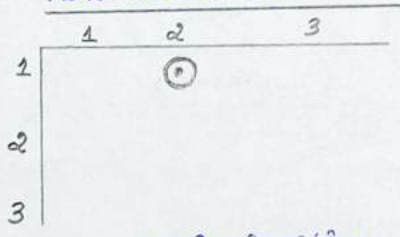
num → doesn't have cities
txt → doesn't have values
raw → fit for own purpose.

Sample

```
>> [~, ~, raw] = xlsread('Distances 1.xlsx', 1, 'A1:E5');
                                  ↓          ↓        ↓
                                File.      1sheet   Cells
```

```
>> raw =

{ [  NAN] }  . . . .
{'Abilene, TX' }  . . . .   ⟶ cell type.
{'Akron, OH' }  . . . .
{  .  }  . . . .
```

Take row & col number from city



1, 2
↓
Concentrate on col number
∴ Row & col are same

main function

```
function distance = get_distance(city1, city2)

[~, ~, data] = xlsread("Distances.xlsx");

city = string(data(1, :));  ⟶ For comparing city

city1 = city_valid(city1, city);
                  ↳ city 1  ⟶ city names list

city2 = city_valid(city2, city);  ⟶ city name list
                  ↳ city 2

if (city1 > 1  &&  city2 > 1)
    distance = data{city2, city1};
                  ↳ column  ⟩ replacable
                  ↳ row

else
    distance = -1;  ⟶ city not in list

end
end
```

city1, city2
Index eg city 1 & 2

helping function

```
function index = city_valid (cit, City)
for x = 1 : length (City)
    if strcmp (City (1, x), cit)
        index = x ; return ;
    else
        index = 0 ;
    end
end
end
```

City → list of cities (string)

→ compare with each elements of column array of cities match → that index is the index of City1 (cit)

text files

* Strings of characters: encoded : Matlab takes care of those.
* open/close - permission from OS.

* fid = fopen (filename, permission)
  fclose (fid) → opened file
* permissions : read, write, overwrite.

- 'r' - open text file - read
- 'w' - write (open - discard existing contents)
- 'a' - append data to end (open/create)
- 'r+' - open - don't create - read & write
- 'w+' - open/create - read/write - discard existing content
- 'a+' - open/create - read/write - append data to end.

'r+' → both read & write (don't create)

't' → text file

fopen() → if we don't have write permission/error → returns negative number

Climate Data for Nashville, TN

(Average highs (F), lows (F), and Precip (in)

	High	Low	Precip
Jan :	46.00	28.00	3.98
Feb :	51:00	31.00	3.70
March :	61:00	39.00	4.88
April :	70:00	47.00	3.94
May :	78:00	57.00	5.08
June :	85:00	65.00	4.09
	High	Low	Precip
July :	89:00	69.00	3.78
Aug :	88.00	68.00	3.27
Sep :	82.00	61.00	3.58
Oct :	71.00	49.00	2.87
Nov :	59.00	40.00	4.45
Dec :	49.00	31.00	4.53

```
>> fprintf('%1.3 %.3 %.3', String([1 2 4])
```

1 2 4

%.2 → 2 zeros (or)

%3.00 → 2 zeros.

write_text.m

```
file = fopen(file_name, 'w+t')
```

```
If file < 0
    return;
```
→ error while opening

%.5s → formatting

Text files - Broken into lines - Reading text files

→ one line at a time

→ type points a text file in the command window

fgets → read a line & returns a string. (one time)

for multiple lines → loop it

when nothing to return → −1

So while !ischar(oneline)
    use fgets()

Point_line-by-line.m

Reading - easy (txt files) → converting in to numerical data is much harder

recognize digits, decimal points etc.. → Tedious job.

"Cumbersome"

↓

"Poor way to store data"

Binary files

* "Any file: not a text file – Binary file"
* Represent numbers (have all data)
* we need to type..

> fopen, fclose
>
> 'r', 'w', 'a', 'r+', 'w+', 'a+'

```
function write_array_bin(A, filename)
fid = fopen(filename, 'w+');
if fid < 0
    error('error opening file %s\n', filename);
end.
fwrite(fid, A, 'double') → 'write an array(double) A'
fclose(fid)
```

Reading the array – read a double array from file

* A = fread(fid, inf, data_type)

  ↓

  no.of items (All - inf) – read everything.

* fclose(fid); → else it can't be opened by other functions.

"write_binaryfile.m"

>> write_binary (a, 'datafile.dat') , a = randn (10,12)

open the written file

'write_binary.mat' – using textread
(read_line_by_line.m)

Problem:

>> write_binaryfiles (a, "write_binary.dat")

>> read_line_by_line ("write_binary.dat") → Garbage lines
(Not a text file – So Garbage encoded

>> read_binaryfile ('double', 'write_binary.dat')

ans =

120x1 double array.

∴ Binary files – not encoded – will be encoded by text encoding scheme.
(String of doubles)

>> read_binaryfile ( ) → recovers but not in 10x12 (120x1)
↳ original.

fwrite → writes the each element in a long – one long stream of numbers with nothing written about the dimensions of the array.

↓

120x1 double vector

* read_binaryfile() → simply read as it is

Preserve dimensions: using binary files

↙ Remember that file with a certain name has an array such & such dimensions

↘ write the dimension info into the file as well.
(Better option)
(one wishes to know format of the file)

write_dims_array_bin.m

```
function write_dims_array_bin (A, filename)
fid = fopen (filename, 'w+');
if fid < 0
    error ('error opening file %s \n', filename);
end
dims = size(A);
fwrite (fid, length (dims), 'double' ); → In case dimension is always 1×2 (no problem) no need
fwrite (fid, A, 'double'); ⟶ dimension
fclose (fid);
```

(A×2)

(2) 4 2

take 2 numbers

Problems

```
1  2  3  4
5  6  7  8   ⟶ fwrite ⟶
9  10 11 12
```

A

.dat file

{1 5 9 2 6 10 3 7 11 4 8 12}

one long line.

B

∴

A(1,1) = B(1)	A(1,2) = B(4)	A(1,3) = B(7)
A(2,1) = B(2)	A(2,2) = B(5)	A(2,3) = B(8)
A(3,1) = B(3)	A(3,2) = B(6)	A(3,3) = B(9)

'written Columnwise'

mod_read_binaryfile.m

mod_write_binaryfiles.m

* save dim→2×3 → as 1 & 2nd element of .mat file
* using loop → mod_write.m → create the reqd matrix.

» isequal (1, 1) → 1 (only both are true)

Efficient method

reshape( ) ⟶ reshapes the matrix (vector)

Take dim

$$a = \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{matrix}$$

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

reshape(a, [2,4]) ⟶

1	3	5	7
2	4	6	8

Single data type – matlab

* y = single(10); ⟶ 4-byte (32-bit) – floating point values

fread returns ⟶ column vectors

Possible to return any type: – Altern 3rd argument

o1 = char(fread(fid, nums(1), 'char ⇒ char'))';

↓ i/p ↓ o/p

Same (in this case)

'single ⇒ int64'

To know:

'int16', 'single'

fread(file, int8, 'int16')

Keeping file sizes small: important when handling huge data

↓

Allot according

(read – single or int8 instead of double)

* Find real problem – Solve them – give real solutions

matlab inbuilt programs

>> edit numgstr [popup] → mathworks programmers.

(or)

matlab central

Saddle points

Saddle → M (saddle point: whose element is greater or equal to every element in its row, & less than or equal to every element in its column

↳ Indices (matrix of saddle points) indices

Indices: 2 columns → row, column → index

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$, compare 1 with all elements of row >= , 1 with all elements of col <= → saddle

row = 51, col = 41

my idea

```
1 for loop  x = 1:row  → row
    for loop  y = 1:col  → col
        r = 0, c = 0;
        for s = 1:col
            if M(x,y) >= M(x,s)  → saddle point
                r = r+1;
            end
        end
        for t = 1:row
            if M(x,y) <= M(t,y)  → saddle point
                c = c+1;
            end
        end
```

r = col, c = row → both True → saddle point.

'saddle_point_me.m'

```
>> min([2 1 3])
1
>> max([1 3 2])
3
```

>> Saddle_min_max_method.m

Logical indexing

```
>> min(2,3)
2
```

```
>> a = randi(10,4,4)
min(a)
```

→ default.

2 1 2 2 → In each column

>> a = min(a, [], 1) → 2 1 2 2 (default case 1, [])

↓ No input (2nd one)

↓ Columnwise

>> a = min(a, [], 2)

↳ Row wise.

```
7
1
2
2
```

★ Compare row wise greatest with each element in row

$$a = \begin{bmatrix} 9 & 7 & 10 & 10 \\ 10 & 1 & 10 & 5 \\ 2 & 3 & 2 & 9 \\ 10 & 6 & 10 & 2 \end{bmatrix} \quad b = \begin{bmatrix} 10 \\ 10 \\ 9 \\ 10 \end{bmatrix} \quad c = \begin{bmatrix} 2 & 1 & 2 & 2 \end{bmatrix}$$

↓ Row wise greatest

↓ Columnwise smallest.

>> a ≥ b

→ 10 ≥ 10

$$\begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

>> a ≤ c

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2 ≤ 2 ←

* Number both greatest of row, smallest of column has 1 in it

* S & K = $\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$ → no saddle points.

---

* isempty([]) → 1

* isrow([1 2]) → 1 (row vector)

[4 3 5 12 ; -1 0 -2 0 -1 ; -4 1 4 3 5;
-3 0 -1 0 -2 ; 3 2 -7 3 8]

Some cases answer will be like

[2 2 3 1 4 4] Instead of

2	1
2	4
3	4

Solution

[Row' col'] = [Row' | col']

Row'	col'
2	1
2	4
3	4

when even isrow([row col]) → safety measure (if needed)

ans = [row' col']

Image blur

function output = blur(img, w) img → 2d array (0,255 values)

Blurring → Average the pixel values in the vicinity of every pixel.

outer pixel value = mean of the pixels in a square submatrix (2w+1)

★ when $w = 1 \rightarrow$ use 3×3 matrix

$w = 1$ at index (1, 1)

mean of (1, 1), (1, 2), (2, 1), (2, 2)

I/p & output are units

1	2	3
4	5	6
7	8	9

>> nan (1,4)

NaN NaN NaN NaN

$2w + 1 \rightarrow$ 3×3 sub-matrix

11	12
21	22

If $w = 1 \rightarrow$

11	12
13	14

when row = 1, col = 1

r1 =

r2 =

r3 =

r4 = col + 1

11	12	13
21	22	23

1 : 2 , 2 : 3

problem: 11, 21, (12, 22) → only 1×2 → so take front & back

when $w = 1$

11	12	13
21	22	23

max: 3×3

```
function output = blur(img, w)
img = double(img);
[row col] = size(img);
output = zeros(row, col); ——→ Initialize matrix for speed
for i = 1: row
    for j = 1:col
        r1 = i - w; ——→ sub matrix may be upto 3×3 matrix for w = 1
        r2 = i + w;                                  (w = 2(1) + 1 = 3)
        c1 = j - w;
        c2 = j + w;
        if r2 > row
            r2 = row;
        end
        if c2 > col
            c2 = col;
        end
        if c1 < 1
            c1 = 1;
        end
        if r1 < 1
            r1 = 1;
        end
        mat = img(r1:r2, c1:c2);
        output(i, j) = mean(mat(:));
    end
end
output = uint8(output);
end
```

Echogen

Echogen - add an echo effect to the audio recording. An echo is the original signal delayed & attenuated. First compute echo then add to the original signal with the correct offset.

output = echo_gen (input, fs, delay, amp);

input → column vector [values b/w −1 and 1] - sequence representing a time series of digitized sound data

fs - sampling rate - CD player uses 44,100 samples/second.

delay - delay of the echo in seconds

* Echo should start after the delay seconds passed from the start of the start of the audio signal.
* amp - amplification of echo [As values < 1 → not loud, amp > 1]

* o/p - column vector having orig sound with echo superimposed
* o/p vector - no longer the same size (As delay ≠ 0)

* (round to the nearest value) : no of points - to get the delay → ceil / floor
* Sound → values b/w −1 & 1. So if the echo causes values to be outside of this range - scale down the entire vector, so that all values are within the range while retaining their relative amplitudes.

audiofiles : splat, gong & handel

* load gong %. loads two variables, y and Fs
* sound (y, Fs) %. outputs sound.

audio → 2 columns - stereo file
1 column - mono audio file.

10
5
20 → scale all to 1
÷ max
30
(30)

$$\begin{pmatrix} 1/3 \\ 1/6 \\ 2/3 \\ 1 \end{pmatrix} \rightarrow \text{scaled.}$$

function output = echo_gen (s, FS, delay, amp)

dt = 1/FS ;

N = round (delay/dt) → calculate no. of points.

∴ 1 second FS samples are required, for delay seconds we need to add delay * FS sample points.

FS * delay (or) $\frac{delay}{dt}$ = delay * FS

(or)

$\frac{delay}{time\ per\ sample}$ = Total sample req.

output = superimpose (echo signal + original signal)

echo signal → delay (0's) up to delay time then original signal.

Add = echo_sig + orig signal

↳ due to add of delay in echo_sig (data size - add zeros at end - delay)

Scale:

div by maximum

'echo_gen.m'