جـــامـــعـــة دمـــشـــق
كـلـيـة الـهـنـدسـة الـمـدنـيـة

تـصـمـيـم بـمـعـونـة الـحـاسـب
Computer Aided Design

Lecture 2

Matrices & Control Structures

18/11/2025

Wael Darwich

Damascus University

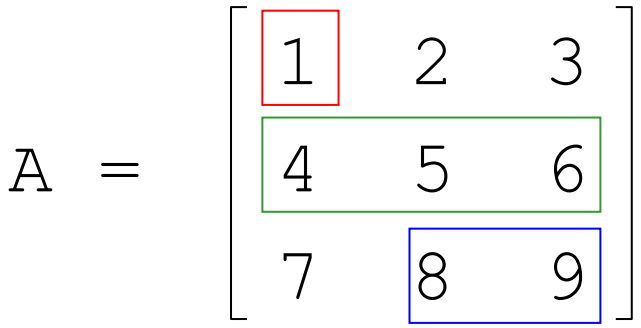
Civil Engineering Faculty

Last Lecture

- Introduction
- Memory and Variables
- Operators
- Scripts vs. Functions
- Matrix Calculator
- Colon :
- Semicolon ;
- Comma ,

Matrix Index

```
A=[1 2 3; 4 5 6; 7 8 9]
```




```
B = A(1,1)
C = A(2,1:3) = A(2,:)
D = A(3,[2 3]) = A(3,[2:3]) = A(3,2:3)
A(1, 1) = 4
```

Content

- Matrix Calculator Continued…
  - Linear Equations
  - Symbolic Math Toolbox
- Control Structures
  1. Conditions
  2. Loops

Vectors

Colon :

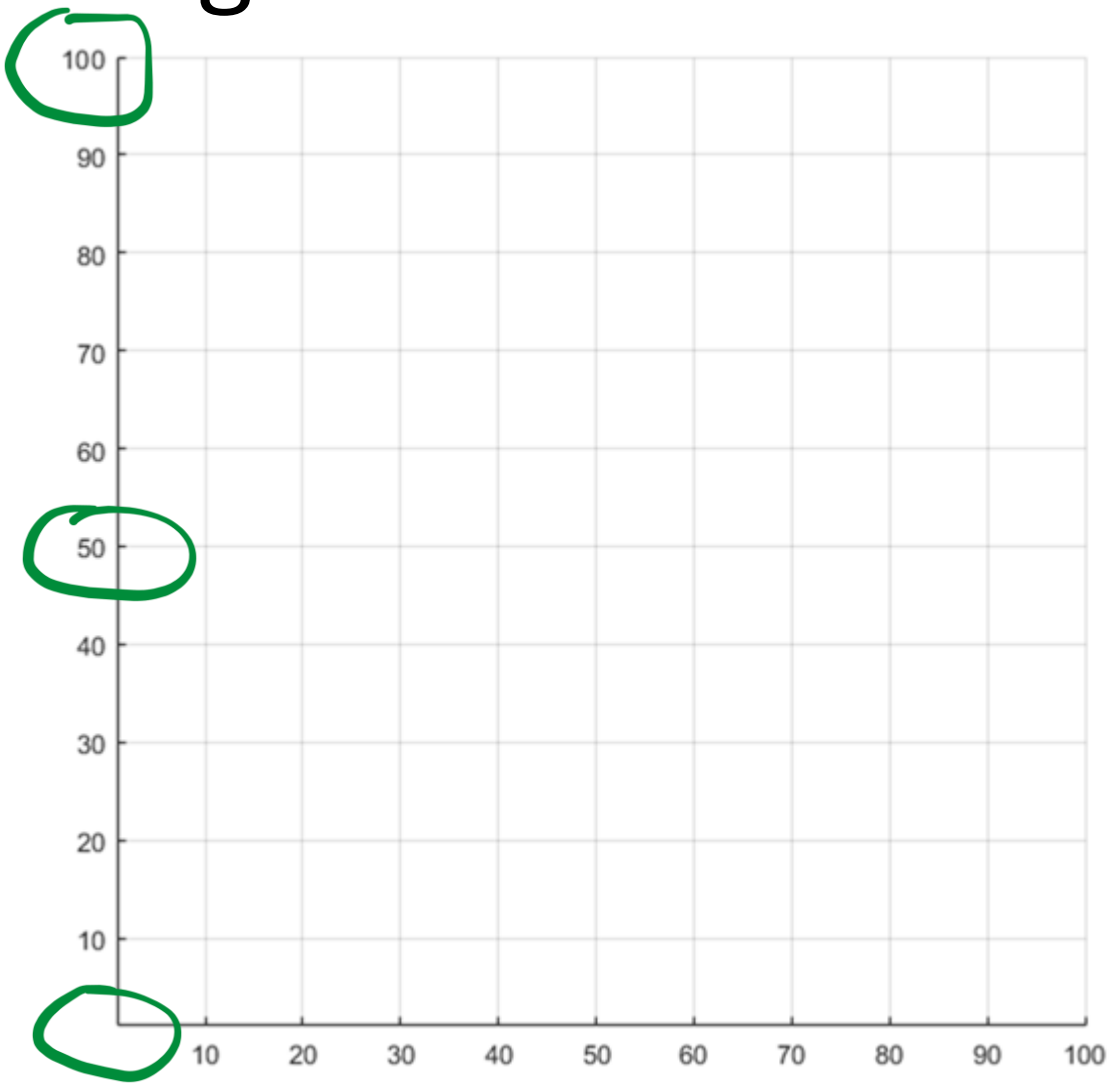
- `start : end`
  - `1 : 5` `[1 2 3 4 5]` `1×5 vector`
  - `5 : 1` `[]` `1×0 empty vector`
- `start : step_size : end`
  - `1 : 2 : 5` `[1 3 5]`
  - `1 : 2 : 6` `[1 3 5]`
  - `5 : -2 : 1` `[5 3 1]`
  - `5 : -2 : 0` `[5 3 1]`
- Exact start and step size, *but not end*

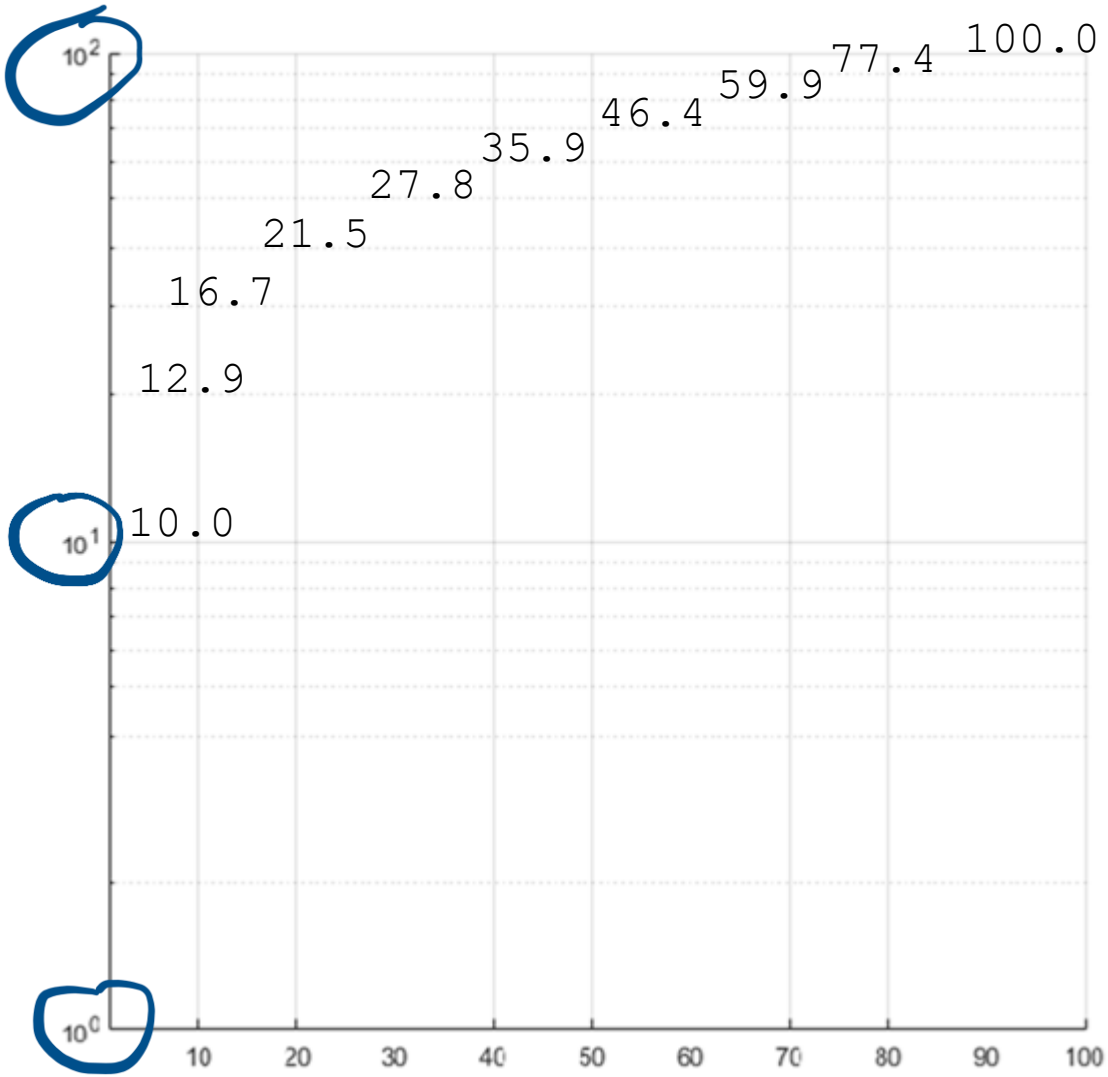
Vectors

- `linspace` — Linearly spaced vector
- `linspace(X1, X2, N)` — Generates N points between X1 and X2
  - `linspace(0, 4, 2)` → `0  4`
  - `linspace(0, 4, 3)` → `0  2  4`
  - `linspace(0, 6, 2)` → `0  6`
  - `linspace(0, 6, 3)` → `0  3  6`
  - `linspace(1, 5)` → `1  1.0404  1.0808  …  4.9192  4.9596  5`

  Default **`N = 100`**
  - `linspace(1, 100)` → `1  2  3  …  98  99  100`

Logarithmic Scale





Vectors

- `logspace` **Logarithmically** spaced vector
- `logspace(X1, X2, N)` Generates N points between X1 and X2
  - `logspace(0, 2, 2)` `1 100`
  - `logspace(0, 2, 3)` `1 10 100`
  - `logspace(1, 5, 2)` `10 100000`
  - `logspace(1, 5, 3)` `10 1000 100000`
  - `logspace(0, 1)` `1 1.0481 1.0985 … 9.1030 9.5410 10`
- Default **N = 50**

Matrices: Creation

- `zeros`

`zeros(3)=` $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

`zeros(1, 4)` $= \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}$

- `ones`

Matrices: Creation

- `eye`

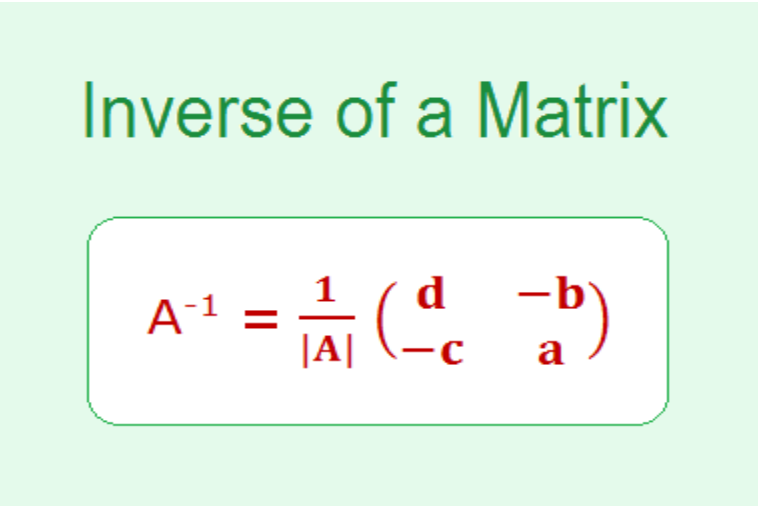
$$\texttt{eye(3)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\texttt{eye(1, 3)} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

$$\texttt{eye(3, 1)} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

Matrices Functions

- A = $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$
- Determinant = |A| = ad - bc
- Inverse

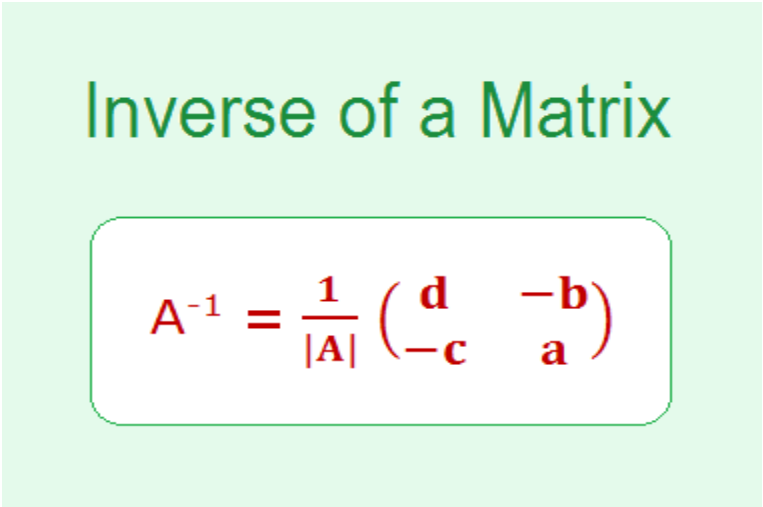


Matrices Functions

- `det` Determinant
- `inv` Inverse

Matrices Functions

- A = $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$
- Determinant = |A| = ad - bc
- Inverse




```
a =

     1     2
     3     4

>> det(a)

    -2

>> inv(a)

   -2.0000    1.0000
    1.5000   -0.5000
```

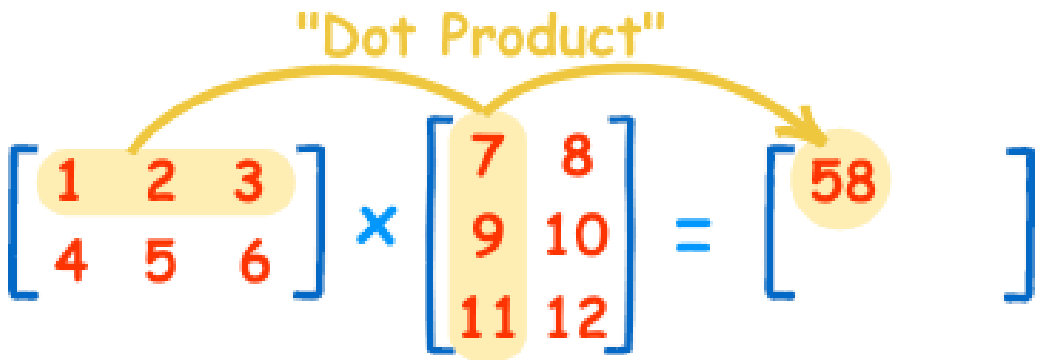
Matrices Functions

- `transpose` `'`

```
>> [1:4;5:8]

ans =

     1     2     3     4
     5     6     7     8

>> transpose([1:4;5:8])

ans =

     1     5
     2     6
     3     7
     4     8

>> [1:4;5:8]'

ans =

     1     5
     2     6
     3     7
     4     8
```

Matrices Functions

- `a * inv(a)= eye(2)`
- `a * a = a ^ 2`
- `a .* a = a .^ 2`




```
>> a = [1 2;3 4];
>> disp(a * inv(a))
    1.0000         0
    0.0000    1.0000

>> disp(a * a)
     7    10
    15    22

>> disp(a .* a)
     1     4
     9    16
```

Matrices Functions

- `repmat` Replicate and tile an array

```
>> a = [1 2;3 4];
>> disp(repmat(a, 2, 3))
     1     2     1     2     1     2
     3     4     3     4     3     4
     1     2     1     2     1     2
     3     4     3     4     3     4
```

Matrices Functions

- `reshape` Reshape array

```
>> b=[1:3;4:6]

b =

     1     2     3
     4     5     6

>> reshape(b, 3, 2)

ans =

     1     5
     4     3
     2     6
```

Matrices Functions

- `min` Minimum elements of an array
- `max` Maximum elements of an array
- `sum` Sum of elements (by columns)
- `mean` Average or mean value (by columns)

```
c =

     1     2     3
     4     5     6
     7     8     9

>> disp(min(c))
     1     2     3

>> disp(max(c))
     7     8     9

>> disp(sum(c))
    12    15    18

>> disp(mean(c))
     4     5     6
```

Matrices

- num2str

```
num2str([1:3;4:6])
 '1  2  3'
 '4  5  6'
```

Matrices Extraction

- `diag` Diagonal of a matrix
- `triu` Extract upper triangular part
- `tril` Extract lower triangular part

```
c =

     1     2     3
     4     5     6
     7     8     9

>> disp(diag(c))
     1
     5
     9

>> disp(triu(c))
     1     2     3
     0     5     6
     0     0     9

>> disp(tril(c))
     1     0     0
     4     5     0
     7     8     9
```

Matrices

- `size` Size of array
- `length` Length of vector
- `length(x) = max(size(x))`

```
b =

     1     2     3
     4     5     6

>> length(b)

ans =

     3

>> b'

ans =

     1     4
     2     5
     3     6

>> length(b')

ans =

     3
```

Matrices: Members' Functions

- `sqrt`

```
sqrt([1:4])
1.0000    1.4142    1.7321    2.0000
sqrt([-1:1])
0.0000 + 1.0000i   0.0000 + 0.0000i   1.0000 + 0.0000i
```

- `factorial`

```
factorial([1:3])
1     2     6
```

Linear Equations

$$
\begin{aligned}
3.x + 2.y - z &= 10 \\
-x + 3.y + 2.z &= 5 \\
x - y - z &= -1
\end{aligned}
$$

$$
\begin{bmatrix} 3 & 2 & -1 \\ -1 & 3 & 2 \\ 1 & -1 & -1 \end{bmatrix}
\begin{bmatrix} x \\ y \\ z \end{bmatrix} =
\begin{bmatrix} 10 \\ 5 \\ -1 \end{bmatrix}
$$

$$
\begin{aligned}
A.X &= B \\
A^{-1}.A.X &= A^{-1}.B \\
X &= A^{-1}.B = A \setminus B
\end{aligned}
$$

```
>> A=[3 2 -1;-1 3 2;1 -1 -1]; B = [10 5 -1]';
>> A\B

ans =

   -2.0000
    5.0000
   -6.0000
```

Symbolic Math Toolbox

$$\begin{aligned} 3.x + 2.y - z &= 10 \\ -x + 3.y + 2.z &= 5 \\ x - y - z &= -1 \end{aligned}$$

`syms` Short-cut for constructing symbolic variables (not numeric)

```
syms x y z
eq1 = 3*x +2*y - z == 10;
eq2 = -x +3*y +2*z == 5;
eq3 = x - y - z    == -1;
solve([eq1, eq2, eq3], [x, y, z])
```

```
ans =

  struct with fields:

    x: -2
    y: 5
    z: -6
```

Functions

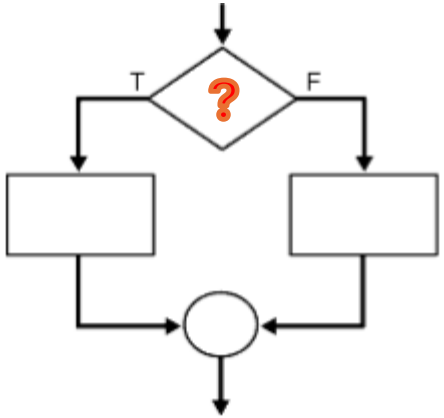
- Set of steps

Input ➡ Process ➡ Output

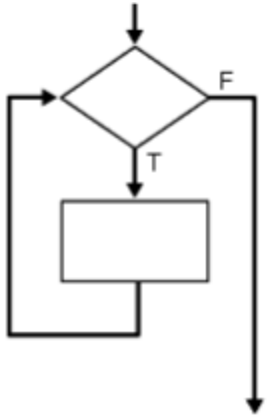
Control Structures

- Sequential

- Branching



- Repetition



Example

- Quadratic (second order) equations:

$$a \,.\, x^2 + b \,.\, x + c = 0$$

- Inputs:

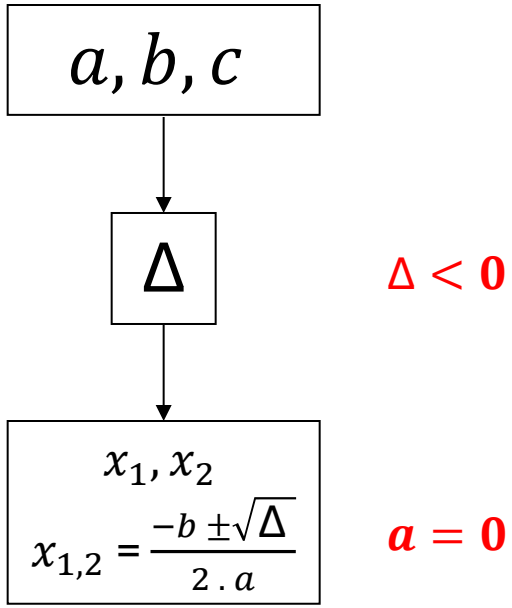
$$a, b, c$$

- Outputs:

$$x_1, x_2$$

Second Order Equation

$$a \,.x^2 + b \,.x + c = 0$$

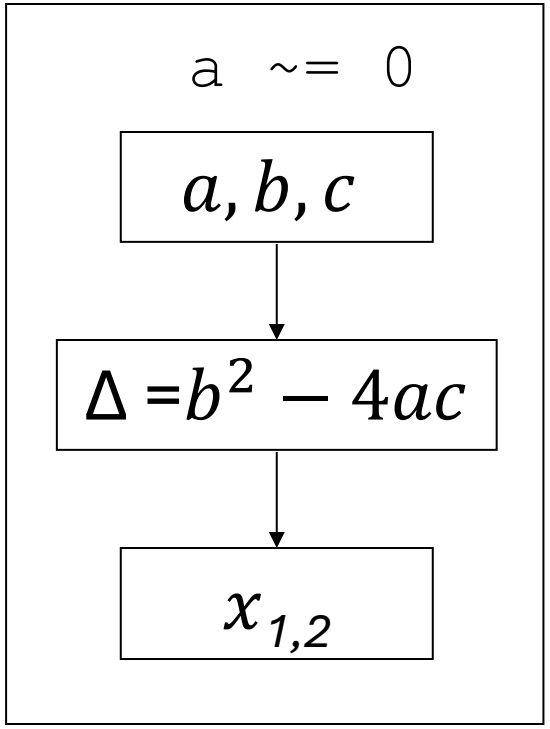


$\Delta < 0$

$a = 0$

Decisions

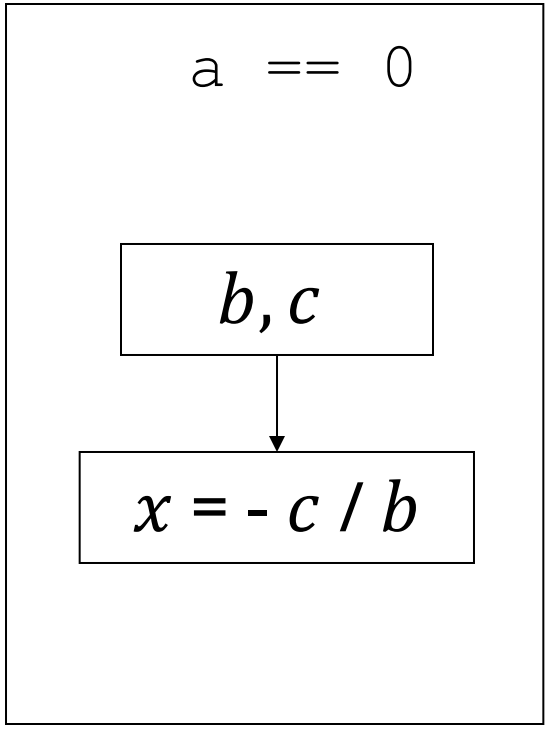
$$a\,.x^2 + b\,.x + c = 0$$



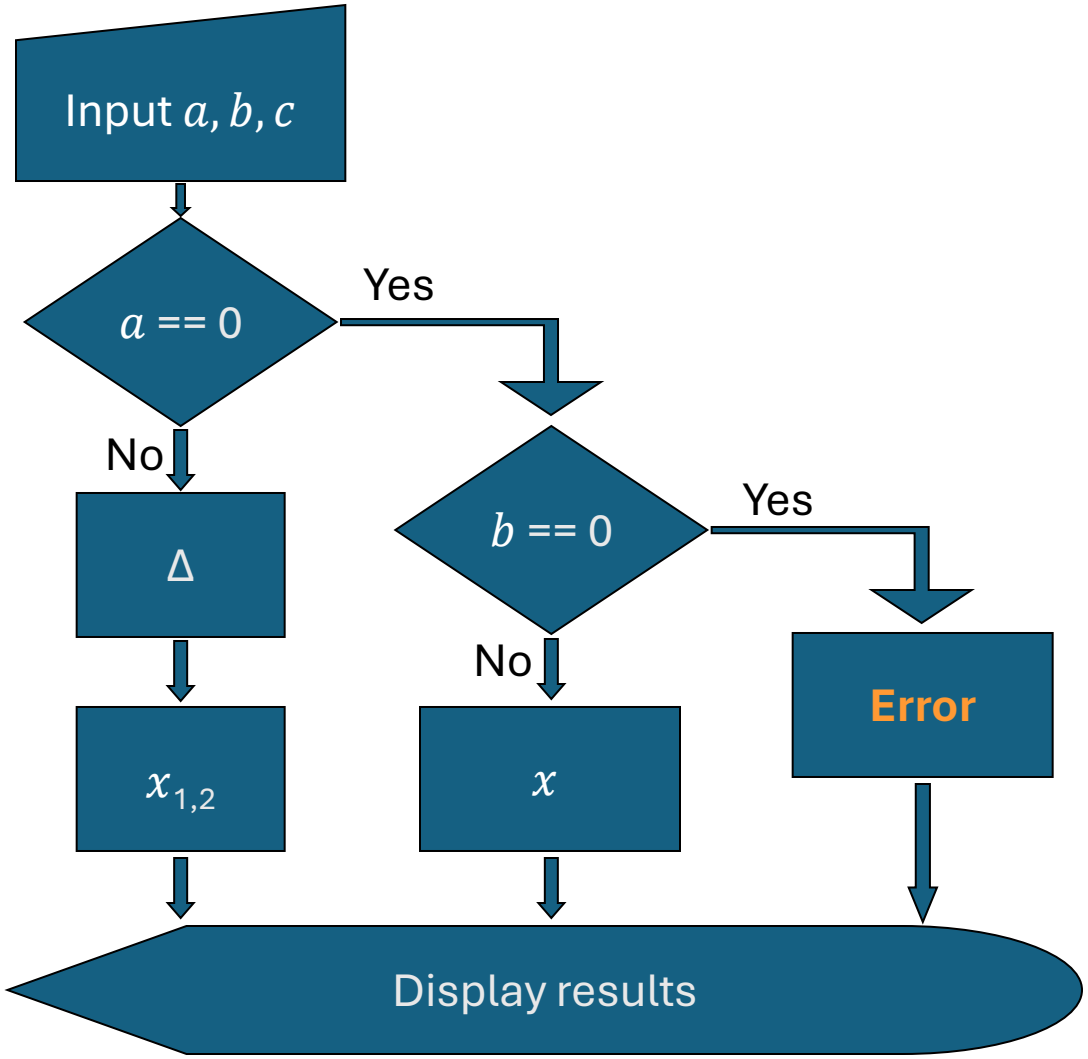
a == 0

b == 0

Check input!



Flowchart


Input a, b, c
$a == 0$
Yes
No
Δ
$b == 0$
Yes
No
Error
$x_{1,2}$
x
Display results

Relational Operators

• Equal	==	(not equal)
• Unequal	~=	(VB <>)
• Smaller	<	
• Greater or equal	>=	
• And	&	
• Or	|	
• Not	~	

Relational Operators

- `false  =  0`
- `true  =  1`
- Any none zero value is true

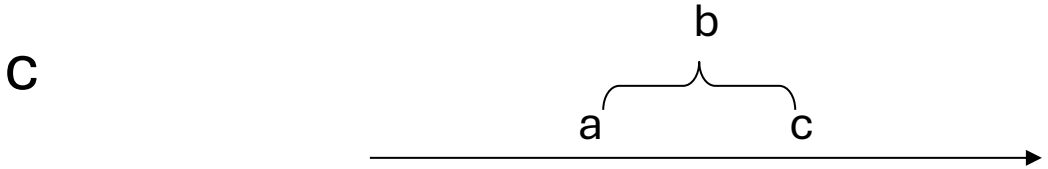
Relational Operators

- Not: ~

```
~False = True;  ~0  = 1
~True  = False;
~1  = 0;
~2  = 0;
```

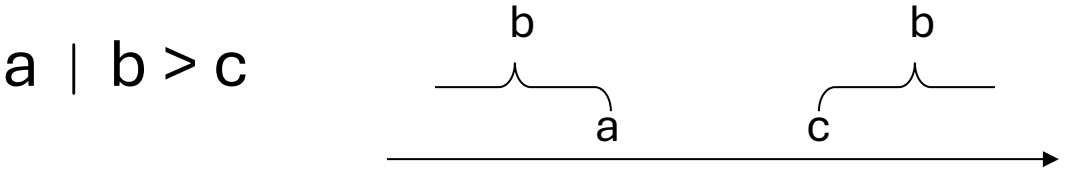
Relational Operators

- And: b > a & b < c




```
True   &  True      = True
False  &  False     = False
True   &  False     = False
False  &  True      = False
```

- Or: b < a | b > c




```
True   |  True      =  True
False  |  False     =  False
True   |  False     =  True
False  |  True      =  True
```

Precedence of Operators

1.	Brackets	( )
2.	Power	^ .^
3.	Unary Plus/Minus, Not	+ – ~
4.	Multiple and divide	* / \ .* ./ .\
5.	Addition and Subtraction	+ –
6.	Colon	:
7.	Compare	< > ~=
8.	And	&
9.	Or	|

Precedence of Operators

```
0 & (0 | 1)                    0

(0 & 0) | 1                    1

0 & 0 | 1                      1
```

Precedence of Operators

```
(~ 1) * 0                    0

~ (1 * 0)                    1

~ 1 * 0                      0
```

Decisions Syntax

```
if condition
    true statements
end
```

Decisions Syntax

```
if condition
   true statements
else
   false statements
end
```

Decisions Syntax

```
if condition1
  true1
elseif conditio2
  true2
else
  all conditions false
end
```

Second Order Equation Solver

```
function [x1, x2] = root_ex(a, b, c)
% root_ex(a, b, c) computes the roots of the
% second order equation a x^2 + b x + c = 0
```

Second Order Equation

```
if (a~=0)
    d  = b^2-4*a*c;
    x1 = (-b+sqrt(d))/(2*a);
    x2 = (-b-sqrt(d))/(2*a);
    disp('There are two roots for this equation.')
end
```

Second Order Equation

```
if (a~=0)
    d2 = sqrt(b^2-4*a*c);
    x1 = (-b+d2)/(2*a);
    x2 = (-b-d2)/(2*a);
    disp('There are two roots for this equation.')
end
```

Second Order Equation

```
if (a~=0)
    d2 = delta2(a, b, c);
    x1 = (-b+d2)/(2*a);
    x2 = (-b-d2)/(2*a);
    disp('There are two roots for this equation.')
end

function d2 = delta2(a, b, c)
% Sub function to calculate delta square root
d2 = sqrt(b^2-4*a*c);
```

Second Order Equation

```
if (a==0)




end
```

Second Order Equation

```
if (a==0)
    if  (b==0)



    else



    end
end
```

Second Order Equation

```
if (a==0)
    if  (b==0)
        x1 = NaN;
        x2 = NaN;
        disp('This equation is degenerate!')
    else
        x1 = -c/b;
        x2 = x1;
        disp('There is a single root for this equation.')
    end
end
```

```
function [x1, x2] = root_ex(a, b, c)
% root_ex(a, b, c) computes the roots of the
% second order equation a x^2 + b x + c = 0
if (a==0)
    if  (b==0)
        x1 = NaN;
        x2 = NaN;
        disp('This equation is degenerate!')
    else
        x1 = -c/b;
        x2 = x1;
        disp('There is a single root for this equation.')
    end
else
    d2 = delta2(a, b, c);
    x1 = (-b+d2)/(2*a);
    x2 = (-b-d2)/(2*a);
    disp('There are two roots for this equation.')
end
function d2 = delta2(a, b, c)
% Sub function to calculate delta square root
d2 = sqrt(b^2-4*a*c);
```

Roots Solver

- `[x1, x2] = root_ex(1, 0, -4)`

```
>> [x1, x2] = root_ex(1, 0, -4)
There are two roots for this equation.

x1 =

     2


x2 =

    -2
```

- MATLAB built in solver
  - `roots([1 0 -4])`
  - `roots(1:5)`

Find polynomial roots

```
>> roots(1:5)

ans =

  -1.2878 + 0.8579i
  -1.2878 - 0.8579i
   0.2878 + 1.4161i
   0.2878 - 1.4161i
```

Roots Solver

- Symbolic Math Toolbox

```
syms x
solve(x^2 -4 == 0, x)
```

```
>> syms x
solve(x^2 -4 == 0, x)

ans =

-2
 2
```

Example

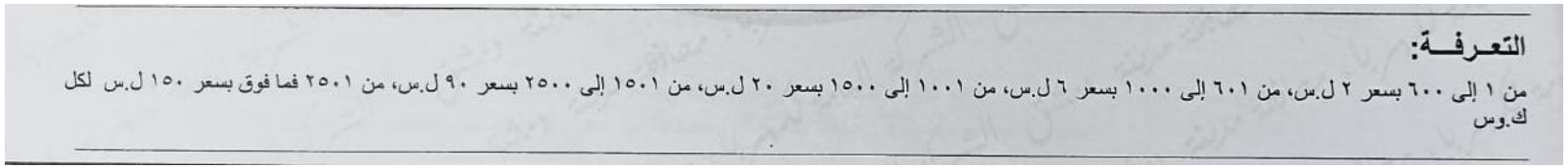
- حساب كلفة فاتورة الكهرباء
- *اختيار التعرفة الأفضل حسب الاستهلاك*
  - *سعر ثابت 25 ليرة للكيلو الواط الساعي*
  - تعرفة متغيرة حسب الاستهلاك:

التعـرفـــة:

من ١ إلى ٦٠٠ بسعر ٢ ل.س، من ٦٠١ إلى ١٠٠٠ بسعر ٦ ل.س، من ١٠٠١ إلى ١٥٠٠ بسعر ٢٠ ل.س، من ١٥٠١ إلى ٢٥٠٠ بسعر ٩٠ ل.س، من ٢٥٠١ فما فوق بسعر ١٥٠ ل.س لكل ك.وس

من	الى	سعر ك.و.س
0	600	2
601	1000	6
1001	1500	20
1501	2500	90
2501	-	150

Example



التعـرفـــة:

من ١ إلى ٦٠٠ بسعر ٢ ل.س، من ٦٠١ إلى ١٠٠٠ بسعر ٦ ل.س، من ١٠٠١ إلى ١٥٠٠ بسعر ٢٠ ل.س، من ١٥٠١ إلى ٢٥٠٠ بسعر ٩٠ ل.س، من ٢٥٠١ فما فوق بسعر ١٥٠ ل.س لكل ك.وس

```
function tariff = ebill_ex(kwh)
% Calculate electricity bill
if kwh < 1
                    tariff = 0;
elseif kwh <= 600
                    tariff = kwh * 2;
elseif kwh <= 1000
                    tariff = 600 * 2 + (kwh - 600) * 6 ;
elseif kwh <= 1500
                    tariff = 600 * 2 + 400 * 6 + (kwh - 1000) * 20;
elseif kwh <= 2500
                    tariff = 600 * 2 + 400 * 6 + 500 * 20 + (kwh - 1500) * 90;
else
                    tariff = 600 * 2 + 400 * 6 + 500 * 20 + 1000 * 90 + (kwh - 2500) * 150;
end
```

Example

```
function tariff = ebill_ex(kwh)

if kwh < 1
elseif kwh <= 600  & kwh >= 1
elseif kwh <= 1000 & kwh >= 600
elseif kwh <= 1500 & kwh >= 1000
elseif kwh <= 2500 & kwh >= 1500
else
end
```

```
if kwh <= 2500
elseif kwh <= 1500
elseif kwh <= 1000
elseif kwh <= 600
elseif kwh <= 1
else
end
```

```
if kwh > 2500
elseif kwh > 1500
elseif kwh > 1000
elseif kwh > 600
elseif kwh > 1
else
end
```

Bottle Problem

- Half full / half empty?
- Measure volume

Loops

- Repeated statements
- Use condition (`while`) or counter (`for`)

```
counter = 1;
while counter <= 3
    disp(counter)
    counter = counter + 1;
end
```

```
for counter = 1:3
    disp(counter)
end
```

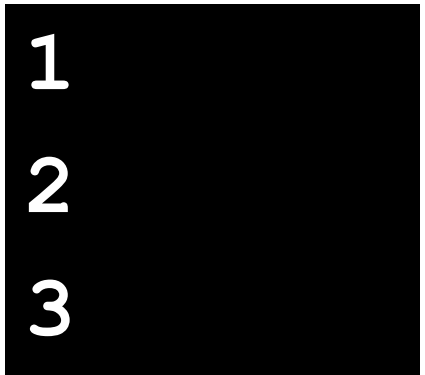
```
1
2
3
```

```
2
3
4
```

Loops

- Repeated statements
- Use condition (`while`) or counter (`for`)

```
counter = 1;
while counter <= 3
    disp(counter)
    counter = counter + 1;
end
```




```
counter = 1;
while counter <= 3
    counter = counter + 1;
    disp(counter)
end
```

Endless Loops!

```
counter = 1;
while counter <= 3
   disp(counter)
end
```

```
1
1
1
1
...
```

```
for counter = 1:3
  disp(counter)
  counter = 1;
end
```

```
1
2
3
```

Endless Loops!

```
counter = 1;
while true
    disp(counter)
    counter = counter + 1;
    if counter > 3
        break
    end
end
```

```
1
2
3
```