Why the anonymous!

a = @sin ;
↳ Function

* a is not the function name
* a is the name of the variable that handles the function sin

(sin itself has no name: Anonymous)
"handle for sine"

b = @(x) x+2 ;
↳ No name

"Function itself has no name"

---

splot (@(x) x+sin(x) , [-5,5])
↳ Interval

Anonymous function
(no name)

---

multiple o/p arguments — supported (limited way!)

* only way to get multiple o/p - have the expression (body of the anonymous function) - be a call of a function that already returns.

```
>> Smax = @(A) max(A, [], 2);
>> [mx ind] = Smax([1 2; 3 4])
```

```
mx =
     9    16
ind =
     2     2
```

→ Row index.

```
>> Smax(1:10)
ans =
    100
```

* with anonymous functions, we can get multiple o/p s only by calling a function that already produces multiple o/p s.

Versatility

eg: I want Sum & Product as o/p.

```
>> xyz = @(x,y) deal(x*y, x+y)
>> [P, S] = xyz(10, 20)
P = 200
S = 30
```

↳ deal( )
↓
gives multiple o/p s.

* deal(-,...) → returns o/p argument equals to its i/p argument.

```
>> a = @(b) max(b, ^3)
>> a([1 2 3; 4 5 6]);
64   125   216
```

```
>> [a,f] = a([1 2 3; 4 5 6])
a =
    64   125   216
r =
     2     2     2
```

```
>> xyz = @(x,y) deal(x*y, x+y)
>> [p,s] = xyz(10,20)
p = 200
s = 30
```

```
>> xyz = @(x,y) deal(x+y, x-y, x*y, x/y);
>> [a b cd] = xyz(1,2)
a = 3
b = -1
c = 2
d = 0.5000.
```

>> xyz(1,2) → error (not enough o/p arg)

Note: no. of i/p arg of arg must = to no. of o/p arg.

* Function handle for normal function a = @sin → Any no. of statements in its body
* " Anonymous fun a = @(x,y) x+y; → only one statement in its body

Saves defining normal functions.

* Functions → main, sub, nested, Anonymous (So far)

Nested Functions

* lives in an M-file - with a main function (like a subfunction but nested function is actually defined inside the body of another function) → parent function (main function)

eg:

```
function [y1, y2] = first_nested_example(x)
    c = 10;
    sub(c, x);
    y1 = inner(x);
    function out = inner(in)
        out = c * in;
    end
    c = 11;
    sub(c, x)
    y2 = inner(x);
end
function sub(in1, in2)
    fprintf('Multiplying %d times %d \n', in1, in2)
end
```

→ Nested function (Anywhere in the body eg main function).
No effect on execution eg main function.
'Ending functions within statements is optional in an m-file with no nested functions'

→ Sub function

* If nested function is there - every function must be terminated.

*

main difference b/w nested & Sub function

↓

'Scope'

* Usual: each function has its own scope! - Invisible to all statements outside the function!
* Nested function: → 'Social media' has not only access to all the variables inside it but also the variables inside its parent function.

'Benefit eg sharing variables b/w nested function & its parent' → makes it simpler to pass info back & forth b/w the two functions

↓

Comparing passing arguments. (B/w sub functions).

↓

No need for arguments - access directly.

'more efficient' - when parent wants the sub-function to change the values in a large array. [Instead eg copying - directly access, edit]. → No need to return (saving time & memory]

'work together more efficiently'.

» [a b] = first_nested_example(3)

multiplying 10 times 3
multiplying 11 times 3

a = 30
b = 33
} change.

Difference b/w nested function & anonymous function:

30 → change c → 33 change ans

In this example if c changed, the changed value will be used by nested function.

Anonymous function fixes c at the time of definition - even if c was cleared - works fine.

If c updated → Ano. fun remains same

↓ (different than global) → global inside main func

↓ Not global (got fixed) at defining function

'Global variables' → prone to bugs.

Exception:

* c can't be a i/p or o/p argument
* Since ↙ it will be a formal argument (showcase)

```
function out = inmeas(in)
    out = c*in;
```
→ 'Not i/p or o/p argument'

```
end
```

'Assign it in parent & use it in child' → Rule of thumb

eg:

```
function assignment_rule
    r=7;
    calculate_area
    fprintf('Area of circle with radius %.1f =
        %.1f\n', r, circle_area)

    function calculate_area
        circle_area = pi*r^2;
    end
end.
```

For avoiding inconsistency 2018b introduced this!

∴ circle_area is inside nested function - not assigned / initialized in the parent. So we can't access in parent function! → 'Error'

what Inconsistency?

If there is a function named circle_area on the path one time & not another time.

Repair: use an assignment of the empty array to circle area in the parent function like this.

∴ circle area is considered as a variable instead of function!

↓

noted: same name is used (colon in editor) in the nested function!

Try! (me)

```
c = 1; a = @(x) x+c;
a(1)
2
>> c = 100;
>> a(2)
3
```

→ Anonymous function fixes c at assignment! never updates.

'Tested'

first_nested_example.m.

'over case'

```
function assignment_style
    r = 7;
    circle_area = calculate_area;
    fprintf ("Area = %.d", circle_area);
    function circle_area = calculate_area (r)
        circle_area = (r^2) * pi
    end
end
```

↓

'make as o/p'

↓

make as o/p

```
circle_area = []
function circle_area =
function calculate_area
    circle_area = (r^2) * pi;
end
```

↓

Assign (won't be mistaken as function)

↓

Not necessary to be at the top (Anywhere inside parent function)

Just as an editor: Matlab scans the entire m files → looking for errors & determining the structure. 'During scanning - it sees the empty assignment'.

* 'circle_area' → as shared variable!

Note: All i/p & o/p functions of arguments of parent functions are accessible by their nested function!

'i/p and o/p arguments of parent: accessible by the nested function of it.

eg:

```
function circle_area = assignment_rule(r)
calculate_area
fprintf('Area = %.1f \n', circle_area)
function calculate_area
        circle_area = pi * r^2;
end
end.
```

```
>> assignment_rule(4)
Area of circle with radius 4.0 = 50.3
ans =
   50.2655.
```

↓

No need for empty assignment (as it is an argument).

'Nested function inside another'

'matlab allows descendants as deep as you want'!

---

* C can access both access its parent & grand parent!

* 'Non-global scope of a variable comprises the bodies of every function that is a descendant of the function in which the variable is defined'

* XA accessible by everyone. [Non local scope of A is B, C & D].
* Non local Scope of XB includes XB and XC not XD
* XC & XD are no non-local scope at all.

```
function A
    XA = 1;
    function B
        XB = 2;
        function C
            XC = 3;
            Show ('C', 'XA', XA)
            Show ('C', 'XB', XB)
            Show ('C', 'XC', XC)
        end % C
        Show ('B', 'XA', XA)
        Show ('B', 'XB', XB);
        C
        D
    end % B
    function D
        XD = 4;
        Show ('D', 'XA', XA);
        Show ('D', 'XD', XD);
    end % D
    Show ('A', 'XA', XA).
    B
    D
end % A

function show (funct, name, value)
    fprintf ('\n%s : %s = %d \n', funct, name, value);
end
```

C

B

D

A

sub function

* child (nested) can access parent.
* until initialized / arguments - parent can't access child.

∴ A can access only X_A
∴ B can access X_A and X_B
∴ C can access X_A, X_B & X_C
∴ D can access X_A & X_D

→ clean picture on scope (local)

Rule of accessibility

* 'A parent can call a child' - Not grand child.
* A - can call B & D
* A sibling can call sibling. (B can call D)
* A descendant can call any ancestor!

  'C can call B/A'

Sibling calling feature

* Sibling functions can call each other but they can't access each others local variables.
* calling an ancestor - recursive!

↓

'having base case & recursive case is must!'

↓

Else ∞ loop

---

* A function can return a handle to its nested functions! → Comes handy.

```
c = 10;
f = @(x) c*x;
f(3) → 30
c = 11
f(3) → 30
clear c
f(3) → 30 → Still works
```

Still

→ we need to redefine the function for updating 11.

'Save time! write function'

```
function fh = get_anon_handle(c)
    fh = @(x) c*x;
end
```

↓

```
>> f10 = get_anon_handle(2)
>> f11 = get_anon_handle(3)
```

Build new anonymous function!

* Anonymous functions can have more than 1 statements!

  Conditional & for loops.

1 2 4

$x^2 + 2x + 4$

```
function fh = get_polynomial_handle(P)        % P → i/p argument [accessible by nested]
    function polynomial = poly(x)
        polynomial = 0;
        for ii = 1:length(P)
            polynomial = polynomial + P(ii) .* x.^(ii-1)
        end
    fh = @poly;
end.
```

get_polynomial_handle(P) → creates a polynomial & assigns it as handle to fh.

↓

Polynomial creating (won't produced in a single step)

* Nested function comes handy!

```
function out = sample_poly
out = @(x) x^3 + x^2 + x + 1
End
```

→ o/p make a function return as handle.

(or)

"Is it possible without nested function"?

"No!" → Unrecognized variable ×

↓

* Build function : (nested)
* make that function as handle.

* function polynomial = poly(x) → Build polynomial using P (parent variable)
* make as handle.

>> fplot(fpc, [-3, 2]); hold on; fplot(fpv, [-3, 2])

↓ $-4x^3 - x^2 + 3x + 1$

↓ $-10x^2 + 7$.

! A sibling can call sibling but can't access each others variables

1950s! → Nesting fun was invented (Algol) — 20 years before MATLAB.

Algol - Algorithmic language

Concept in nested function: automatic read - write access ability, to variables in it's ancestors → through non-local scope almost died!

↓

Not supported by c/c++/Java

↓

MATLAB keeping it alive.

(simplify the passing eg info back & forth b/w functions) → while avoid confusion eg global variables & toughness eg passing parameters.

1:5

$(1 \times x^0) + (2 \times x) + (3 \times x^2) + (4 \times x^3) + (5 \times x^4)$

$\ldots + (10 \times x^9)$

↓

pf(2) = 1023.

1:5 → one o/p

2 → one o/p.

* create something in a function using 1:5
* make handle in other function.

@ functions

p = fun(1) → p = (handle to a polynomial)

p(1) → give value to that handle.

Polynomial without loop

```
function  ans = poly_fun (P)
    function polynomial = poly(x)
        polynomial = sum (P. * (x. ^ (0: length (P) - 1)));
    end
    ans = @ poly;
end
```

↓

single time : a = poly_fun (1:5)

```
function  ans = poly_fun (P, x)
    ans = sum (P. * (x ^ (0: length (P) - 1)));
end.
```

↓

Set a polynomial then give x values (2 functions)

1. Set Polynomial

Calculate x.

'mixed mode arithmetic'

'double + integer' → like.

single: 32 bits - single precision floating point

double: 64 bits - double "

int8: 8-bit integer [-128 ... 127]

uint8: 8 bit unsigned int [0 - 255] → slide.

>> 2 * pi

6.2832

>> int8 (200) + int8 (300)

int8

127 → Fitted

→ Clipping.

» -3 → unary operation.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}^{\wedge} 11$$

Error: Not Suitable (element wise possible) (or) use multiplication.

(2 x 3) (2 x 3)

Not possible.

Personal branding

Honesty, Creativity & Integrity.

'Arrays & Vectors' with equal row or column length: added & sub.

```
>> A = [100 200 300; 400 500 600];
>> C = [7;9]
>> A = A+C;
```

→ Rule no: Columns or Rows must be equal.

```
107  207  307
409  509  609.
```

↓

we need this → many problems

add a row to every row, add a column to many column.

↓

Built in function – Before 2016 this feature not available.

repmat → copies array.

```
C = [7;9]
C = repmat (C, 1, 3) → (1 by 3) copies
```

↳ No copies row wise

↳ 3 copies column wise

```
7  7  7
9  9  9
```

Now we can $\begin{bmatrix} 100 & 200 & 300 \\ 400 & 500 & 600 \end{bmatrix} + \begin{bmatrix} 7 & 7 & 7 \\ 9 & 9 & 9 \end{bmatrix}$

↓

But using several times! Duplication.

Mixed mode Arithmetic

```
>> Int8 (4) + Int16 (5)   → Can't be added.
>> Int8 (4) + Single (5)  → Can't be added.
```

Not allowed.

int8(123) – int8(40) → Rule 2

uint16([12; 34]) + uint16([64 28; 10 55]) → Rule 5.

'Same type: allowed'

'we can do +, −, .* (dot) in same operating data types

we can't mul (or) div

uint16([1 2; 34]) * uint16([64 28; 10 55])

uint16([1 2; 3 4]) * uint16(64)

error: MTIMES (name of the function doing *)

---

mtimes – matrix multiplication

$$C(i,j) = \sum_{k=1}^{p} A(i,k)\,B(k,j)$$

C = mtimes([1 2 3 4; 1 2 3 4; 1 2 3 4],
[1 2 3; 1 2 3; 1 2 3; 1 2 3])

C =

10	20	30
10	20	30
10	20	30

'matlab's arithmetic operation has an equivalent function

plus, minus, power, mpower etc...

help: Arithmetic.

---

* mtimes → Not fully supported for integer classes.

↳ Atleast one operand is scalar (works fine!)

scalars

>> uint16(4) * uint16([64 28; 10 55]) → scalar works fine!

256 112

40 220.

In case of doubles - peachy

uint16([1 2; 3 4]) * 64

64 128
192 256

4 * uint16([64 28; 10 55])

256 112
40 220.

"No problem: when a scalar is in action"

Problem: uint16 (vector - matrix) × uint16 (matrix)

etc...

---

* Single (1) + double (2) = 3 (single)

Least restrictive to most restrictive.

Binary arithmetic operations, x op y

	Type-Shape Combinations	Allowed operators
Least	1) x, y = Floating point numbers	All
	2) x, y = scalars: Integers of same type or Integer & double	All
	3) x = int array, y = scalar int (same type/double)	All except \ and ^
	4) y = integer array, x = scalar integer of same type or scalar double	All except / and ^
	5) x, y → Non-scal → Integers of same type	*, - & .op
most	6) x, y = int of diff types (ex) int & single	None.
(random)	Least to most restrictive	1, 2, 6 → same mode 3, 4, 5 → mixed mode.

Exception: we can't raise an integer to a fractional power.

>> int64 (3) ^ 2

(9)

>> int64 (9) ^ 0.5

"Error"

>> sqrt (int64 (9))

↓

Not defined for int64.

↓

Not possible (Fractional)

* 'Didn't use a fractional power' – Integer

* what type – returned?

>> Type of o/p – any legal arithmetic operation – Same as the o/p that occupies least space & memory.

* int , double = int (8 bit)
  (64 bits)

* single , double = single (32 bits)

Same type → Same type (output too).

>> x = 12	n*x , n/x , n^x	>> x + int16(9876)	>> n = int16(9876);
x = 12	32767 → int16	ans = int16	>> x = 12;
double	823 → int16	9888	>> x/n
	32767 → int16.		ans = 0

∴ x/9876 = 0.0012 → But → why?

Answer

"Integer can't hold a fractional result: double/int16 = int16"

↓

we get int16 (closest) as answer = 0.

0.6 ≈ 1 rounding.

"Consequence of having smaller space occupying data as o/p to mixed-mode arithmetic"

double & int64 → int64 (o/p)

(even in this case rounded off)

Hazard! (Involves integer!)

why? this small memory datatype as o/p?

"Save memory!"

('Smaller size - Save memory')

eg: Logo: 400 x 400 x 3 uint8

'each element occupies 1byte'

400 x 400 x 3 = 480000

= 480 KBytes.

= 0.48 MBytes (480,000 bytes)

double

(400x400x3)x8byte = 3840000 bytes = 3.84 MBytes.

make doorkeen

```
>> M=imread('matlab.png');
>> imshow(M)
>> whos
   M = 480000 bytes.
>> D=M/3 ;
>> imshow(D)
>> whos
   D = 480000 bytes
```

o/p will be same type. (smallest operand type)

---

widening: done by C, C++, Java → In mixed mode arithmetic (making o/p to longer data type).

MRI: 100 images) Huge (GB)
videos: 60 frames ↓
Factors eg 8 becomes huge!

'Save memory' — Save time

narrowing - Saves memory: (longer bit (double) takes more processing time).

* Difference eg 0.5 → Rarely detectable in an image! → Lot of sense in Image processing.

'we can do by making it double' - If we need.

'Bargain accuracy to memory & time'.

```
>> a = int8(17)
>> b = double(a)/2  ⟶  17/2 = 8.5
>> c = double(a/2)  ⟶  double(9) = 9
```
] Different.

Edge detection

Sobel operation

$$M = \sqrt{S_x^2 + S_y^2} \longrightarrow \text{For Edge detection.}$$

edges: highlighted by gray/white ↔ others black.

Automatically detecting edges in images – Important in Image processing. An edge detection takes an I/P & generates another img where the edges in the original img are highlighted by gray/white colors while other pixels are black. For each

3×3 pixel subset A, of an image: we calculate the magnitude of the gradient at the central pixel of A as a weighted sum of all values in A.

Sobel operator

$$M = \sqrt{S_x^2 + S_y^2}$$

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} : A \qquad , \qquad S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} : A.$$

↓ double dot operator.

Each pixel S_x & S_y will be a weighted sum of values of the neighbouring pixels in the original img using the weights specified in the 3×3 matrices above.

↓

S_x → current pixel & its top & bottom neighbours – not used (weighted as zero)

Top right neighbour → weight 1

Top left neighbour → weight -1.

Final o/p pixel → root sum squared Sx & Sy.

Solution

* Take care of multiplying Sx & Sy → each matrix (3 by 3)

* $\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$ → Takes care of root.

$\begin{bmatrix} 400 \times 400 \\ \end{bmatrix} \rightarrow \begin{bmatrix} 400 \times 400 \end{bmatrix}$ Same size

I/P O/P

Each element has matrix

$\begin{bmatrix} 1 & \cdots & 400 \\ \vdots & & \\ 400 & \cdots & 400 \end{bmatrix}$

Go from 1 to 398

$\begin{bmatrix} 398 & .. \\ 399 & .. \\ 400 & .. \end{bmatrix}$ 3x3

Last two rows - empty (Since

$\begin{pmatrix} 398 & 398 & 398 \\ 399 & 399 & 399 \\ 400 & 400 & 400 \end{pmatrix}$ → Last row

Column

$\begin{bmatrix} 398 & 399 & 400 \\ \vdots & \vdots & \vdots \end{bmatrix}$ 3x3

∴ Remove them.

"See Github"

Convert all to a range

$2 \times \frac{0}{65535} - 1 = -1$ → Bringing all to this range.

$2 \times \frac{65535}{65535} = 1$

exceeds 1 → divide by maximum

multiply by N.

multi track - audio - uint16 matrix of N columns - each column - 1 track.
(Recording of one band playing a song).

Range: 0 to 65535 $= 2^{16} - 1$

eA digital from analog conversion would provide.

Task: write simple mixing function that takes the tracks & generate a weighted sum of them.

Function: mix1k → K by N matrix
→ N (no. of tracks)

* K - No. of samples per track
* N - No. of tracks

* N (second I/P) - double scalars representing the weights of the tracks. The o/p of function: K element column vector: double: single track audio recording - (mixing individual tracks - static weights).

Interval: [-1 1]

0 → -1

65535 → 1

eIf exceeds - scale by absolute maximum point.

Solu:

Steps: * multiply by weighted of the tracks to M
* Then scale.

$$\begin{bmatrix} | & | \end{bmatrix} \quad [1 \quad 1] \rightarrow \begin{bmatrix} | & | \end{bmatrix} \quad \begin{bmatrix} | \end{bmatrix}$$

multiply each column by its weight.

---

make 0 → -1

65535 → 1

$$\therefore 2 \times \frac{65535}{65535} - 1 = 1 \qquad \left| \quad 2 \times \frac{0}{65535} - 1 = -1. \right.$$

'Find maximum of absolute value of the o/p vectors: > 1 → Divide entire vectors by that value'

1) matrix mul $(m \times n)\ (n \times p) = (m \times p)$

<u>caution</u>

abs (max (matrix))

↓

(-)ve elements omitted

we need absolute value to compare.

```
function ans = mixit (A, N)
if size (A,2) ~= length (N)
        ans = [];
else
    A = double (A);
    A = 2* (A/65535) - 1;  ⟶ Constant div
    ans = A * N (:);
    while max (abs(ans)) > 1
        ans = ans/max (abs(ans));
    end
end
end.
```

Linear equations

eqn: $AX = b$

↓ matrix ↳ column vectors.

Simultaneous linear algebraic equations

$$Ax = b$$

$$\begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1N} \\ A_{21} & A_{22} & \cdots & A_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ A_{M1} & \cdots & \cdots & A_{MN} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_M \end{bmatrix}$$

$$A_{11}x_1 + A_{12}x_2 + \cdots + A_{1N}x_N = b_1$$
$$\vdots$$
$$A_{M1}x_1 + A_{M2}x_2 + \cdots + A_{MN}x_N = b_M$$

→ In to eqns

M equations involves N elements A, x & 1 element eg b.

∴ $x_1, x_2, \cdots, x_N$ → common

∴ $x_1, \cdots x_N$ → should satisfy all eqns (M eqns) simultaneously

"simultaneous linear eqns"

---

* why linear? No unknowns (x) multiplied by eachother / themselves
* eg: No x_1^2, $x_1 x_2$ etc.... → linear.

∴ Algebraic expressions

simultaneous linear algebraic equations

---

* motto: Solve for x - which solves all M eqns simultaneously

Powertools: MATLAB

$$4x_1 + 5x_2 = 6$$
$$3x_1 - 2x_2 = 14$$

$$\begin{bmatrix} 4 & 5 \\ 3 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 6 \\ 14 \end{bmatrix}$$

```
>> A = [4  5 ; 3  -2];
>> b = [6  14]
```

A * x = b

∴ A/b = 10/5 = 2

b\A = A/b = 10/5 = 2 → same: matrix

$Ax = b$

$x = b/A$

$x_1 = 3.5652$

$x_2 = -1.6522$.

↓

Forward slash doesn't work" put $x = A \backslash b$

∴ $x = A/b$
∴ $x = b/A$
] matrix dimensions must agree.

Use backward slash for matrix division

For forward slash: same no. of columns & same no. of rows for

backward slash.

Our case

A multiple columns, b has 1 column.

↳ Forward slash won't work

1 row & 1 row → Backslash work.

$x = b/A$ ⟶ » $x = A \backslash b$ (matlab) ✓

```
» rng(0)
» A = rand(20,20);
» b = rand(20,1);
» x = A\b;
» e = A*x - b;
» max(abs(e))
```

ans =

$8.8818\ e^{-16}$ ⟶ very very minimum

$Ax = b$

$Ax - b = 0$ (Ideal)

"may be some error"

e – error vector.

" Point eg pride : \ gives world class solution to any linear algebra problem"

* C, C++, Java → doesn't provide this matrix solving!
(No inbuilt)

Function

mldivide → matrix left divide

↓

same result as \ operator.

"For some evu: we can't get a good solution"

* Inconsistent info

* eg:

$$\left.\begin{aligned} 4x_1 + 5x_2 &= 6 \\ 4x_1 + 5x_2 &= 12 \end{aligned}\right\} \text{? (Inconsistent)}$$

"No solution exist"

Ans =

$-\text{Inf}$
Inf) From MATLAB.

"singular to working precision"

'when the solution is singular to working precision: it can't be solved'.

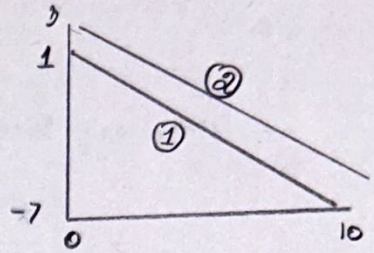
we can visualize: 2d: (2 unknowns): In a plot. (any eqn)

>> x1 = 0 : 0.01 : 10;

>> x2 = (6 - 4 * x1) / 5; → $\frac{6-4x_1}{5}$ (Same no. of points – x1, x2).

>> plot(x1, x2); grid on.

>> x2 = (12 - 4 * x1) / 5; $\frac{12-4x_1}{5}$.



'Linear'

$4x_1 + 5x_2 = 6$

$4x_1 + 5x_2 = 12$

$x_2 = \frac{6-4x_1}{5}$

$x_2 = \frac{12-4x_1}{5}$

From graph: 'No Crossover – Common points' → Nothing.

'parallel lines – Intercept at no where'

↓

Ans: (∞, −∞) → way off to ∞

'Matrix : singular to working precision: double'

↓

some round off errors : causing the matrix appears to be singular: It is not singular: Instead of being parallel, these lines are like just the tiniest bit. (such a tiny bit = Solution far away from the origin that the magnitude of x_1 & x_2 are greater than the largest precision of double – it can hold.

Any number > double precision can hold : ∞ (INF)

'Same holds for singular precision'

For matrix: Its determinant = 0

'Some algorithms: A^{-1} → Involves dividing by determinant'

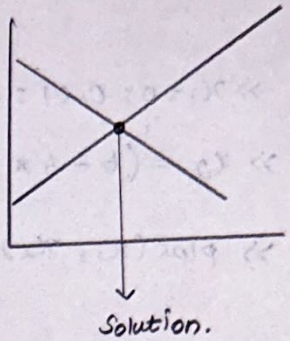
$\frac{\text{Non zero}}{0} = (\text{Sign of Non zero})\ \infty$

'sign of ∞' → $x_1 \to \infty$ (or) $x_1 \to -\infty$

'which direction'

'They never intersect : intersect at ∞'

```
>> x1 = 0:0.01:10;
>> x2 = (6 - 4*x1) / 5; plot(x1, x2)
>> x2 = (14 - 3*x1) / (-2); plot(x1, x2)
```



'For consistent set of eqn: intersect'

Inconsistent: 'never'

∴ Lines only intersect - is eqn consistent? ⟶ No

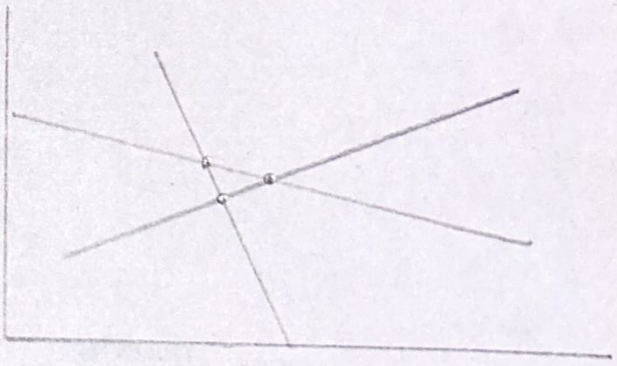
★ eg: Inconsistent - but lines intersect!

$$4x_1 + 5x_2 = 6$$
$$3x_1 - 2x_2 = 14 \longrightarrow \text{'overdetermined equations'}$$
$$7x_1 + 2x_2 = 19$$

(3 eqn: instead of 2' - only 2 unknowns)

'more eqn than unknowns'

★ Each intersection: Solution for a pair of equations.



'Not a Common Solution'

↓

overdetermined!

'Different intersection points'

'Inconsistent'

'Not exist!' - Almost when no. of eqn > unknowns

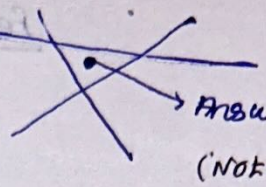
Use backslash

$$A = \begin{bmatrix} 4 & 5 \\ 3 & -2 \\ 7 & 2 \end{bmatrix}, \quad B = \begin{bmatrix} 6 \\ 14 \\ 19 \end{bmatrix}$$

A \ b Ans: 3.2853
-1.5754

⊃ perfectly normal!

Let's plot: Plot (x(1), x(2), '*')



→ Answers (Not Intersection)

Why? matlab gives solution?

'Not an accident'

* 'Closest' - which solves all → Special way (determined)

e (error) = Ax − b

usual case : Ax − b = 0

e =

−0.7358

−0.9934 → with maximum absolute values.

0.8462

'Squared error' = Sum of all the elements of e.

SSe = Sum (e.^2)

SSe = 2.2443

∴ matlab gives answer: which has the least SSe.

↓

Least Square Solution (most likely to be the best solution)

In Engg. linear eqn algebra

eps (epsilon) =

$2.2204e^{-16}$ (difference b/w 1.0 & the next largest no. a double precision can grap.)

If the SSe → more than 1000 times of eps

↳ Sure * overdetermined eqn

* Typed something wrong!

Few eqns than unknowns

'underdetermined' – equations

① $2x_1 + 10x_2 = 4$

② $2x_1 + 3x_2 - 6x_3 + 7x_4 = 5$

$3x_1 - 2x_2 + 4x_3 + 9x_4 = 14$

$$A = \begin{bmatrix} 2 & 10 \end{bmatrix} \quad B = \begin{bmatrix} 4 \end{bmatrix}$$

$$A = \begin{bmatrix} 2 & 3 & -6 & 7 \\ 3 & -2 & 4 & 9 \end{bmatrix}, \begin{bmatrix} 5 \\ 14 \end{bmatrix} = B$$

use backslash

∞ number of solutions.

```
A = [2 10]; b = [4];
x = A\b
    0
    0.4000
```

```
x = A\b
    0
    0
    0.6463   → set 2 unknowns as
    1.2683     zero.
```

'Any one zeros : probably underdetermined'

'Stability of Solution'

* check for a given set of equations

$Ax = b$, changes on x have on b.

* $Ax = B$ (must be solved for x) – elements in b measured values (say)

'Any measurement' – subject to errors

'Small change in b – drastic change in x' → dilemma.

'Ill Conditioned' → find where is the problem.

* Small for x, make small changes in b - Solve again.

'more than 50% - if 1% change' → Ill conditioned!

Better way! - than do 1000 times!

* Cond → Condition - look at the matrix - tell how ill conditioned!

* eg: $27.3x_1 + 59.4x_2 = 78.1$

$63.2x_1 + 33.4x_2 = 91.1$

```
>> A1 = [27.3 59.4 ; 63.2  33.4];
>> Cond(A1):
2.9640
```

```
>> A2 = [41.9  59.1 ; 57.5  81.1]
>> Cond(A2)
ans =
   9.4574 e+04   → Ill conditioned.
```

'Bigger - worse' - max factor by which error % can ↑ from the i/p to o/p s.

Typical measurement % → 1 (or) 2 %

Example 1: No bigger than 2.964 %

2: Large as (error) → 94,574 % → huge!

'Depends on accuracy'

$$\text{equal to} = \frac{\text{max \% error (tolerate) in o/p}}{\text{max \% error expect in i/p}}$$

Saw some i/ps

```
b1 = [78.1 ; 91.1];
x1 = A1 \ b1
0.9861
0.8616
```

```
b1_error = b1 + [1.0 + 0.73]
>> b1_error =
   79.100
   91.830
```

```
pct_b1_error = 100* norm(b1_error - b1)/norm(b1)
```

$\frac{\text{new} - \text{old}}{\text{old}} \times 100$

```
>> 1.0318 (1 % error)
```

norm → +ve (euclidian)

```
x1_error = A1\b1_error
```

0.9896
0.8768

o/p change:

```
pct_x1_error = 100* nor(x1_error - x1)/nor(x1)
```

1.1930 → Not a huge change!

```
>> Cond(A1)
2.9640
```

→

```
>> 2.9640 * 1.0583
3.0583
```

↳ i/p change made.

Second eqn

```
b2 = [58.9 ; 80.8];
x2 = A2\b2
x2 =
   -9.4375
    7.6875
```

```
b2_error = b2 + [1.0; 0.73];
b2_error =
   59.9000
   81.5300
error: 1.2382
```

% change ⇒ 2389.1 % error!

'Disaster'

3 eqn: $41.9x_1 + 59.1x_2 = 58.9$

$57.55x_1 + 81.1x_2 = 80.8$

$69.9x_1 + 31.7x_2 = 31.7$

Cond(A3)

4.1781 → we can live with it!

For 1% i/p error → 4% o/p error

Voltage → A, B, C (Computes voltage at)

I/ps → V – Supply

→ R – Resistors in ohm

KVL:

$\Sigma V_K = 0 \quad (V = IR)$

$\Sigma I_K = 0 \quad (I = \frac{V}{R})$

$\frac{V-A}{R_1} + \frac{A-B}{R_7} - \frac{A}{R_2} = 0$

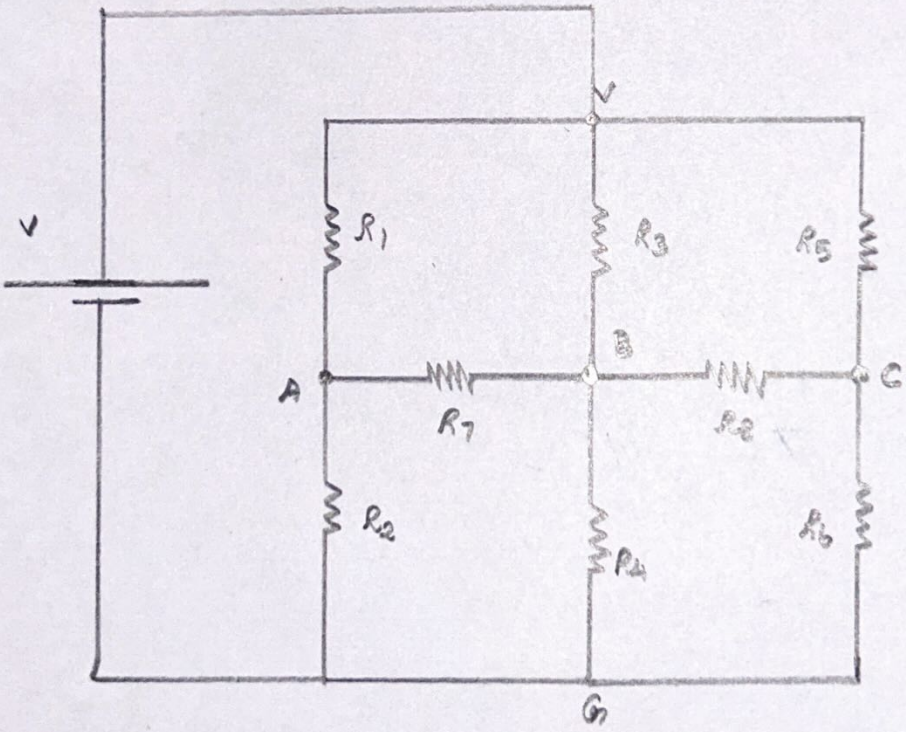
(Sum of current flowing in & out of a junction = 0)

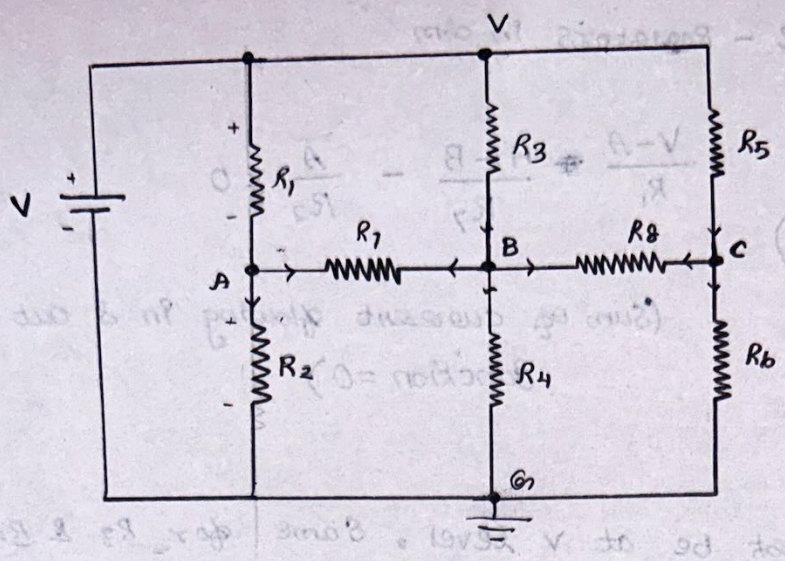
Assume: $A > B$

* $R_1 = 0$ means ⇒ A must be at V level, same for R_3 & R_5 for B & C resp.

* $R_2 = 0$, makes $A = 0$. Same for R_4 & R_6 for B & C resp

* $\frac{R_1}{R_2} = \frac{R_3}{R_4} = \frac{R_5}{R_6}$, then A, B & C will be at same level Ind. of R_7 & R_8





Incoming current = outgoing current.

① $\frac{V-A}{R_1} - \frac{A-B}{R_7} - \frac{A}{R_2} = 0$

② $\frac{V-B}{R_3} - \frac{B-A}{R_7} - \frac{B}{R_4} - \frac{B-C}{R_8} = 0$

③ $\frac{V-C}{R_5} - \frac{C-B}{R_8} - \frac{C}{R_6} = 0$

$\frac{V-A}{R_1} - \frac{A}{R_1} - \frac{A}{R_7} + \frac{B}{R_7} - \frac{A}{R_2} = 0$

$\frac{V}{R_1} = \left(\frac{A}{R_1} + \frac{A}{R_7} + \frac{A}{R_2}\right) - \left(\frac{B}{R_7}\right)$

$\frac{V}{R_3} - \frac{B}{R_3} - \frac{B}{R_7} + \frac{A}{R_7} - \frac{B}{R_4} - \frac{B}{R_8} + \frac{C}{R_8} = 0$

$\frac{V}{R_3} = \left(\frac{B}{R_3} + \frac{B}{R_7} + \frac{B}{R_4} + \frac{B}{R_8}\right) - \frac{A}{R_7} - \frac{C}{R_8}$

$\frac{V}{R_5} - \frac{C}{R_5} - \frac{C}{R_8} + \frac{B}{R_8} - \frac{C}{R_6} = 0$

$\frac{V}{R_5} = \left(\frac{C}{R_5} + \frac{C}{R_8} + \frac{C}{R_6}\right) - \frac{B}{R_8}$

$$\begin{bmatrix} \left(\frac{1}{R_1}+\frac{1}{R_7}+\frac{1}{R_2}\right) & -\left(\frac{1}{R_2}\right) & 0 \\ -\frac{1}{R_7} & \left(\frac{1}{R_3}+\frac{1}{R_4}+\frac{1}{R_7}+\frac{1}{R_8}\right) & -\frac{1}{R_8} \\ 0 & -\frac{1}{R_8} & \left(\frac{1}{R_5}+\frac{1}{R_6}+\frac{1}{R_8}\right) \end{bmatrix} = \begin{bmatrix} \frac{V}{R_1} \\ \frac{V}{R_3} \\ \frac{V}{R_5} \end{bmatrix}$$