INTRODUCTORY COMPUTER SCIENCES

DR. MEHMET GÜR

MAİL TO : gur@yildiz.edu.tr

AVESİS: http://www.avesis.yildiz.edu.tr/gur

Title	Code	Local Credit	ECTS	Lecture (hour/week)	Practical (hour/week)	Laboratory (hour/week)
Introductory Computer Sciences	HRT1172	3	4	2	2	0

Course Objectives	-Teaching Programming Language Concepts -Teaching Problem Analysis using algorithmic approach and to teach coding with a programming language

Course Learning Outcomes

- Students Will Be Able To Define Fundamental Concepts Of Programming.
- Students Will Be Able To Compile A Programming Language Program.
- Students Will Be Able To Use Arrays And Matrices.
- Students Will Be Able To Write Functions And M-file.
- Students Will Be Able To Use If Then Else And Switch Case.
- Students Will Be Able To Use Loops (For, Do While).
- Students Will Be Able To Use Graphics.

References:

- BUSCH R, (1985), BASİC FÜR EİNSTEİGER
- UZUNOĞLU VD. (2002), MATLAB 6.0-6.5, TÜRKMEN KİTABEVİ, İSTANBUL
- WWW.MATHWORKS.COM
- THE MATHWORKS INC., (2003), STATİSTİC TOOLBOX FOR USE İN MATLAB'S USER'S GUİDE
- DOĞAN, U, (2006), TEMEL BİLGİSAYAR BİLİMLERİ, YTÜ DERS NOTLARI, İSTANBUL
- CHAPMAN, STEPHEN J. (2008), MATLAB PROGRAMMİNG FOR ENGİNEERS

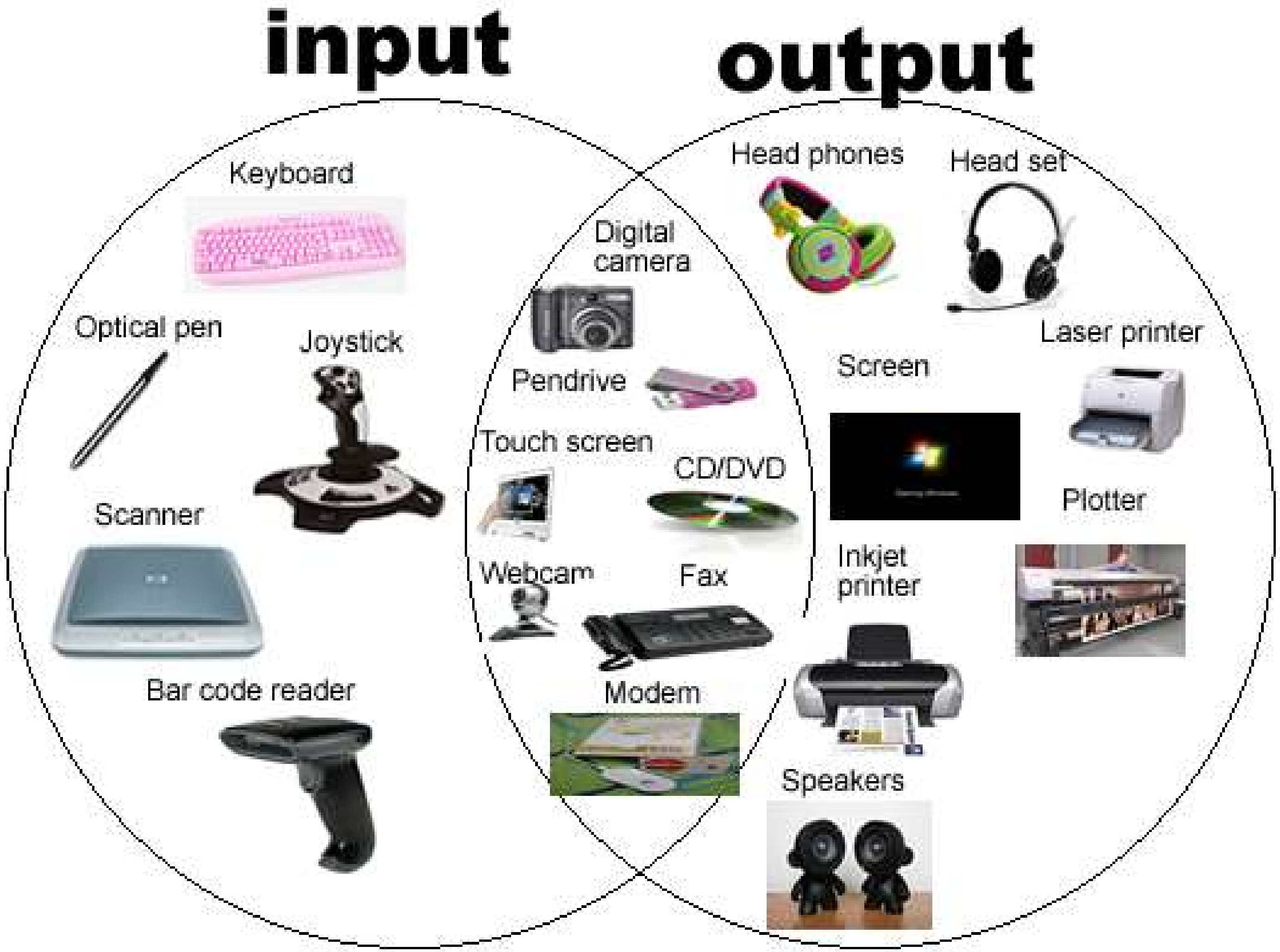
Content

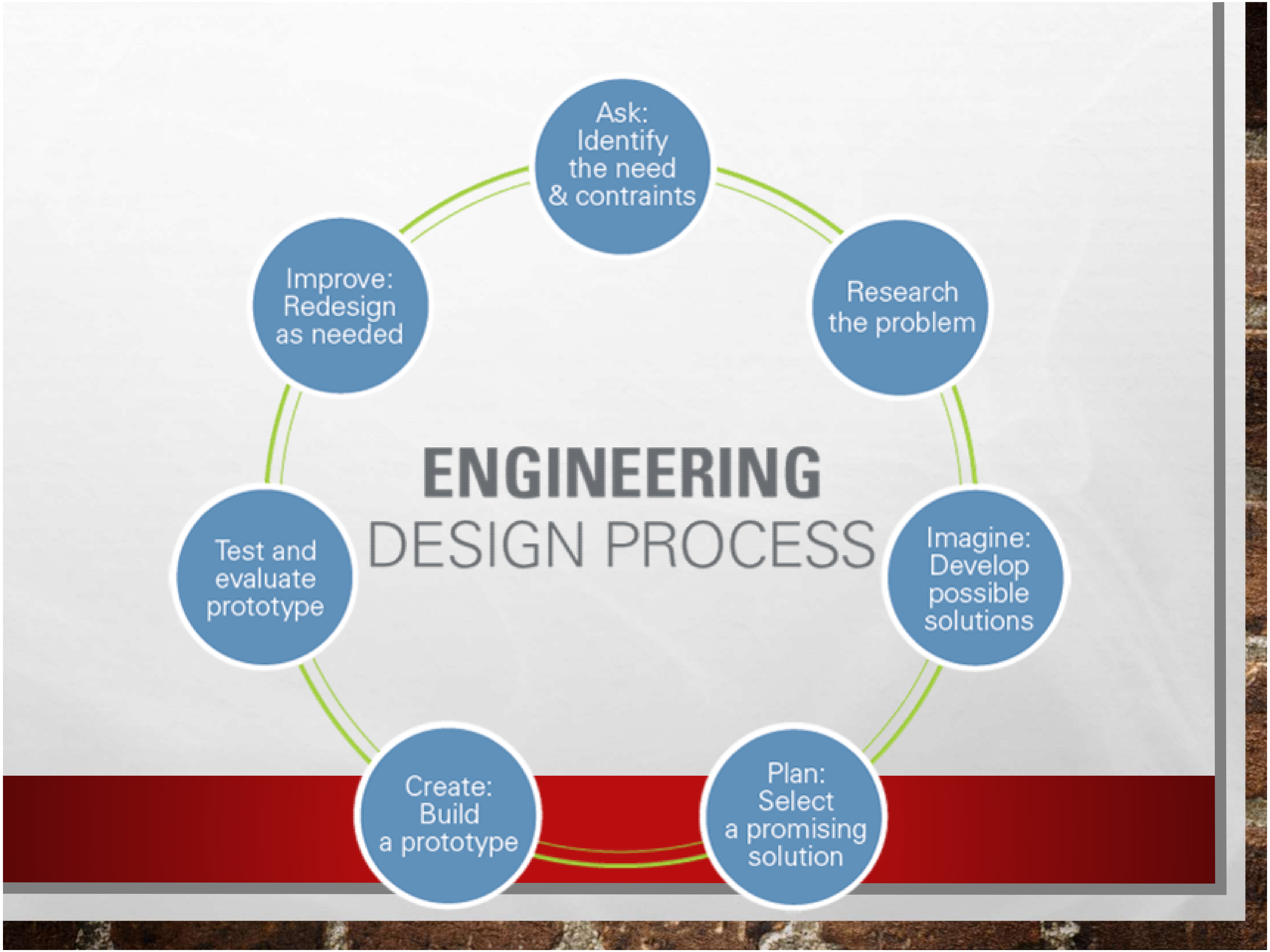
- Introduction
- Programming Languages
- Problem-solving Process

1. INTRODUCTION to Computer Sciences

- A computer is an electronic device that manipulates information, or data. It has the ability to store, retrieve, and process data.
- A programming language is a formal computer language designed to communicate instructions to a machine, particularly a computer. Programming languages can be used to create programs to control the behavior of a machine or to express algorithms.

Input & Output Devices




ENGINEERING
DESIGN PROCESS
Ask: Identify the need & contraints
Research the problem
Imagine: Develop possible solutions
Plan: Select a promising solution
Create: Build a prototype
Test and evaluate prototype
Improve: Redesign as needed

PROBLEM-SOLVING PROCESS

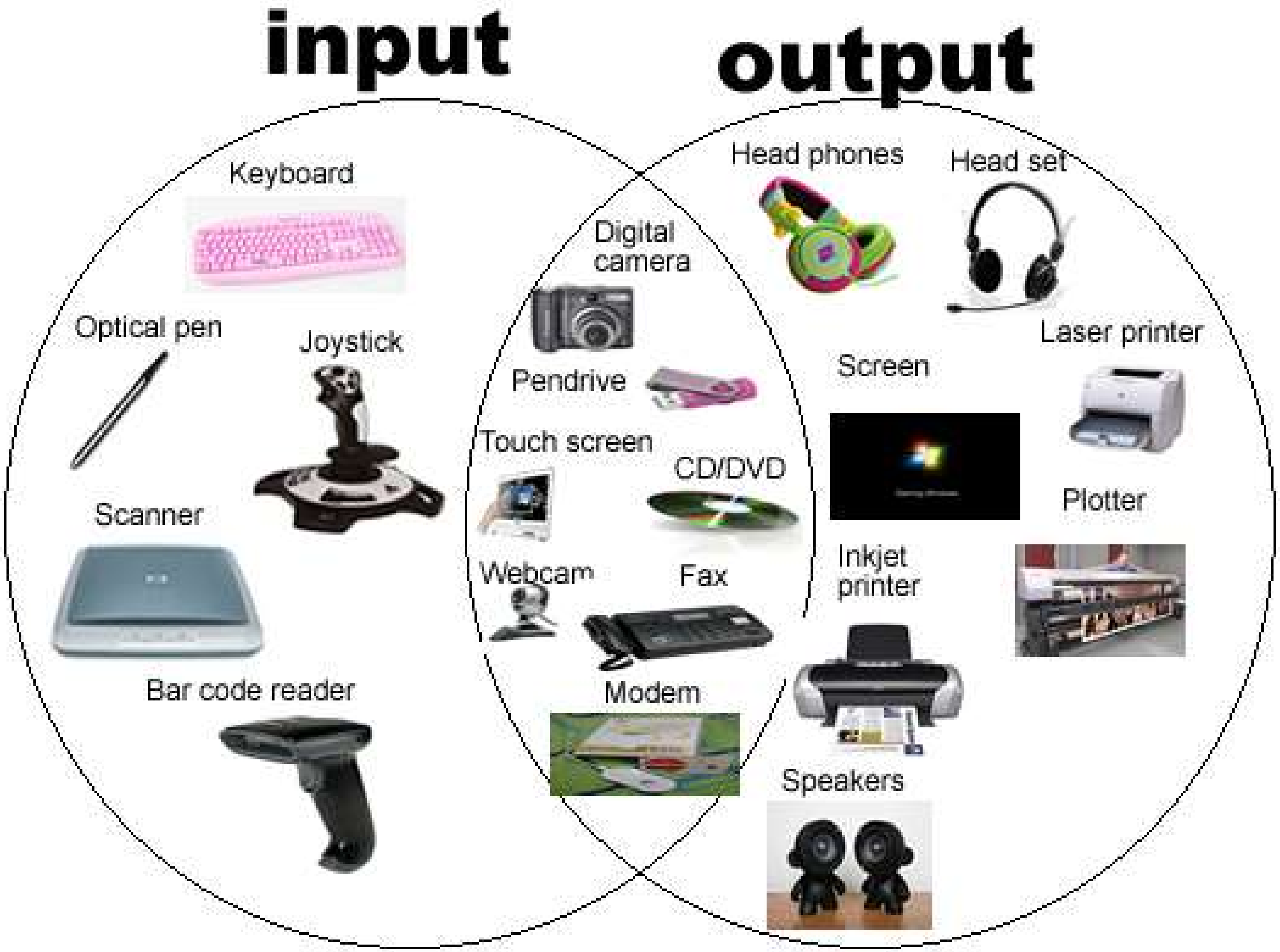
The Problem-solving Process For A Computational Problem Can Be Outlined As Follows:

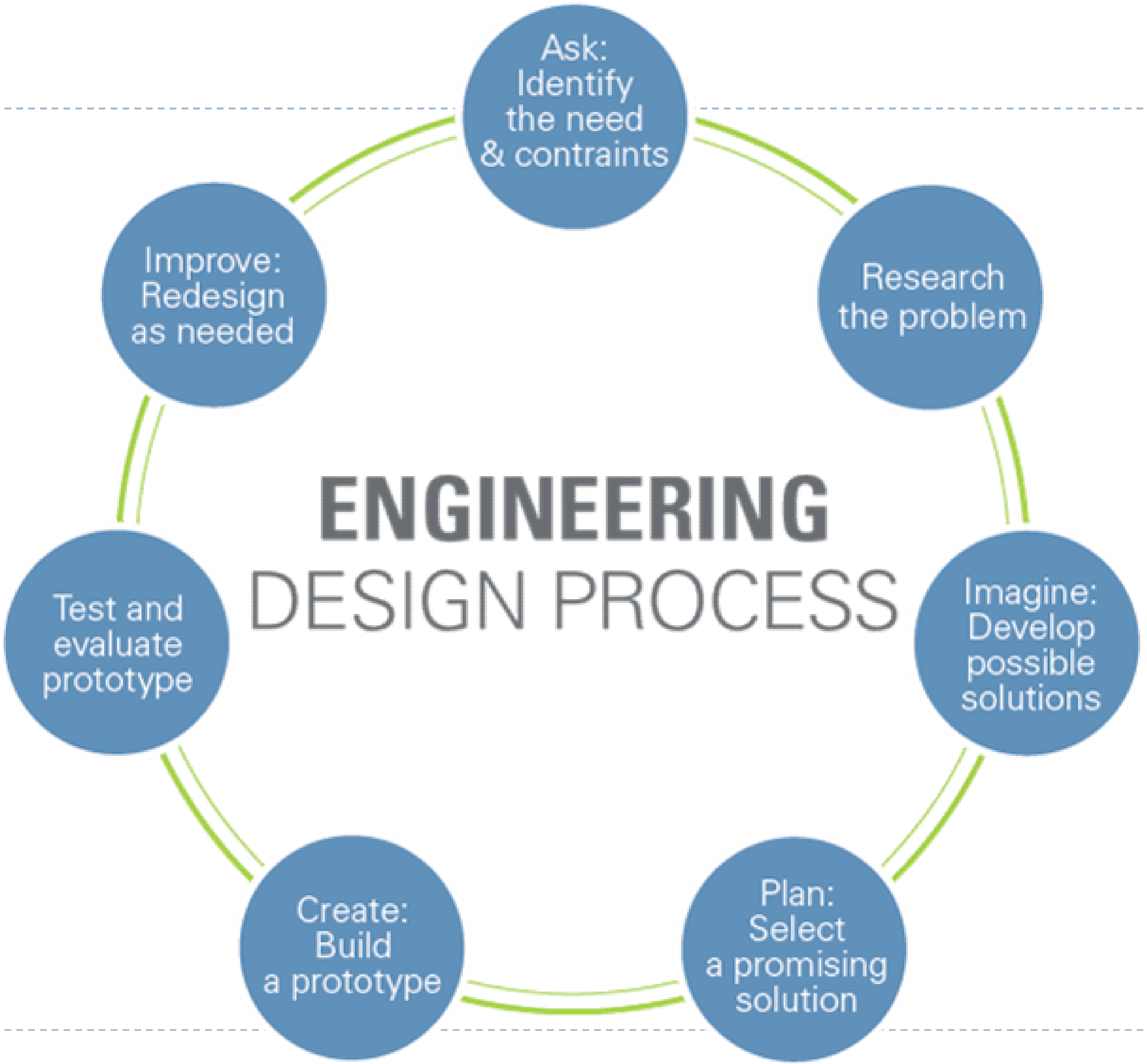
1. **Define The Problem.**
2. **Create A Mathematical Model.**
3. **Develop A Computational Method For Solving The Problem.**
4. **Implement The Computational Method.**
5. **Test And Assess The Solution.**

1. INTRODUCTION to Computer Sciences

- A **computer** is an electronic device that manipulates information, or data. It has the ability to **store**, **retrieve**, and **process** data.
- A **programming language** is a formal computer language designed to communicate instructions to a machine, particularly a computer. Programming languages can be used to create programs to control the behavior of a machine or to express algorithms.

Input &Output Devices




Ask:
Identify
the need
& contraints
Research
the problem
Imagine:
Develop
possible
solutions
Plan:
Select
a promising
solution
Create:
Build
a prototype
Test and
evaluate
prototype
Improve:
Redesign
as needed
ENGINEERING
DESIGN PROCESS

PROBLEM-SOLVING PROCESS

The problem-solving process for a computational problem can be outlined as follows:

1. **Define the problem.**
2. Create a mathematical model.
3. Develop a computational method for solving the problem.
4. Implement the computational method.
5. Test and assess the solution.

2. PROGRAMMING LANGUAGES

A programming language is a notation for writing programs, which are specifications of a computation or algorithm.

Command “Yaz”			
Basic	Pascal	C / C++	MATLAB
Print	Writeln	Printf	fprintf

Command “Gir”			
Basic	Pascal	C / C++	MATLAB
input	Readln	Scanf	input, read

2. Programming Languages

- Three types of programming languages
  - Machine languages
    - Strings of numbers giving machine specific instructions
    - Example:

```
+1300042774 (these would really be in binary)
+1400593419
+1200274027
```

  - Assembly languages
    - English-like abbreviations representing elementary computer operations (translated via assemblers)
    - Example:

```
    LOAD    BASEPAY
    ADD     OVERPAY
    STORE   GROSSPAY
```

2. Programming Languages

- High-level languages
  - Instructions closer to everyday English
    - English is a natural language. Although high level programming languages are closer to natural languages, it is difficult to get too close due to the ambiguities in natural languages (a statement in English can mean different things to different people – obviously that is unacceptable for computer programming). However, this is a big research area of computer science.
  - Use mathematical notations (translated via compilers)
  - Example:

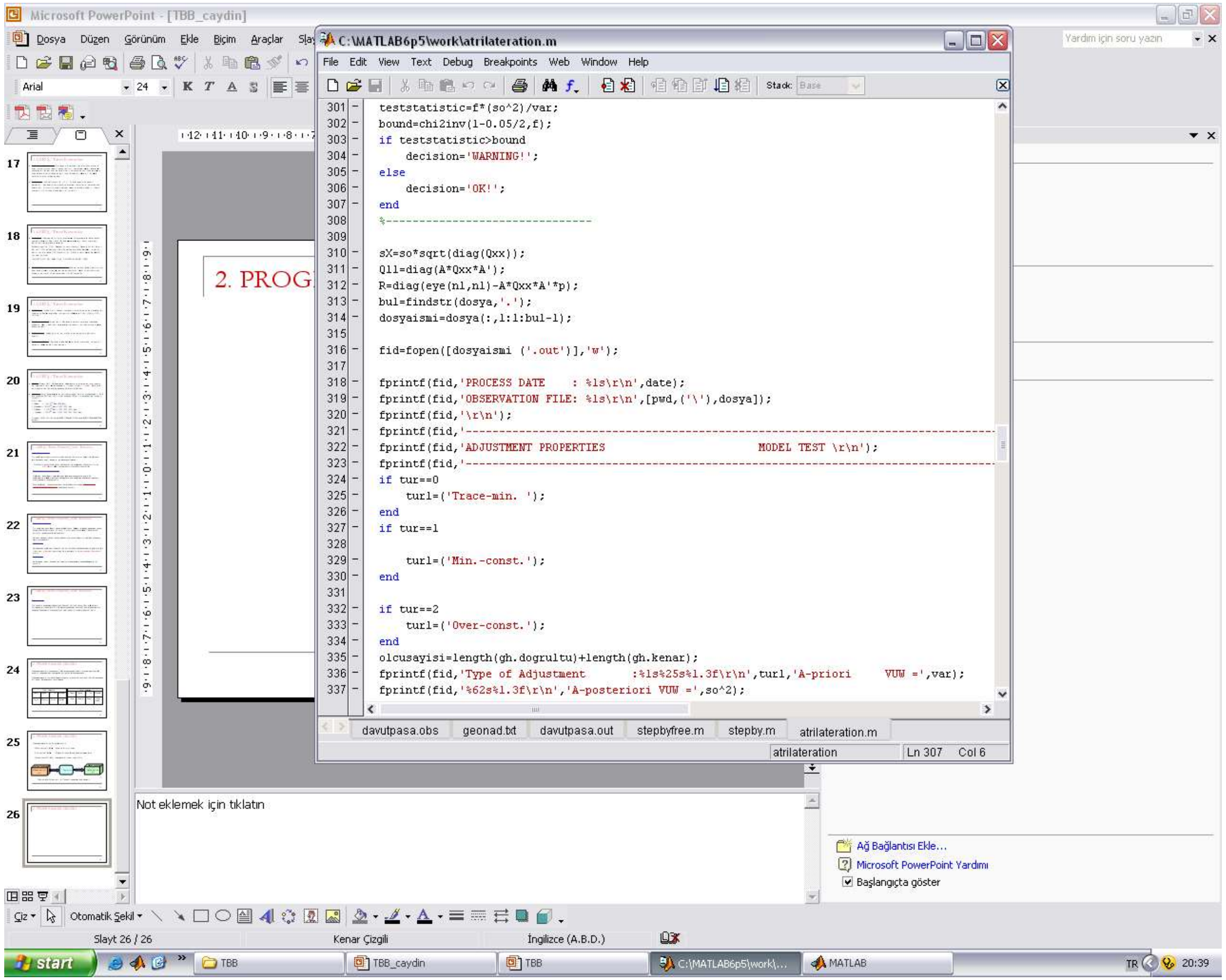
```
grossPay = basePay + overTimePay
```

  - Interpreter – Executes high level language programs without compilation.

High-level languages: For example: BASIC, Delphi, C, C++, COBOL, Fortran, Java, Lisp, Pascal, Flash etc.

2. Programming Languages

An Example of Coding



3. OPERATORS

The MATLAB operators fall into three categories:

1. **Arithmetic Operators:** ***perform numeric computations***
2. **Relational Operators:** ***compare operands quantitatively***
3. **Logical operators:** ***use the logical operators AND, OR***

3. OPERATORS / Arithmetic Operators

- Basic arithmetic operations (addition, subtraction, multiplication, division)

- Mathematical functions (exponential, logarithmic, trigonometric, etc.)

Decimal Digit	
Math (,)	Computer (.)
125,865	125.865

In front of the digits

Positive :	No sign
Negative :	–

Arithmetic Operators

Operator	Description
+	Addition
-	Subtraction
.*	Multiplication
./	Right division
.\	Left division
+	Unary plus

Arithmetic Operators

Operator	Description
-	Unary minus
:	Colon operator
.^	Power
.'	Transpose
'	Complex conjugate transpose
*	Matrix multiplication
/	Matrix right division
\	Matrix left division
^	Matrix power

3. OPERATORS / Arithmetic Operators

Operation	Arithmetic	MATLAB
Addition	a + b	a + b
Subtraction	a – b	a – b
Multiplication	a . b	a * b
Division	a ÷ b	a / b
Exponentiation (to the power of)	a^b	a^b
Modulo (Reminder)		%

Arithmetic Operators Priority

MATLAB's Operator Precedence Rules

1. **The contents of parenthesis are evaluated first starting with the innermost parenthesis**

2. **Exponentials are evaluated working from left to right**

3. **Multiplications and divisions are evaluated working from left to right**

4. **Additions and subtractions are evaluated working from left to right**

Priority	Operators	MATLAB
1	Parenthesis	((.....))
2	Exponential	a^b
3	Multiply and Divide	a*b and a / b
4	Add and Subtract	a+b and a-b

If there are operations with the same priority in a code, the operations are evaluated from left to right.

For example:

Y = A * B / C

In the above equation; there are operators that have the same priority (multiplication and division).
In this case, the code will be worked

first for A * B , and then the answer will be divided to C. (Left to Right)

Y = A ^ B ^ C

In the above equation, first the operation of A power to B is performed , and then the result is computed power of C.

Mathematical Expression	In coding
a + b – c + 2abc - 7	a + b – c + 2 * a *b *c-7
$a + b^2 - c^3$	a + b^2 – c^3
$\sqrt{a+b} - \frac{2ab}{b^2 - 4ac}$	Sqrt(a+b)-2*a*b/(b^2-4*a*c)
$A + \frac{B.C}{D} - E.F$	A+B*C/D-E*F

Example 1: For a = 4, b = 6, c = 8 and d = 10, investigate the results for given 3 equations encoded in programming language.

1. Equation	c * d / (a*d) + b + c *d / a	= 28
2. Equation	c * d / a*d + b + c *d / a	= 226
3. Equation	c * d / a*d + (b + c) *d / a	=235

Example 2: For A = 9, B = 16, solve the equations given below.

Equation	Mathematical Expression	
A + B^1/2	$\mathbf{A} + \frac{\mathbf{B}^1}{\mathbf{2}}$	= 17
A + B^(1/2)	$\mathbf{A} + \sqrt{\mathbf{B}}$	= 13
(A + B)^1/2	$\frac{(A+B)^1}{2}$	= 12.5
(A + B)^(1/2)	$\sqrt{A+B}$	= 5

3. OPERATORS / Relational Operators

Computer can produce decision models besides mathematical operations.

For example,

- **which one is bigger or smaller among two variables?**
- **are two variables equal or not?**

<u>**Comparison can be done for numerical values or strings.**</u>

There are 6 relational operators in MATLAB

Relational Operators

Operator	Description
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
~=	Not equal to

Example 3: Assume that **x=0 , y=sin(pi).**

In programming, When the below expression is entered

```
x==y
```

Then the result,

```
ans =

      0
```

Because; sin(pi)=1.224×10^{-16} and this is not equal to 0.

3. OPERATORS / Logical Operators

Logical operators are used in both relations and mathematical operations.

In programming, it is desired that the expressions should provide more than one conditions. In this case, logical operators are used.

Logical Operation	Command
AND	and (&)
OR	or (|)
NOT	not (~)

3. OPERATORS / Logical Operators

AND (&) Operator: If all conditions are true, the result will be true.
If all conditions should be ensured, the AND operator should be used between conditions.

A	B	A &B , and(A,B)
0	0	0
0	1	0
1	0	0
1	1	1

3. OPERATORS / Logical Operators

OR (|) Operator: If any condition is true, then the result will be true.

A	B	A|B, or(A,B)
0	0	0
0	1	1
1	0	1
1	1	1

Logical operators

```
>>> a, b, c = 10, 20, 30

>>> (a > b) and (b < c)
False

>>> (a < b) and (b < c)
True

>>> (a > b) or (b < c)
True
```

Operator	Description
a **and** b	Logical AND If both operands are True than it returns True
a **or** b	Logical OR If one of the operands is True then it returns True
not	Logical NOT

3. OPERATORS / Logical Operators

Precedence of Logical Operators:

1. **The contents of parenthesis are evaluated first starting with the innermost parenthesis**

2. **NOT (~) has a priority than the other logical operators.**

3. **AND (&) and OR (|) are working then.**

Example 3 :

In a company, a list of workers will be prepared and two conditions are necessary. First, the worker should be over 23 years old and his/her salary should be 600 TL.

$$\text{If} \quad \underbrace{\text{Age} > 23}_{\text{Condition 1}} \quad \& \quad \underbrace{\text{salary} == 600 \ \text{then}}_{\text{Condition 2}} \quad \text{Print Name of Worker}$$

Age	Salary	Condition 1	Condition 2	Result	Print
20	440	0	0	0	No
19	600	0	1	0	No
25	445	1	0	0	No
30	600	1	1	1	YES

Example 4 : Computation of Azimuth angle.
The equation is as below:

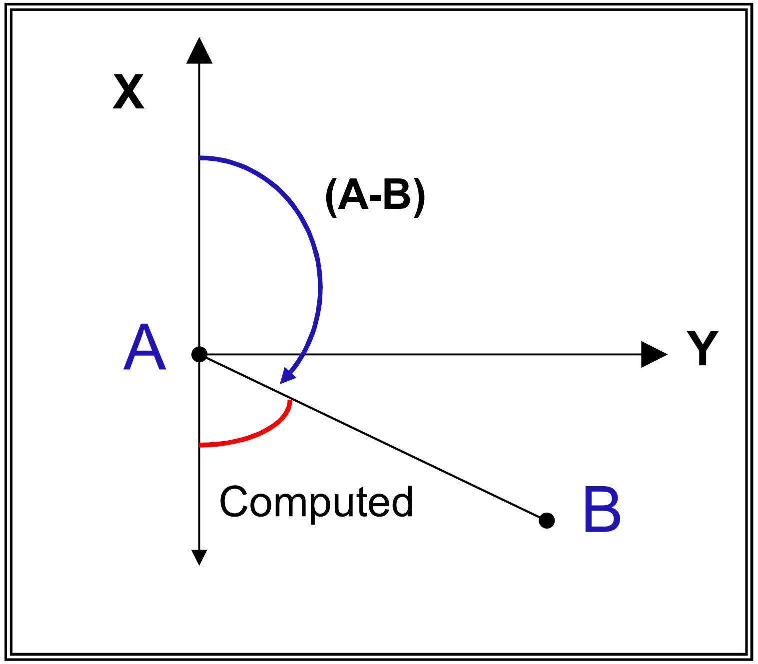
(A-B)=atan((YB-YA)/(XB-XA)).

2nd Quadrant,

dY=YB-YA; dX=XB-XA

If (dY>0) & (dX<0)

(A-B)=(A-B)*200/pi+200



(All languages work in RADIAN!)

3. OPERATORS / Logical Functions

Numbers, numeric array, characters (i.e., names), character array

`a=1000` **(Numeric array)**

`b='Yildiz'` **(Character array)**

For these kind of arrays, there are logical functions in MATLAB
For instance:

ischar(a) : Determine if item is a character array. Returns logical true (1) if A is a character array and logical false (0) otherwise.

isnumeric(a): Determine if input is numeric array. Returns true if A is a numeric array and false otherwise.

isempty(a) : Determine whether array is empty. returns logical 1 (true) if A is an empty array and logical 0 (false) otherwise. An empty array has at least one dimension of size zero, for example, 0-by-0 or 0-by-5.

Content

- **Algorithms**
- **Flow Charts/Diagrams**

4. ALGORITHM

- Step-by-step solution
- "top-down design"
- In Algorithm,

  1. Which data (input) will be used? From where?
  2. Which processes will be applied? How?
  3. What will the results (output) be?
  4. Where will the results be displayed and stored?

4. ALGORITHM/ OPERATOR

Arithmetic Operators

^	Exponential
*	Multiplication
/	Division
+	Addition
-	Subtraction
.	Decimal Digit

Logical Operators

‘	NOT
.	AND
+	OR

Relational Operators

==	Equal to
<>	Not Equal to
<	Less than
>	Greater than
>=	Greater or equal to
<=	Less or equal to

General Operators

=	Assign
( )	Parenthesis

4. ALGORITHM/ TERMS

1. Expression : ***defined by programmer, who encodes the program for naming:***

- variables
- constants
- paragraphs
- store areas
- specific info types
- subprograms etc.

The expression names in the program are more appropriately chosen to associate with the expressions they hold. For example, **“karekök”**

4. ALGORITHM/ VARIABLES

Rules for naming in Matlab;

- 26 Letters in English Alphabet between A and Z
- Numbers between 0 and 9.
- Start with letter
- Variable name can not start with number
- Variable name can not include only numbers

Command Window

```
>> x=2

x =

     2
```

Command Window

```
>> x2=5

x2 =

     5
```

Command Window

```
>> 2x=5
 2x=5
 |
Error: Unexpected MATLAB expression.
```

Command Window

```
>> 234=5
 234=5
    |
Error: The expression to the left of the equals sign is not a valid target for an assignment.

>>
```

4. ALGORITHM/ TERMS

2. Variables : **Every time you run the program you can get different values or information that can be assigned are expressed as variables.**

3. Transfer: **The transfer operator is used to represent the results for assigning values.**

variable = expression

Name of any variable

Transfer operator

Expression, which is not arithmetical, logical or numerical

2. Transfer

variable = expression

1. Process

Process Direction

4. ALGORITHM DESIGN

Example: A=3, B=4, C=5

1. Start
2. T = 0
3. Enter a number (A)
4. T = T + A do the operation
5. Enter another number (B)
6. T = T + B do the operation
7. Enter another number (C)
8. T = T + C
9. Print T
10. End

Results for Example

Operation Order	A	B	C	Initial T	New T
1	3	-	-	0	0 + 3 = 3
2	-	4	-	3	3 + 4 = 7
3	-	-	5	7	7 + 5 = 12
T = 12					

4. ALGORITHM/ TERMS

4. Counter: **In the program, some operations require to run for a certain times and to count them.**

counter = counter + 1

In the right side of this equation, 1 is added to the old value of the variable and the result is assigned again to the same variable. This kind of count process is called as counter.

$$\boxed{\text{Counter variable} = \text{Counter variable} \pm \text{increment}}$$

New value of counter

Old value of counter

Increasing or decreasing

4. ALGORITHM/ Terms

5. Conditional Statements (Conditions): Execute statements if condition is true. Depending on specific condition, the program can take different actions.
"IF condition" correspondences to "if" in programming language.

For example: Let review an algorithm in a case where variable A is equal to variable B, Then assign new value to A, which is C/2.

1. Start
2. A=99
3. B=(A+1)*A/100
4. C=50
5. If A==B A=C/2
6. Print A
7. End

A=C/2=25

Because the condition A=B is provided.

4. ALGORITHM/ Terms

5. Repetition: To achieve the repetition of calculation in a number of times in program, loop is used.

Example : Let review an algorithm, which achieves the addition of the odd numbers between 1 and 10.

1. Start
2. T = 0
3. J = 1
4. If J > 10 then go 8
5. T = T + J
6. J = J + 2
7. Go 4

loop

8. Print T
9. End

Running Loop

Old J	Old T	New T	New J
1	0	0 + 1= 1	3
3	1	1 + 3 = 4	5
5	4	4 + 5 = 9	7
7	9	9 + 7 = 16	9
9	16	16 + 9 = 25	11
11	-	-	-

4. ALGORITHM / Advantages

I. Makes easier to write the program

II. Reduce wrong coding possibility

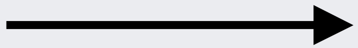
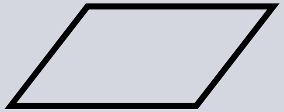
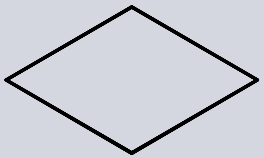
III. Reduces the task into a series of smaller steps of more manageable size.

IV. Problems can be approached as a series of small, solvable sub-problems.

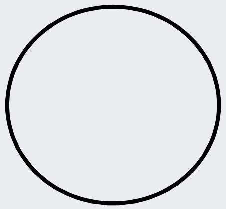
V. Efficient.

5. FLOWCHART

Flow chart –a graphic representation of the logical sequence of instructions.

NAME	OPERATION	SYMBOL	USE in FLOW CHART
Oval	START/END		Denotes the beginning or end of a program
Flow line	FLOW LINES		Denotes the direction of logic flow in a program
Parallelogram	INPUT/OUTPUT		Denotes either an input operation (e.g., INPUT) or an output operation (e.g., PRINT)
Rectangle	PROCESSING	 C = (a^2 + b^2)^ 1/2	Denotes a process to be carried out/ action (e.g., an addition)
Diamond	DECISION MAKING/ CHECKING		Denotes a decision (or branch) to be made.The program should continue along one of two routes (e.g. IF/ELSE)

5. FLOWCHART

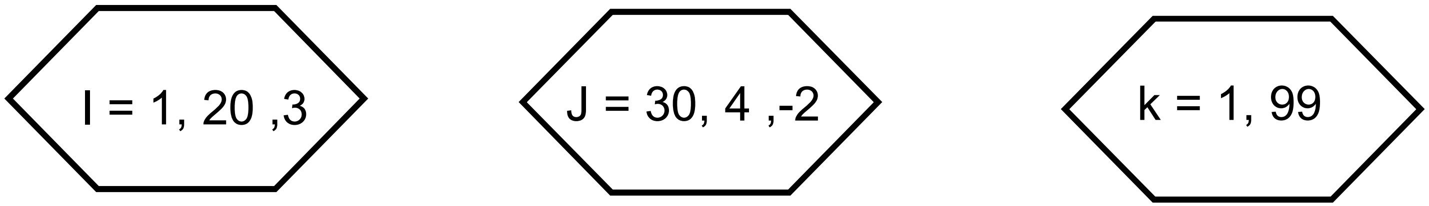
NAME	OPERATION	SYMBOL	USE in FLOW CHART
	LOOPING		Denotes looping which is represented based on condition or value of a variable
Circle	CONNECTION		Denotes the continuing of flowchart in another place of page

5. FLOW CHART

LOOPING

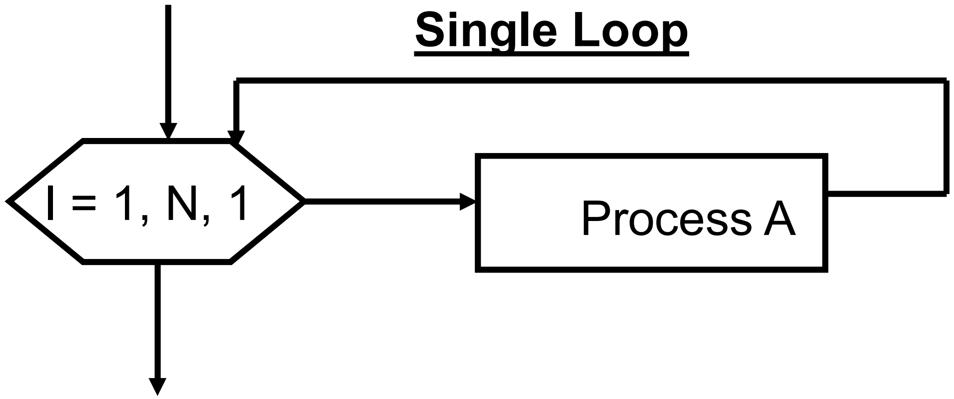
Control Variable= start value, end value, increment

The increment can be any positive or negative number

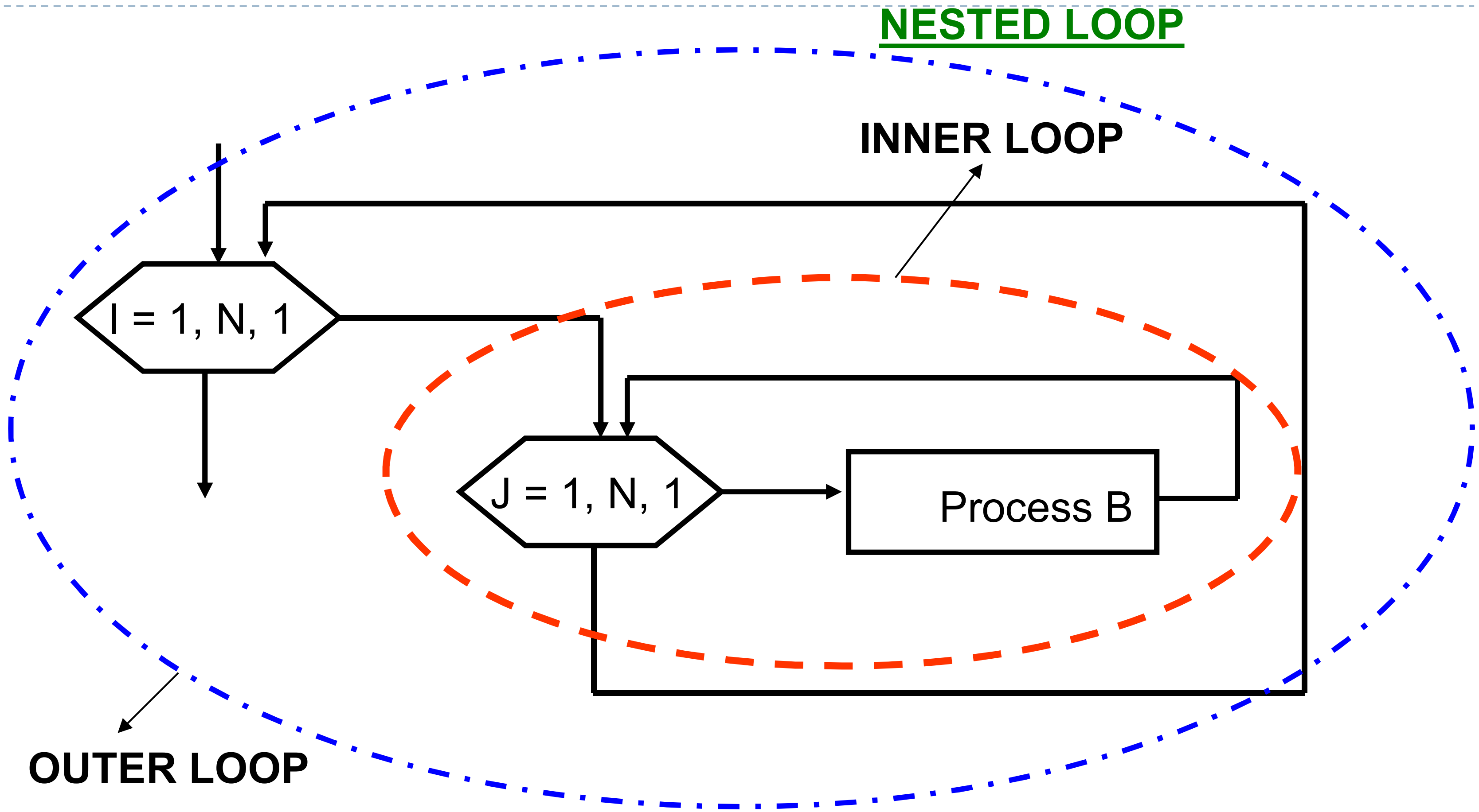


Loop for increasing **Loop for decreasing** **Loop for increasing with 1 step size**

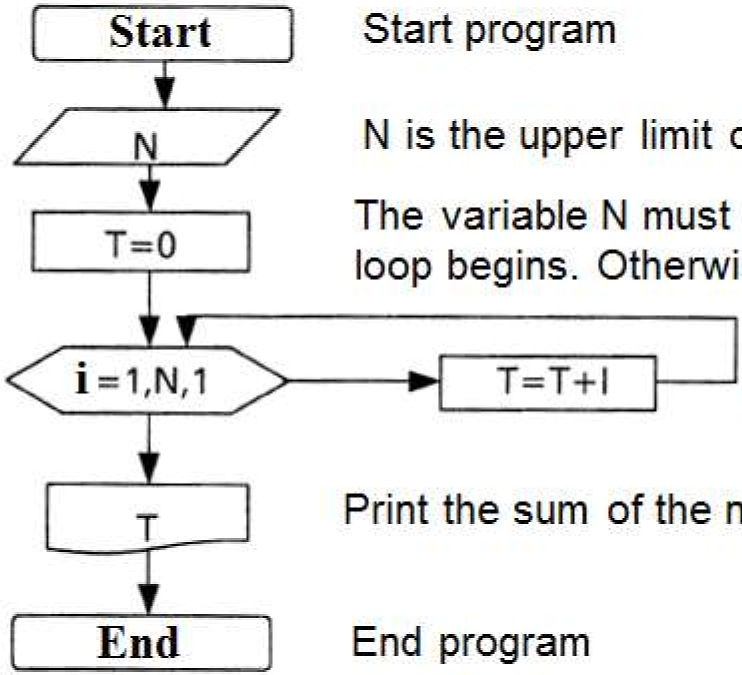
Single Loop



5. FLOWCHART



Example: Draw the flow chart for computing the sum of the first N integers



Start program

N is the upper limit of the numbers to be added

The variable N must exist, and have a meaningful value before the loop begins. Otherwise the expression N + i cannot be evaluated.

The successive numbers computed from loop are added to the 'T' to calculate the new value of T.

Print the sum of the numbers 'T'

End program

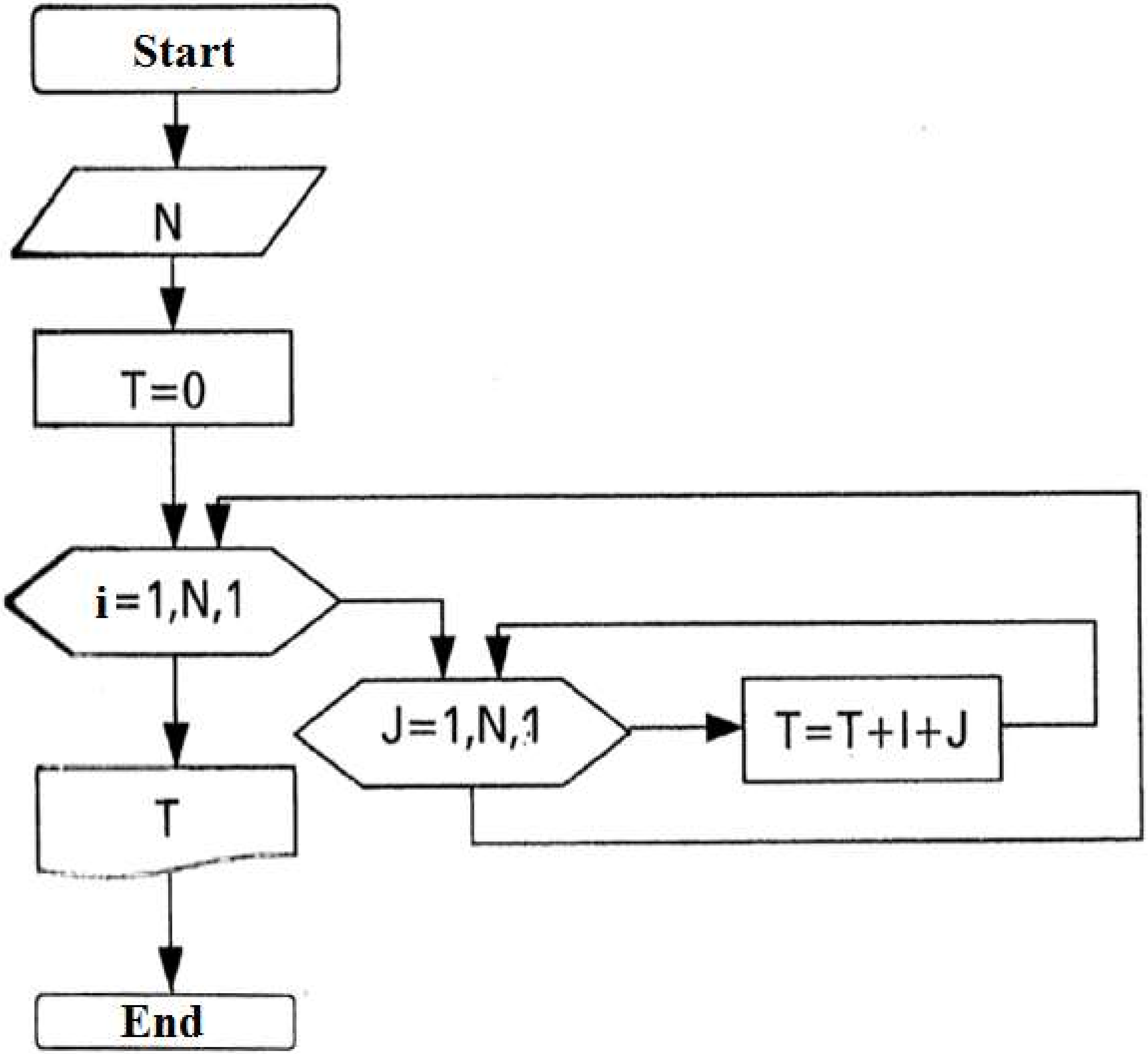
If N = 5; Results =

i	Old T	New T
1	0	0+1=1
2	1	1+2=3
3	3	3+3=6
4	6	6+4=10
5	10	10+5=15

- The statement T = 0 is called an initialization of T because it gives T its initial value before the loop starts.

5. FLOWCHART

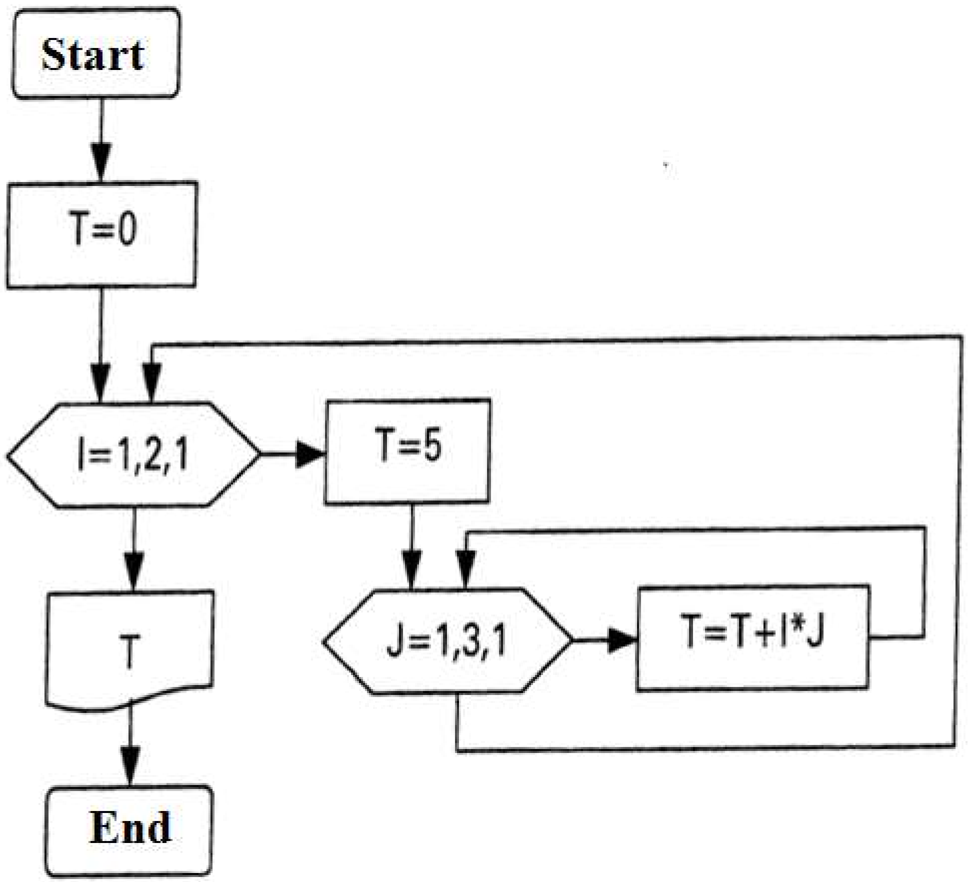
Example: **Compute the results of below flow chart for N=3**



I	J	Old T	New T
1	1	0	0+1+1=2
1	2	2	2+1+2=5
1	3	5	5+1+3=9
2	1	9	9+2+1=12
2	2	12	12+2+2=16
2	3	16	16+2+3=21
3	1	21	21+3+1=25
3	2	25	25+3+2=30
3	3	30	30+3+3=36

5. FLOWCHART

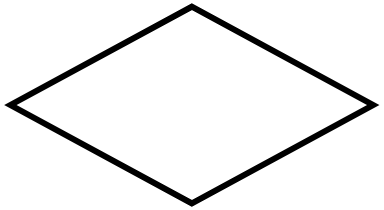
Example: Compute the solution of below flowchart.



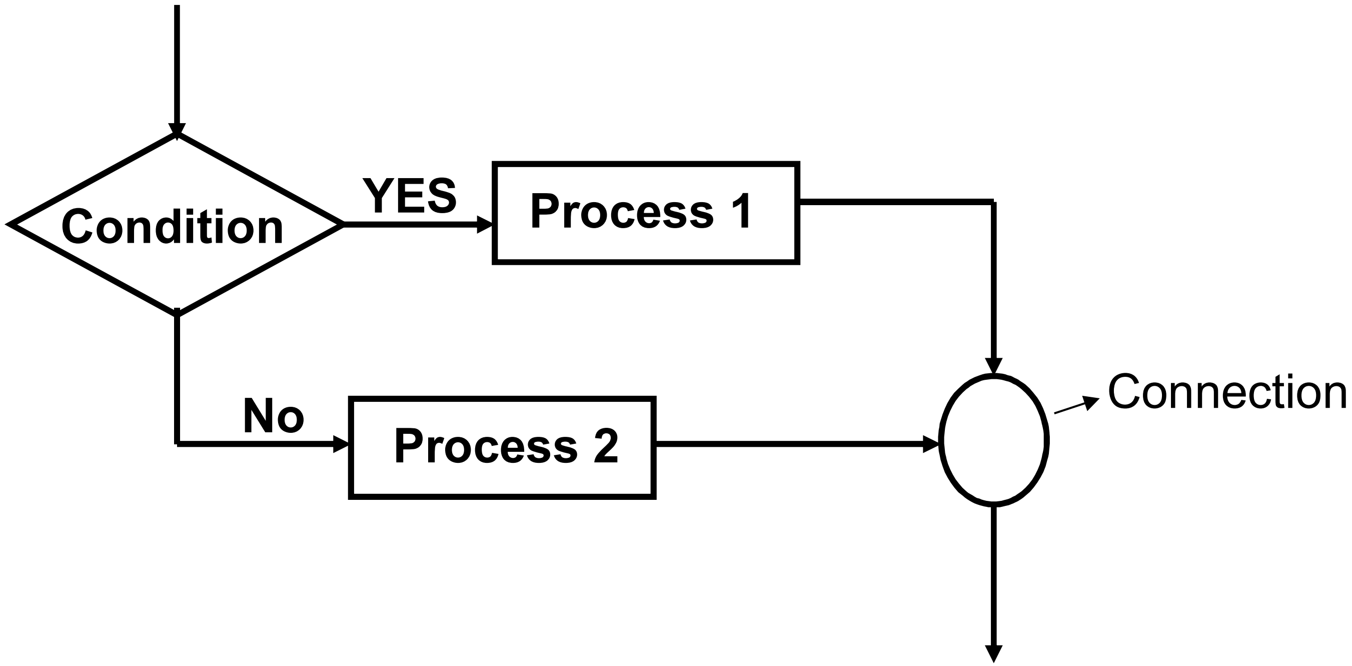
I	J	Old T	New T
1	1	**5**	5+1*1=6
1	2	6	6+1*2=8
1	3	8	8+1*3=11
2	1	**5**	5+2*1=7
2	2	7	7+2*2=11
2	3	11	11+2*3=**17**

5. FLOWCHART

V. Decision Making

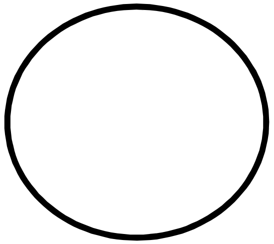


It allows to give a decision in an algorithm and provides a process depending upon this decision.

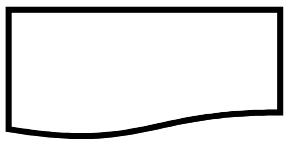


VI. Connection

Denotes the continuing of flowchart in another place of page

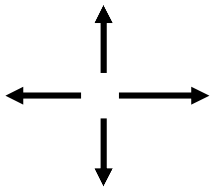


VII. Print/Output



Enable to print results/information to screen

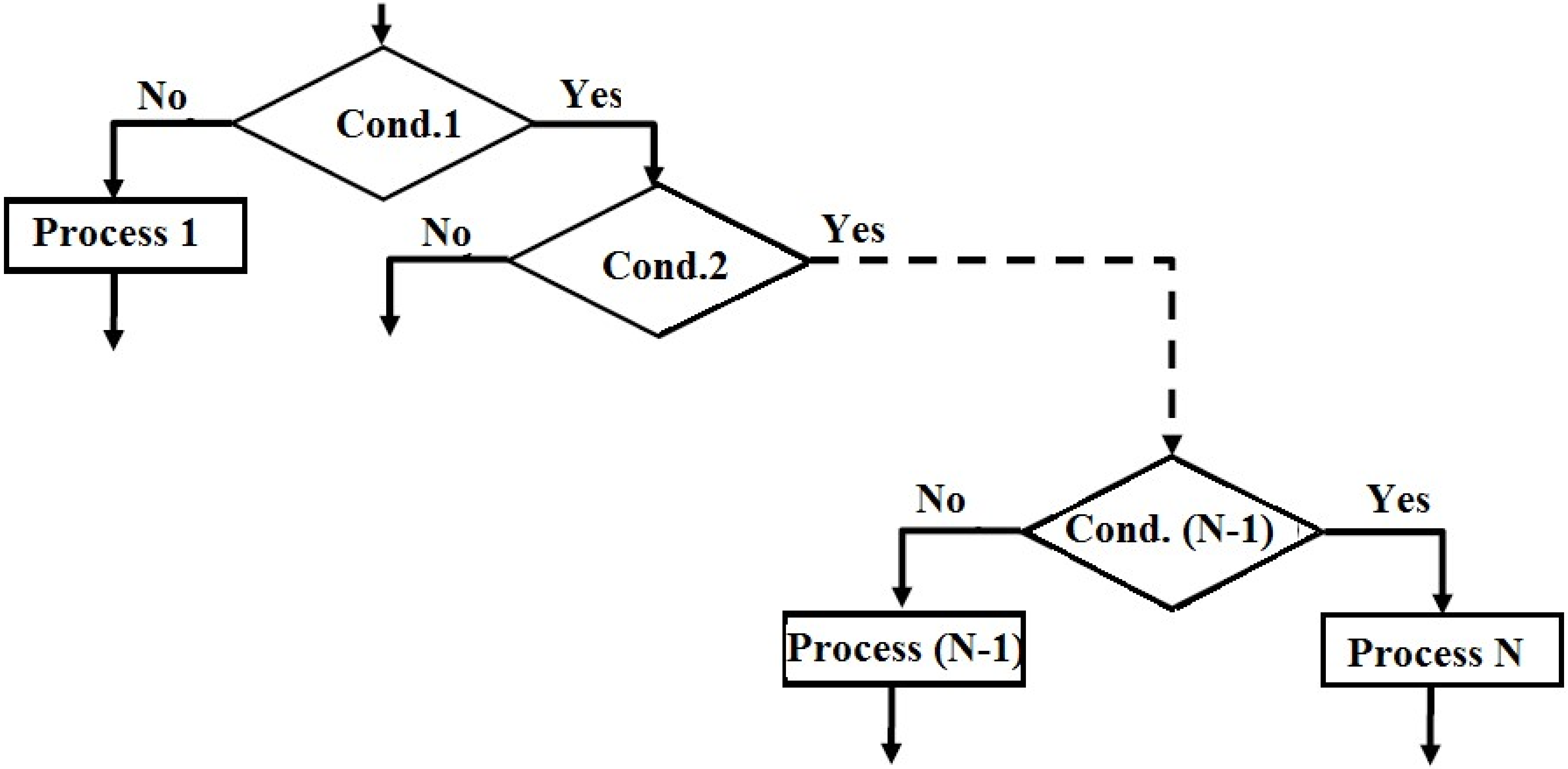
VIII. Flow Lines



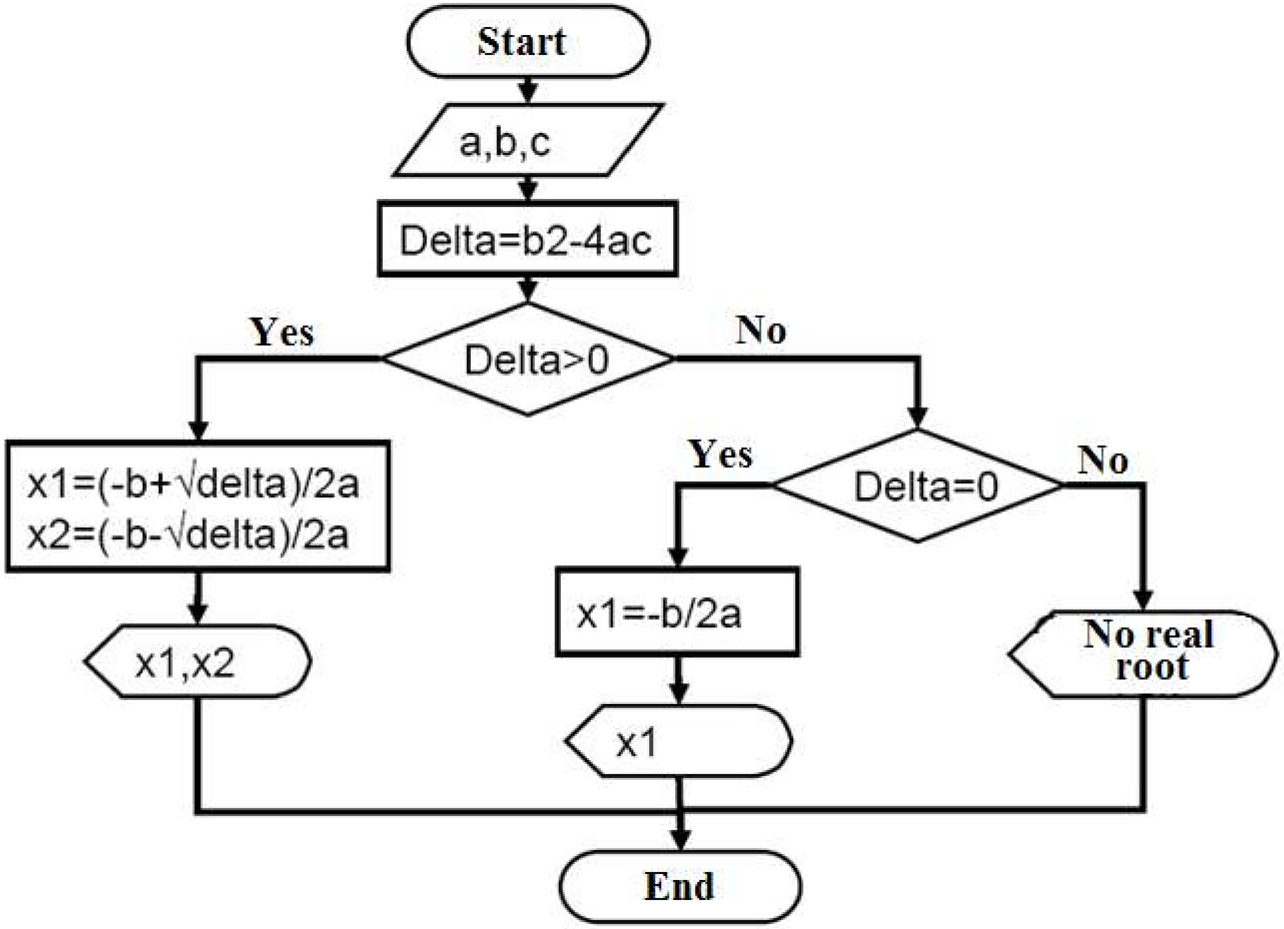
Represents the flow direction of the lines in an algorithm

5. FLOWCHART

A statement can be used successively to make loops



Example: **To find the roots of 2nd order equation $ax^2 + bx + c = 0$, design the flowchart**



Loop Structure

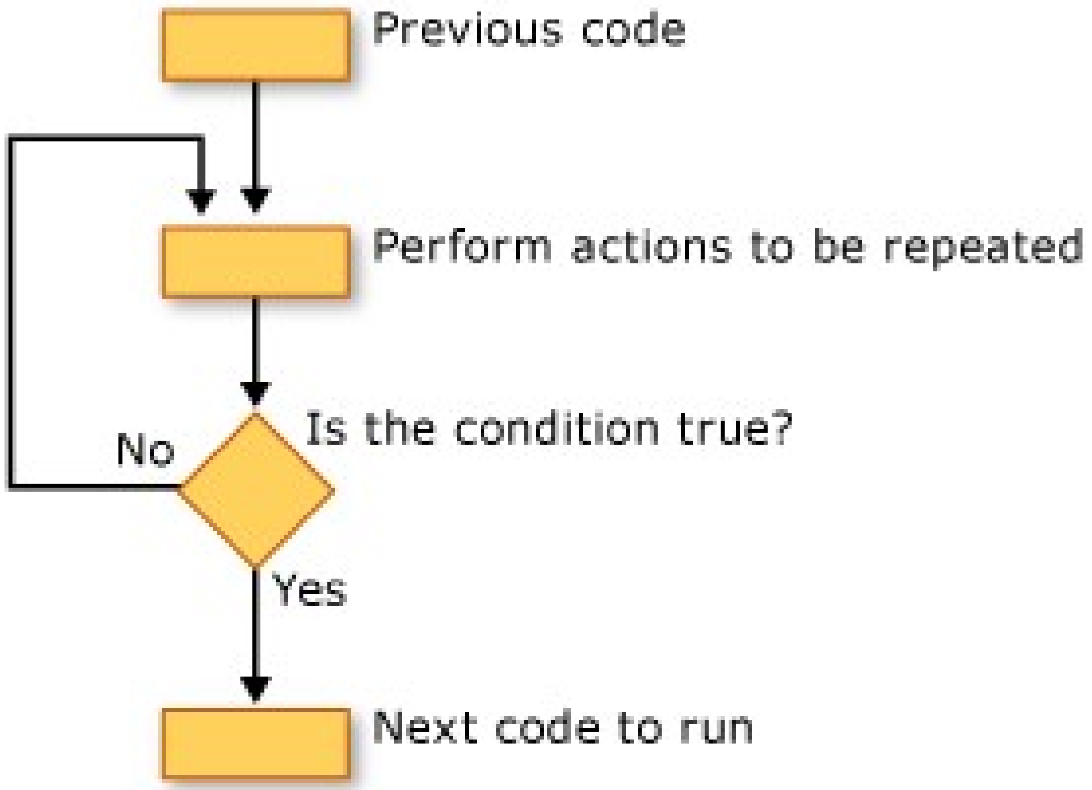
Basic loop structures allow you to run one or more lines of code repetitively. You can repeat the statements in a loop structure until a condition is True, until a condition is False, a specified number of times, or once for each element in a collection.

In general, below terms are used in programming languages;

- **While**
- **Do-while**
- **For**

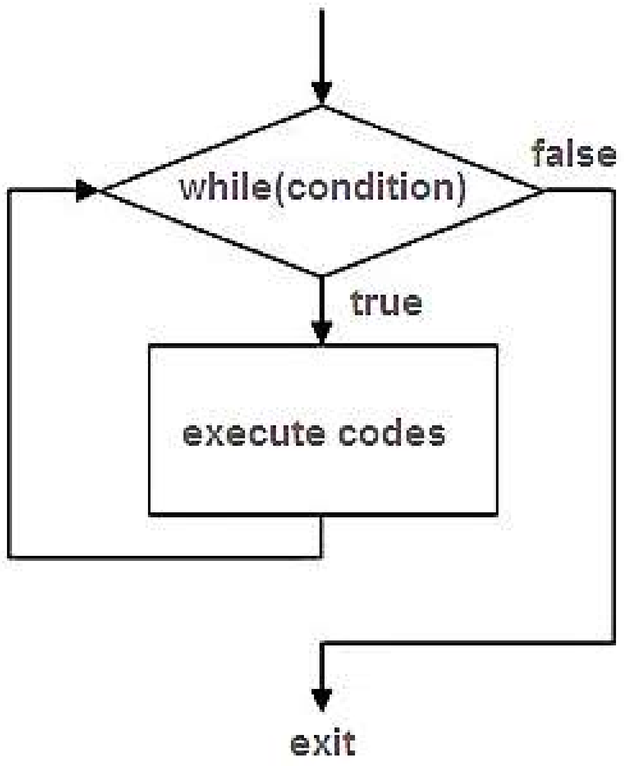
▪ **Although there can be alternatives to these structures in different languages, the running principles are similar to them.**

The following illustration shows a loop structure that runs a set of statements until a condition becomes true.



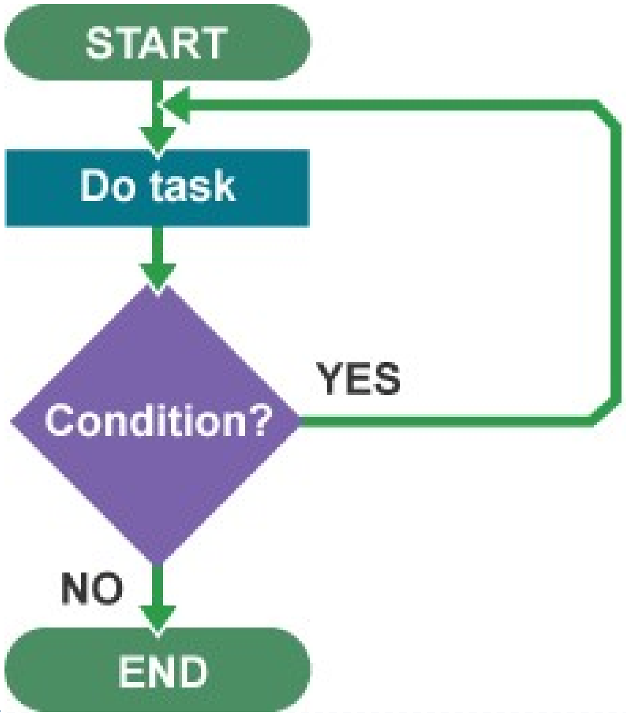
1. Statement (While)

while *expression*, *statements*, end evaluates an expression, and repeats the execution of a group of statements in a loop while the expression is true. An expression is true when its result is nonempty and contains only nonzero elements (logical or real numeric). Otherwise, the expression is false.



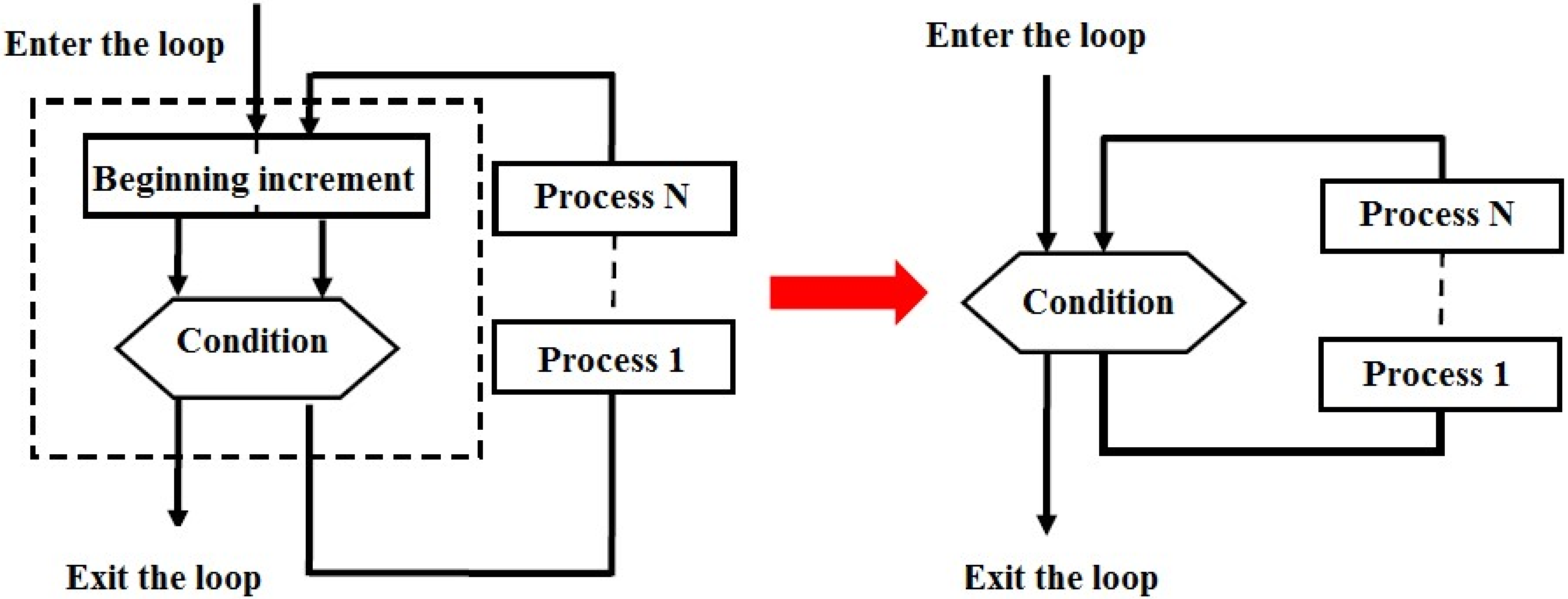
2. Statement (Do-While)

A **do while loop** is a control flow statement that executes a block of code at least once, and then repeatedly executes the block, or not, depending on a given boolean condition at the end of the block.



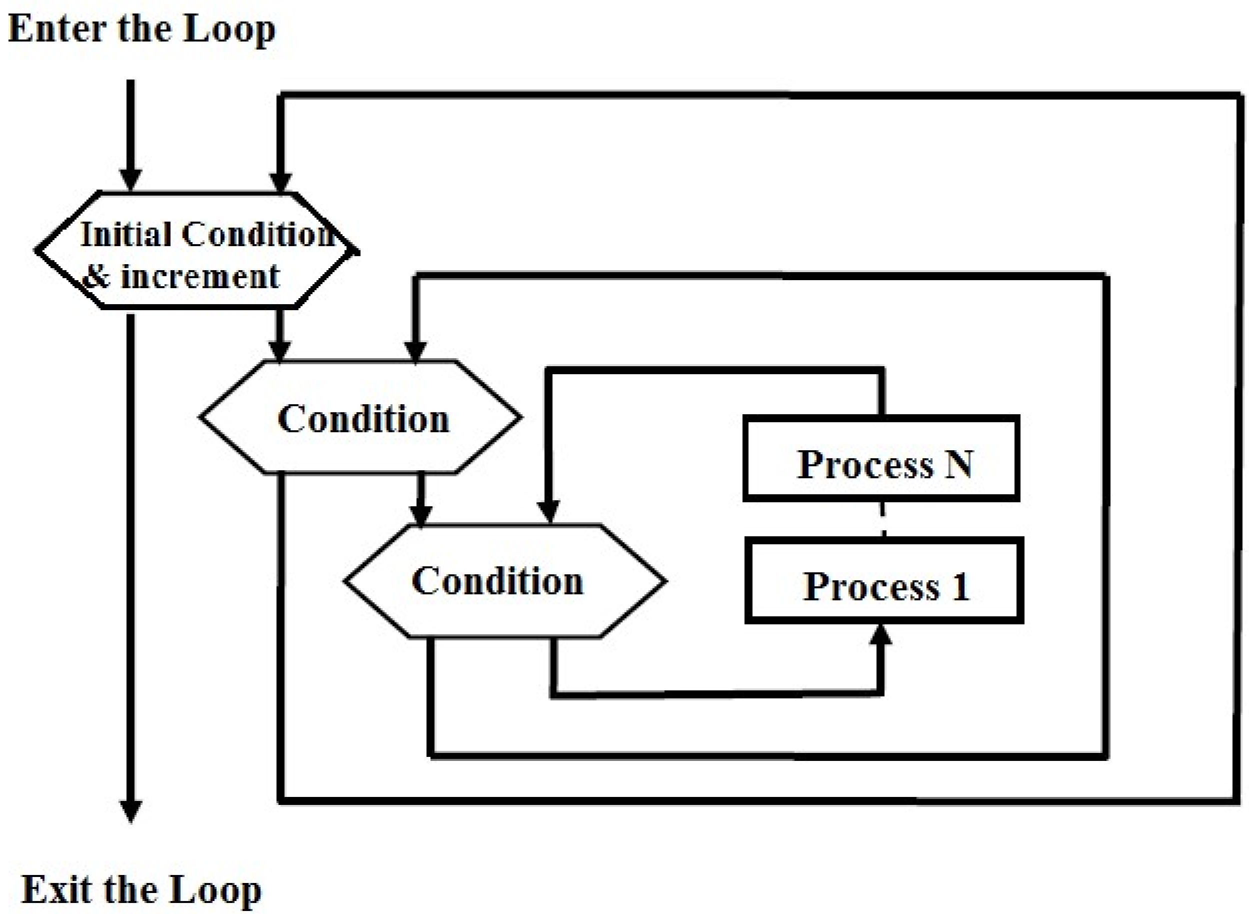
3. Statement (For)

A **for-loop** (or simply **for loop**) is a control flow statement for specifying iteration, which allows code to be executed repeatedly.



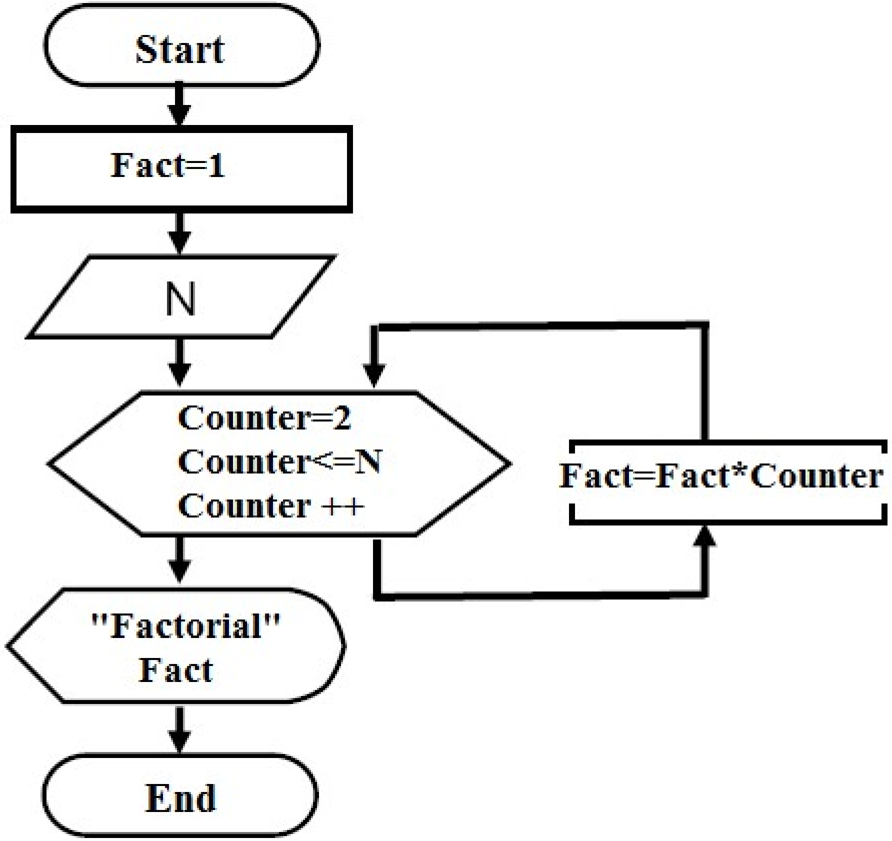
Use of Nested Loop

Rule: First the inner loop should be completed and then outer loop should be run. The loops should not block each other.



In each step of outer loop, the inner loop should be repeated N times.

Example: Design the flow chart of the algorithm, which computes the factorial of N entered by keyboard.



First, determine the N value and design a loop for running N times.

In the first loop, 1!,

In the second loop 2!

And repeatedly in the last loop (N repetition) N!

If Condition (Counter>N) is provided, the loop will be completed.

Print the solution Fact

Outlines For MATlab

- Introduction
- Matrix Operations
- Numerical formats
- Basic Linear Algebra Operations
- Arrays/Vectors
- if-end , switch-case structures
- Loops (for-end and while-end)
- Plots
- File read/write
- Function m files
- Compiler

References

- Doğan, U., (2009), Temel Bilgisayar Bilimleri Ders Notları, YTÜ, Lisans Ders Notları, İstanbul.
- İnan, A., "MATLAB Klavuzu", Papatya Yayınları, İstanbul, 2007.
- Demirel, H., (2005), Dengeleme Hesabı, YTÜ, Lisans Ders Notları, İstanbul.
- Ayten, U. E., "Algoritma geliştirme ve programlamaya giriş", Temel Bilgisayar Bilimleri Ders Notları.
- Serbes, A., "Algoritma geliştirme ve programlamaya giriş", Temel Bilgisayar Bilimleri Ders Notları.
- Uzunoğlu M., vd. (2002), Matlab, Türkmen Kitabevi, İstanbul.
- http://www.mathworks.com/matlabcentral/
- http://www.mathworks.com/matlabcentral/fileexchange/

MATLAB (MATrix LABoratuary)

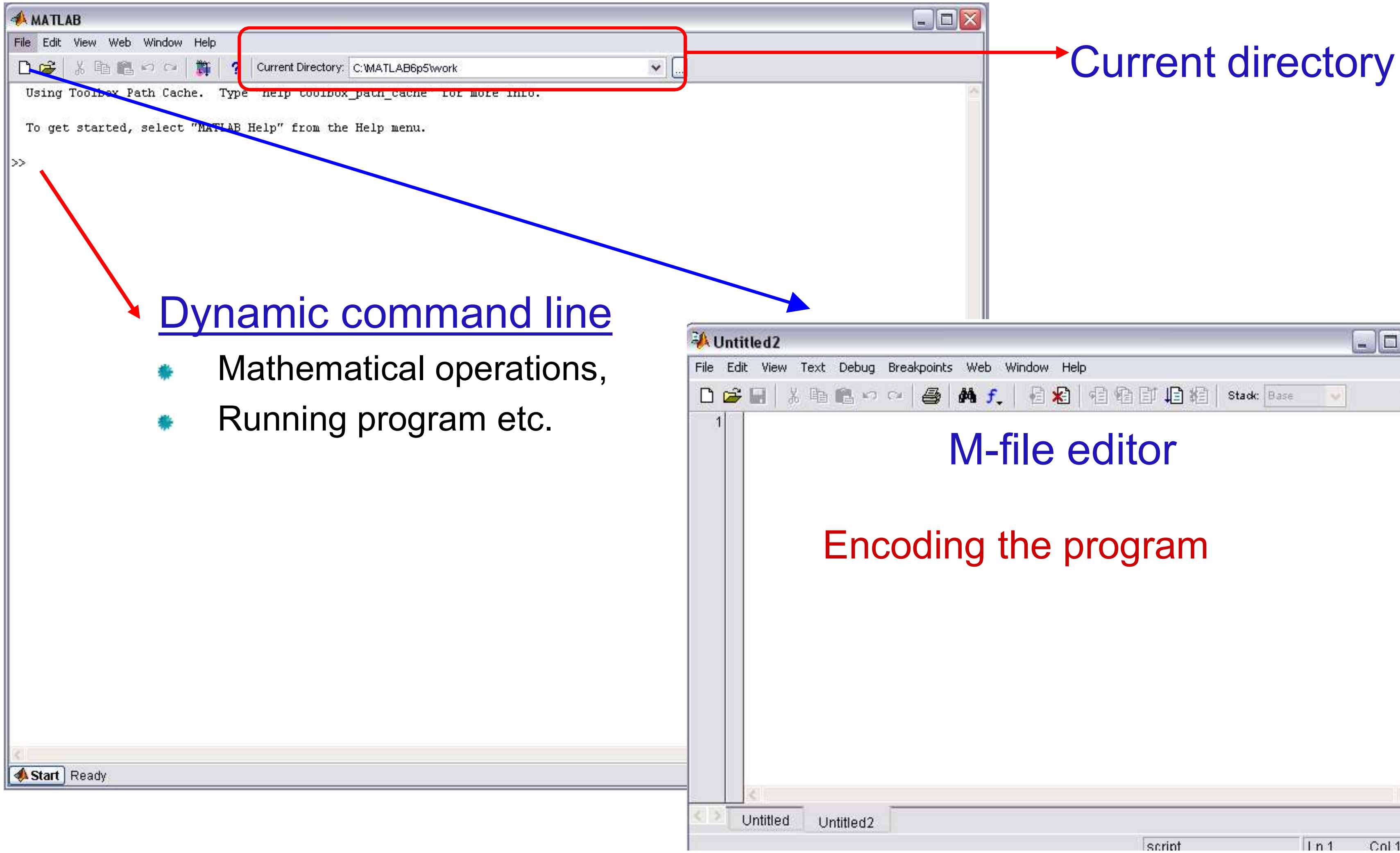
MATLAB (short for MATrix LABoratory) is a special-purpose computer program optimized to perform engineering and scientific calculations. It started life as a program designed to perform matrix mathematics, but over the years it has grown into a flexible computing system capable of solving essentially any technical problem.

http://www.mathworks.com/matlabcentral/

The MATLAB System

- ❑ High level language for technical computing
- ❑ Stands for MATrix LABoratory
- ❑ Everything is a matrix - easy to do linear algebra
- ❑ Development Environment
- ❑ Mathematical Function Library
- ❑ MATLAB language
- ❑ Application Programming Language

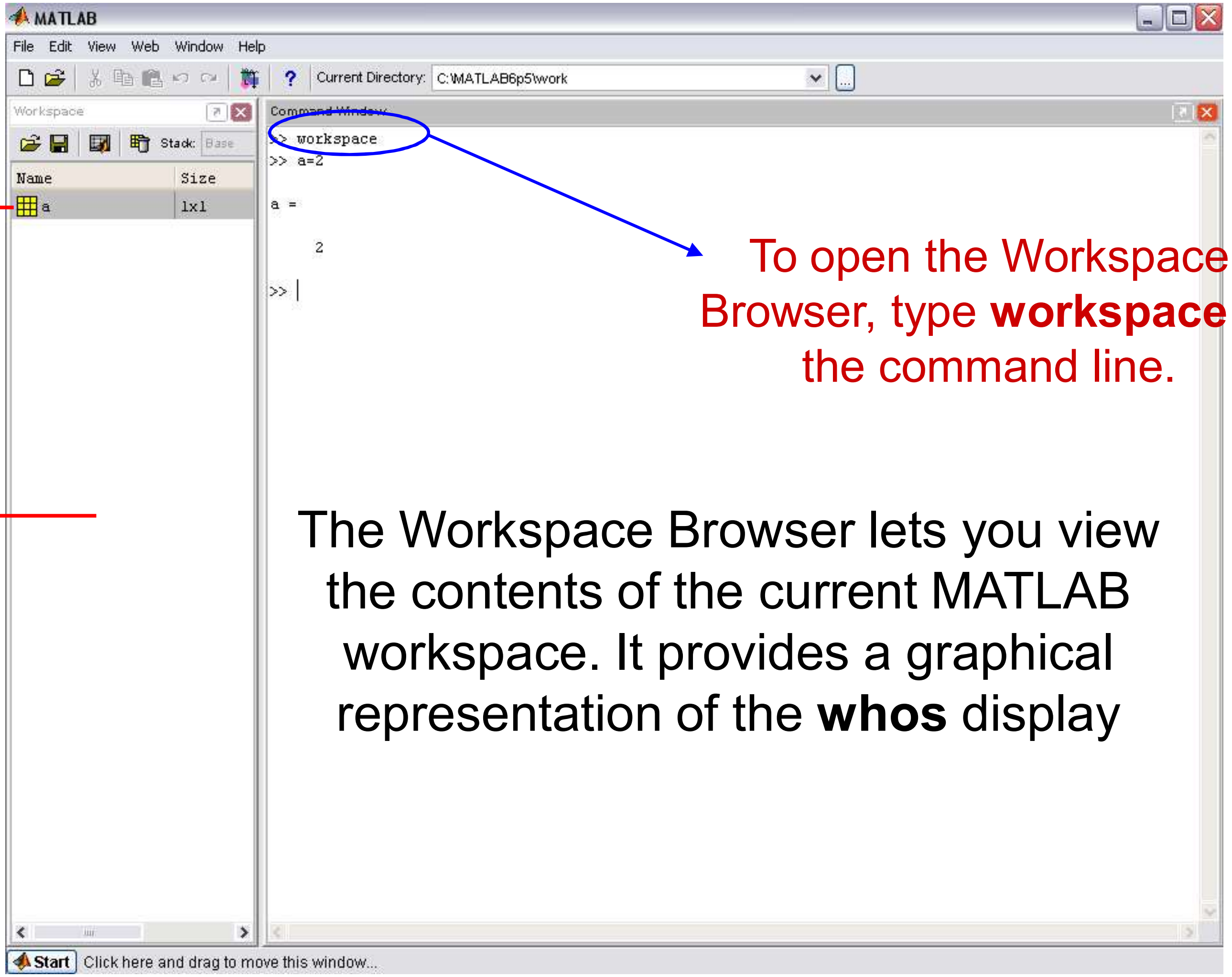
MATLAB/Command window



MATLAB/Workspace

Assigned Variable

Workspace window



To open the Workspace Browser, type **workspace** at the command line.

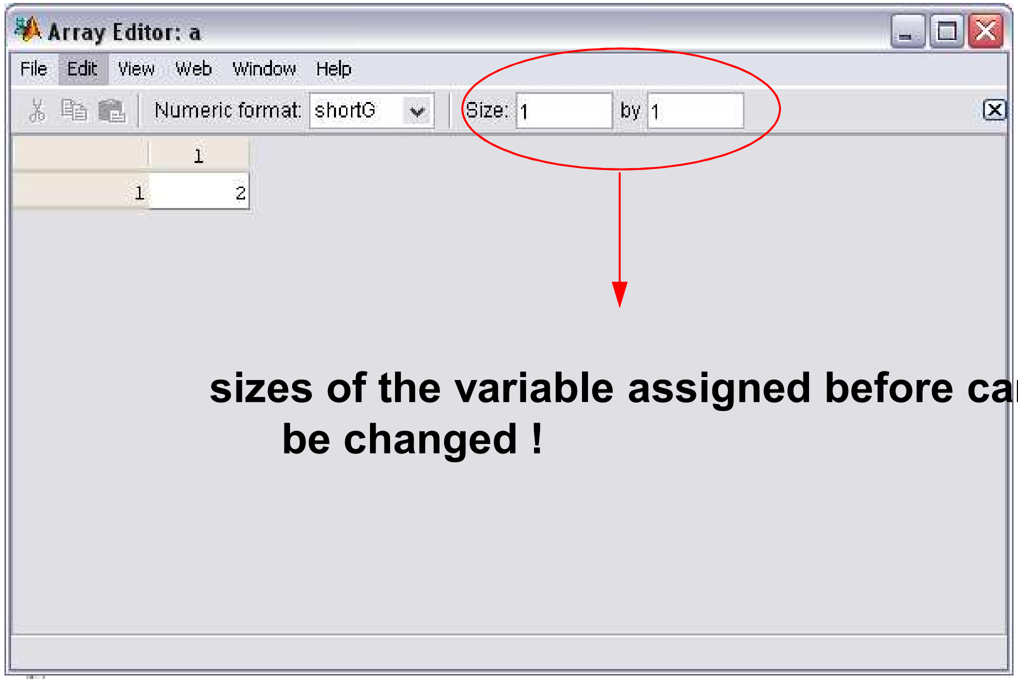
The Workspace Browser lets you view the contents of the current MATLAB workspace. It provides a graphical representation of the **whos** display

MATLAB/Array Editor /Variable Editor

Editor in excel format for matrice, vector and numbers

Two ways to display:

- >>**`open('a')`**
- Double click to the related variable on workspace browser



sizes of the variable assigned before can be changed !

MATLAB/ Basic File Formats

- `*.m` MATLAB program files
- `*.fig` Graphic files and GUI
- `*.mat` Variable and matrices files
- `*.p` pre-parsed pseudo-code files (the content of these files can not be displayed, but can be run as program (run in Matlab)

ARRAYS and VARIABLES

- All MATLAB variables are multidimensional arrays. They can be formed as rows and columns.
- Arrays are divided into two main groups; vector and matrices.
- *Vector*; one-dimensional array.
- *Matrices*; 2 or more dimensional array.
- *Variables*; arrays named by the user

Variable types in MATLAB: “double” ve “char”.

1- Double: These variables can be real, imaginary or complex numbers used to define the scalar (64 byte) or variables.

Example1: deneme = 2 + i

*****double can be converted to the ASCII***

Example2: double(‘deneme’)

ARRAYS and VARIABLES

2- char: converts array A into a character array.

S = char(A)

if A is a string array, then char converts the string array into a character array. char converts each string element of A into a character vector, and then concatenates the vectors to produce a character array

To convert characters into a numeric array, use a function that converts to a numeric type:

- *d=‘selam’*
- *double(d)*
- *g=[115 101 108 97 109]*
- *char(g)*

```
>> d='selam'
d =
selam
>> double(d)
ans =
  115   101   108    97   109
>> char(d)
ans =
selam
```

NAMING ARRAYS and VARIABLES

- **To create a variable, a name should be defined!!**

  >> var = 3.14

  >> string = 'selam'

- **Variable Naming!!**

  first character should be letter! Don't use numbers for starting naming!

  After first letter, there can be number, _ or combinations of them

  Sensitive to capital letter: var and Var are different

  The length of names can be max. 63 characters

- **Do not use constants defined in MATLAB!!**

  pi ➔ 3.1415926…

  ans ➔ shows the last assigned variable

  Inf & –Inf ➔ returns positive and negative infinity

  NaN ➔ 'Not a Number'

>> pi

ans =

3.14159265358979

- **Do not use Turkish letter!!**

  ç, ğ, ı, ö, ş, ü, Ç, Ğ, Ġ, Ö, Ş, Ü

MATLAB
Basic Commands

- `clc` clears all input and output from the Command Window display, giving you a clean screen.
- `clear` removes all variables from the current workspace
- `clear a` removes only variable labelled "a"
- `demo` runs Matlab demo
- `date` Displays on screen Day-Month-Year (Ex: 17-Feb-2009)
- `who/whos` lists in alphabetical order the names of all variables in the currently active workspace/ lists in alphabetical order the names, sizes, and types of all variables in the currently active workspace
- `exit` Terminate MATLAB program (same as quit)
- `help` lists all primary **help** topics in the Command Window
- `help f_na` gives info for f_na function
- `save d a` saves variable «a» with file name «d» as extension **mat**
- `load d` loads variable «a» from MAT-file (d.mat) into workspace

Save ve load commands are crucial for saving matrices etc.

MATLAB
Saving Matrices

- **Command: save; extension *.mat, to recall use load**

- **For example: Let save 'a' matrix in "D:\yildiz" named as "katsayilar.mat"**
  Use below command line:

```
save D:\yildiz\katsayilar a
```

- **To recall/load the 'a' matrix saved as katsayilar.mat,**

```
load D:\yildiz\katsayilar
```

If a new matrix is saved as «katsayilar.mat», there is no possibility to see again the previous version. So, save has overwrite specification.

MATLAB
Creating Matrices

- **Brackets** are used to form **vectors** and **matrices**.
- **Three ways to create matrices and vectors:**

Example:

$$\mathbf{A} = \begin{bmatrix} 1 & 3 & 5 \\ 7 & 8 & 11 \\ 100 & 1 & 4 \end{bmatrix}$$

Way 1:

```
A=[1 3 5
7 8 11
100 1 4]
```

Way 2

```
A=[1 3 5;7 8 11;100 1 4]
```

Way 3:

```
A(1,1)=1,   A(1,2)=3, A(1,3)=5
A(2,1)=7,   A(2,2)=8, A(2,3)=11
A(3,1)=100, A(3,2)=1, A(3,3)=4
```

MATLAB/Basic Algebra Commands

- **`inv(A)`** is the inverse of the square matrix **A**.
- **`A'`** transpose of matrix **A**.
- **`det(A)`** is the determinant of the square matrix **A**.
- **`A+B`** adds matrices **A** and **B** that the sizes are the same
- **`A-B`** subtracts matrices **A** and **B** that the sizes are the same
- **`A*B`** multiply matrices **A** (no. of column: m) and **B** (no. of row: m)
- **`A/B`** If the det(**B**)≠ 0, this command carries out the operation **`A*inv(B)`**
- **`A.*B`** multiplies arrays **A** and **B** element by element
- **`A./B`** divides arrays **A** and **B** element by element.

MATLAB/Basic Algebra Commands

- `trace(A)` is the sum of the diagonal elements of **A**.
- `diag(A)` Diagonal matrices and diagonals of **A** matrix
- `sum(A)` is the sum of the elements of the vector **A**. If **A** is a matrix, Sum is a row vector with the sum over each column.
- `triu(A)` is the upper triangular part of **A**
- `tril(A)` is the lower triangular part of **A**
- `zeros(m,n)` creates an m-by-n matrix of zeros
- `ones(m,n)` creates an m-by-n matrix of ones
- `eye(m)` is the m-by-m identity matrix

MATLAB/Basic operators

`A(:)`	is all the elements of **A**, regarded as a single column.
`A(:,i)`	is the i.th column of **A**
`A(j,:)`	is the j.th row of **A**
`A(:,[i j])`	is the i.th and j.th columns of **A**
`A([i j],:)`	is the ith and jth rows of **A**
`e=a:b:n`	creates a vector start at a, end at n, increment for each step is b.
`e=linspace(a,n,b)`	creates a vector; start at a, end at n, element number b
`e=logspace(a,n,b)`	creates a vector; start at 10^a, end at 10^n, element number b

MATLAB/Basic operators

For Example:

`e=1:1:n`, A vector contains integers from 1 to n.
`e=2:2:n`, A vector contains even integers from 1 to n.
`e=1:2:n`, A vector contains odd integers from 1 to n.
`e=-10:0.1:n`, A vector contains numbers from -10 to n, increment 0.1
`e=linspace(0,10,6)`, e=[0 2 4 6 8 10]
`e=logspace(0,2,3)`, e=[1 10 100]

MATLAB/Basic Matrices Operators

`length(A)`	returns the length of vector A (MAX(SIZE(A)))
`[m,n]=size(A)`	returns the number of rows (m) and columns (n) in a as separate output variables.
`max(A)`	is the largest element in A
`min(A)`	is the smallest element in A
`[m,i]=max(A)`	returns the indices of the maximum values in vector A.If the values along the first non-singleton dimension contain more than one minimal element, the index of the first one is returned.
`[m,i]=min(A)`	returns the indices of the minimum values in vector A.If the values along the first non-singleton dimension contain more than one minimal element, the index of the first one is returned.
`sort(A)`	sorts the elements of A in ascending
`A(:,i)=[]`	Deletes ith column of A
`A(i,:)=[]`	Deletes ith row of A

Creating Matrices

- `zeros(m, n)`: matrix with all zeros
- `ones(m, n)`: matrix with all ones.
- `eye(m, n)`: the identity matrix
- `rand(m, n)`: uniformly distributed random
- `randn(m, n)`: normally distributed random
- `magic(m)`: square matrix whose elements have the same sum, along the row, column and diagonal.
- `pascal(m)` : Pascal matrix.

Some Built-in functions

- `mean(A)`: **mean value of a vector**
- `max(A)`, `min (A)`: **maximum and minimum.**
- `sum(A)`: **summation.**
- `sort(A)`: **sorted vector**
- `median(A)`: **median value**
- `std(A)`: **standard deviation.**
- `det(A)` : **determinant of a square matrix**
- `inv(A)`: **Inverse of a matrix A**

Matrices & Vectors

- All (almost) entities in MATLAB are matrices
- Easy to define:

```
>> A = [16 3; 5 10]
 A =     16      3
             5     10
```

- Use ',' or ' ' to separate row elements
- Use ';' to separate rows

Matrices & Vectors - II

- **Order of Matrix -**

  m=no. of rows, n=no. of columns $m \times n$

- **Vectors - special case**

  n = 1 column vector

  m = 1 row vector

Creating Vectors and Matrices

- **Define**

```
>> A = [16 3; 5 10]
  A =     16      3
           5     10
>> B = [3 4 5
         6 7 8]
        B = 3   4   5
            6   7   8
```

- **Transpose**

Vector :

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

Matrix:

```
>> A=[1 2; 3 4];
     >> A'
     ans =
       1      3
       2      4
```

Array Operations

- **Evaluated element by element**

  `.'` : array transpose (non-conjugated transpose)

  `.^` : array power

  `.*` : array multiplication

  `./` : array division

- **Very different from Matrix operations**

```
>> A=[1 2;3 4];
>> B=[5 6;7 8];
   >> A*B
     19     22
     43     50
```

But:

```
>> A.*B
   5     12
  21     32
```

Indexing Matrices

$$A_{ij}, i = 1 \ldots m, j = 1 \ldots n$$

Given the matrix:

$$\begin{array}{c|ccc} & & n & \\ m & 0.9501 & 0.6068 & 0.4231 \\ & 0.2311 & 0.4860 & 0.2774 \end{array}$$

```
A =
    0.9501    0.6068    0.4231
    0.2311    0.4860    0.2774
```

Then:

```
A(1,2) = 0.6068

A(3) = 0.6068

A(:,1) = [0.9501

          0.2311 ]

A(1,2:3)=[0.6068    0.4231]
```

1:*m*

Adding Elements to a Vector or a Matrix

```
>> A=1:3
A=
  1   2   3
>> A(4:6)=5:2:9
A=
1   2   3   5   7   9

>> B=1:2
B=
  1   2
>> B(5)=7;
B=
1   2   0   0   7
```

```
>> C=[1 2; 3 4]
C=
 1   2
 3   4
>> C(3,:)=[5 6];
C=
 1   2
 3   4
 5   6

>> D=linspace(4,12,3);
>> E=[C D']
E=
 1   2   4
 3   4   8
 5   6   12
```

Creating Vectors

Create vector with equally spaced intervals

```
>> x=0:0.5:pi
x =
0 0.5000 1.0000 1.5000 2.0000 2.5000 3.0000
```

Create vector with *n* equally spaced intervals

```
>> x=linspace(0, pi, 7)
x =
0 0.5236 1.0472 1.5708 2.0944 2.6180 3.1416
```

Equal spaced intervals in logarithm space

```
>> x=logspace(1,2,7)
x =
10.0000 14.6780 21.5443 … 68.1292  100.0000
```

Note: MATLAB uses pi to represent π , uses i or j to represent imaginary unit

Scalar:

```
>> a=12

a =

    12

>> b=78

b =

    78

>> c=4.5

c =

    4.5
```

Vector:

```
>> a=[54 52 0 14 4]

a =

    54    52     0    14     4

>> b=[54;52;0;14;4]

b =

    54
    52
     0
    14
     4
```

```
>> a=a'

a =

    54
    52
     0
    14
     4

>> b=b'

b =

    54    52     0    14     4
```

```
>> a=[2 4]

a =

    2     4

>> b=[6 5]

b =

    6     5

>> c=a+b

c =

    8     9
```

```
>> a

a =

    2     4

>> b

b =

    6     5

>> c=a-b

c =

   -4    -1
```

```
>> a

a =

    2     4

>> b

b =

    6     5

>> c=a*b
Error using  *
Inner matrix
dimensions must
agree.
```

```
>> b=[1:10]
b =

       1     2     3     4     5     6     7     8     9    10
```

```
>> a=[1:2:20]

a =

    1     3     5     7     9    11    13    15    17    19
```

```
>> f=[12:-3:-2]

f =

   12     9     6     3     0
```

According to the given matrix, what are the results of the commands?

```
d =

   2    5    8   11
   8    5    7   21
  88   55   44   33
```

```
>> size(d)

ans =

     3     4
```

```
>> length(d)

ans =

     4
```

```
>> a=d(1:1)
a =
     2
```

```
>> d(1,2)
ans =
     5
```

```
>> d(1,3)
ans =
     8
```

```
>> d(1,4)
ans =
     11
```

```
>> d(2,1)
ans =
     8
```

```
>> d(3,1)
ans =
     88
```

```
>> d(3,2)
ans =
     55
```

According to the given matrix, what are the results of the commands?

```
d =

   2    5    8   11
   8    5    7   21
  88   55   44   33
```

```
>> d(:,1)

ans =

     2
     8
    88
```

```
>> d(:,2)

ans =

     5
     5
    55
```

```
>> d(1,:)

ans =

 2    5    8   11
```

```
>> d(3,:)

ans =

 88   55   44   33
```

How can we find the red colored numbers?

```
d =

  2    5    8   11
  8    5    7   21
 88   55   44   33

>> d(3,1:2)

ans =

 88   55
```

```
d =

  2    5    8   11
  8    5    7   21
 88   55   44   33

>> diag(d)

ans =

  2
  5
 44
```

```
d =

  2    5    8   11
  8    5    7   21
 88   55   44   33

  5    7   21
  2    5    8

>> [d(2,2:4);d(1,1:3)]
```

Calculate the sum of the marked numbers

```
d =

  2    5    8   11
  8    5    7   21
 88   55   44   33
```

Sum of the row

```
>> sum(d(1,:))

ans =

    26
```

```
>> sum(d(2,2:4))

ans =

    33
```

Sum of the colon

```
>> sum(d(:,1))

ans =

    98
```

Exercise 1 Find a *short* MATLAB expression to build the matrix

```
d =

     1     2     3     4     5     6     7     8     9
     9     7     5     3     1    -1    -3    -5    -7
     4     8    16    32    64   128   256   512  1024
```

```
d=[1:9;9:-2:-7;2.^(2:10)]
```

According to the given matrix, find the results of the commands

```
d =

   2    5    8   11
   8    5    7   21
  88   55   44   33
```

```
>> max(d)

ans =

88   55   44   33
```

```
>> min(d)

ans =

2    5    7   11
```

```
>> mean(d)

ans =

32.667   21.667   19.667   21.667
```

```
>> median(d)

ans =

8    5    8   21
```

```
>> sum(d)

ans =

98   65   59   65
```

Give a MATLAB expression that multiplies two vectors to obtain

(a) the matrix $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}$ (b) the matrix $\begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \\ 4 & 4 & 4 \end{pmatrix}$

Example solution:

```
>> a=[1 1 1]' * (1:5)

a =

     1     2     3     4     5
     1     2     3     4     5
     1     2     3     4     5
```

```
>> b=(0:4)' * [1 1 1]

ans =

     0     0     0
     1     1     1
     2     2     2
     3     3     3
     4     4     4
```

MATLAB/Basic Matrices Operators

`sortrows(a,i)` sorts the matrix a based on the columns specified in the vector i

Example:

```
a =
     1    1000
     3      10
     2       5
     4       1
```

```
>> sortrows(a,1)

ans =

     1    1000
     2       5
     3      10
     4       1
```

```
a =
     1    1000
     3      10
     2       5
     4       1
```

```
>> sortrows(a,2)

ans =

     4       1
     2       5
     3      10
     1    1000
```

MATLAB/Assign value to a variable

input	enter data from keyboard
Syntax	x = input(prompt)

```
>> a=input('enter data=')
   enter data=12

   a =

      12
```

If you assign a character to a variable;

str = input(prompt,'s')

Displaying the results-1

disp	Display value of variable
Syntax	disp(X)

```
disp(' ')
disp('    A-Deg   B-Deg   C-Deg')
disp('    =====   =====   =====')
disp(rand(4,3))
```

On screen:

```
    A-Deg   B-Deg   C-Deg
    =====   =====   =====
  0.1389    0.2722    0.4451
  0.2028    0.1988    0.9318
  0.1987    0.0153    0.4660
  0.6038    0.7468    0.4186
```

```
name = 'Alice';
age = 12;
X = [name,' will be ',num2str(age),' this year.'];
disp(X)
```

Displaying the results-2

fprintf	Write data to text file
Syntax	fprintf(formatSpec,A1,...,An)
formatSpec	Format of the output fields, specified using formatting operators.
Value (A1,…An)	output.

```
x =
            541.45
>> fprintf('x is %f m. \n',x)
x is 541.450000 m.
```

```
x =
            541.45
>> fprintf('x= %f m. \n',x)
x= 541.450000 m.
```

MATLAB

```
>> x=123.2;
>> fprintf('output = %5.1f \n', x)
output= 123.2
```

```
fprintf(output = %5.1f   \n', x)
```

Area size

Decimal part

```
fprintf('x= %5.2f m. \n',x)
x= 541.45 m.
```

```
>> fprintf('x= %0.2f m. \n',x)
x= 541.45 m.
```

```
>> fprintf('x= %55.2f m. \n',x)
x=                                                541.45 m.
```

```
>> fprintf('x= %0.5f m. \n',x)
x= 541.45000 m.
```

fprintf Formatting Operator

Conversion	*Details*
%e	Exponential notation, such as 3.141593e+00 (Use a precision operator to specify the number of digits after the decimal point.)
%E	Same as %e, but uppercase, such as 3.141593E+00 (Use a precision operator to specify the number of digits after the decimal point.)
%f	Fixed-point notation (Use a precision operator to specify the number of digits after the decimal point.)
%s	Character vector or string array. The type of the output text is the same as the type of format.
%d	Base 10

Text Before or After Formatting Operators-1

formatSpec can also include additional text before a percent sign, **%**, or after a conversion character.

Special Character	Representation
Single quotation mark	' '
Percent character	%%
Backslash	\\
Backspace	\b
Form feed	\f
New line	\n
Carriage return	\r
Horizontal tab	\t
Vertical tab	\v

Text Before or After Formatting Operators-2

Command Window

```
>> a=100.25;
>> b=511.12;
>> fprintf('output a= %5.2f\r and output b= %5.3f\n',a,b)
output a= 100.25
 and output b= 511.120
>> fprintf('output a= %5.2f\b and output b= %5.3f\n',a,b)
output a= 100.2 and output b= 511.120
>> fprintf('output a= %5.2f\t and output b= %5.3f\n',a,b)
output a= 100.25     and output b= 511.120
```

Displaying the results-3

sprintf	Format data into string
Syntax	str = sprintf(formatSpec,A1,...,An)

```
>> a=542.87
a =
     542.87
>> out=sprintf('a=%5.2f',a)
out =
   a=542.87
```

```
>> a=542.87
a =
          542.87
>> out=fprintf('a = %5.2f',a)
a = 542.87
out =

  10
```

Count the expression

```
>> a=5;
>> b=6;
>> str=sprintf('The size is %dmx%dm',a,b)
str =
The size is 5mx6m
```

Displaying the results-4

sscanf	Read formatted data from string
Syntax	A = sscanf(str,formatSpec)

```
>> chr = '12.1452  13.8457  10.7841'
    chr =

             12.1452  13.8457  10.7841

>> A=sscanf(chr,'%f')

A =

               12.1452
               13.8457
               10.7841
```

ARRAYS

Numbers	Numeric array
Characters	Character array

For instance:

```
c=1999                                  (numeric array)
d='Yildiz Teknik Universitesi'          (character array)
f=[1999 2000]                           (numeric, matrix)
g=[d ' Insaat Fakultesi']               (character, matrix)
```

Warning: Numeric and character arrays can not be found in the same matrix! A matrix can only contain either numeric or character values

Cells	Cell array
Structures	Structure array

Character Arrays

```
>> lecture='Introductory Computer Sciences'

lecture =

Introductory Computer Sciences
```

```
>> code=double(lecture)

code =

  Columns 1 through 15

   73   110   116   114   111   100   117    99   116   111   114   121    32    67   111
  Columns 16 through 30
  109   112   117   116   101   114    32    83    99   105   101   110    99   101   115
```

String to ASCII

```
>> char(code)

ans =

Introductory Computer Sciences
```

ASCII to string

```
>> x='ENF1170';
>> a=[lecture,' lecture code ',x]
a =
Introductory Computer Sciences lecture code ENF1170
```

Combining characters

Comparing Character Arrays-1

strcmp	Compare strings. tf = strcmp(s1,s2) compares s1 and s2 and returns 1 (true) if the two are identical and 0 (false) otherwise. Text is considered identical if the size and content of each are the same. The return result tf is of data type logical.
strcmpi	Compare strings (case insensitive)
strncmp	Compare first n characters of strings (case sensitive)
strncmpi	Compare first n characters of strings (case insensitive)

Comparing Character Arrays-1

```
>> a='matematik';
>> b='mathematik';
>> c='MaTematiK';
>> e='matematik';
>> x=strcmp(a,b)

x =

    0

>> x=strcmp(a,e)

x =

    1
>> x=strcmp(a,c)

x =

    0
```

sensitive

```
>> a='matematik';
>> b='mathematik';
>> c='MaTematiK';
>> e='matematik';
>> x=strcmpi(a,b)

x =

    0

>> x=strcmpi(a,c)

x =

    1
```

insensitive

```
>> a='matematik';
>> b='mathematik';
>> c='MaTematiK';
>> e='matematik';
>> x=strncmp(a,b,4)

x =

    0

>> x=strncmp(a,b,2)

x =

    1
>> x=strncmp(a,c,1)

x =

    0
```

sensitive

```
>> a='matematik';
>> b='mathematik';
>> c='MaTematiK';
>> e='matematik';

>> x=strncmpi(a,c,1)

x =

    1
```

insensitive

Comparing Character Arrays-2

Comparing arrays one by one	The lengths should be the same	> , <
		>= , <=
		== , ~=

```
>> x='matlab';
>> y='matema';
>> x==y
ans =
    1    1    1    0    0    0
```

upper	Convert string to uppercase
lower	Convert string to lowercase

```
>> upper('matLab')

ans =

MATLAB
```

```
>> lower('MATlAB')

ans =

matlab
```

Comparing Character Arrays-3

isletter	Determine which character array elements are letters	TF = isletter(A)
isspace	Determine which character array elements are space characters	TF = isspace(A)
ischar	Determine if input is character array	tf = ischar(A)

```
>> lecture='ICS Code:1170'

lecture =

ICS Code:1170

>> chr=isletter(lecture)

chr =

   1   1   1   0   1   1   1   1   0   0   0   0   0
```

```
>> lecture='ICS Code:1170';


>> chr=ischar(lecture)

chr =

   1
```

```
>> lecture='ICS Code:1170'

lecture =

ICS Code:1170

>> chr=isspace(lecture)

chr =

   0   0   0   1   0   0   0   0   0   0   0   0   0
```

```
>> code=1170;
>> chr=ischar(code)

chr =

   0
```

Cell Arrays

cell array	create a cell array using the {} operator
	When you have data to put into a cell array, create the array using the cell array construction operator, {}.

Example:

```
C{1}=[1 2;3 5];
C{2}=[4 4 4 4];
C{3}=[('yildiz teknik'),(' insaat')];
```

```
C =
```

[2x2 double]	[1x4 double]	[1x20 char]
C{1} - Cell	C{2} - Cell	C{3} - Cell

<u>Each cell is seperately represented</u>

Cell Arrays

C=cell(n)	is an N-by-N cell array of empty matrices.

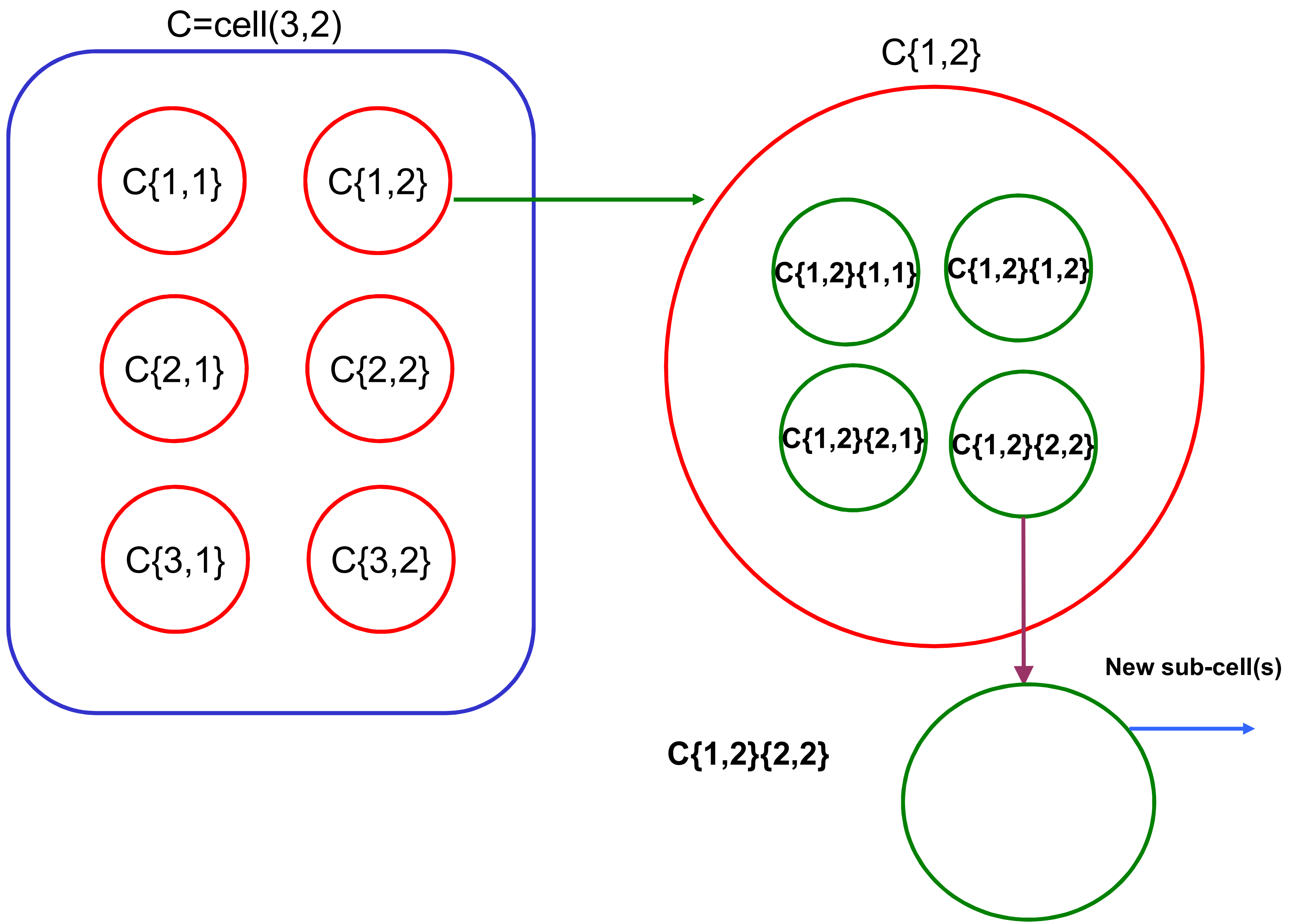
For n=2;

```
>> C=cell(2)

C =

    []        []
    []        []
```

It is possible to add new cells in a cell.
For example:
We can add variables into C.

```
C{1}{1}=[2 3]
C =

  {1x1 cell}      []
      []          []
```

Cell Arrays


C=cell(3,2)
C{1,1}
C{1,2}
C{2,1}
C{2,2}
C{3,1}
C{3,2}
C{1,2}
C{1,2}{1,1}
C{1,2}{1,2}
C{1,2}{2,1}
C{1,2}{2,2}
New sub-cell(s)
C{1,2}{2,2}

Structure Array

- **Structure arrays** ***used for databases***

```
A.name= 'Bahattin';
A.sname='Erdogan';
A.univ='YTU';
A.city='Istanbul';
A.email= 'berdogan@yildiz.edu.tr';
A.year=2018;
```

that provides structure array - **A**

Recalling A,

```
>>A

A =
     name:  'Bahattin'
    sname:  'Erdogan'
     univ:  'YTU'
     city:  'Istanbul'
    email:  'berdogan@yildiz.edu.tr'
     year:  2017
```

Cell and structure arrays can be saved with mat extension (save command) and recalled by load

Data Type Conversion

Converting between numeric arrays, character arrays, cell arrays, structures, or tables

Functions

`char`	Convert to character array
`cellstr`	Convert to cell array of character vectors
`int2str`	Convert integers to character array
`mat2str`	Convert matrix to character vector
`num2str`	Convert numbers to character array
`str2double`	Convert string to double precision value
`str2num`	Convert character array to numeric array

Conversion between arrays

- `num2str(a)`   Convert numbers to a string. (From **num**eric to (**2**) **str**ing)

```
>> a=25;
>> tr=num2str(a)
tr =
25
>> ischar(tr)

ans =

    1
```

- `str2num(a)`   Convert string matrix to numeric array

```
>> val=str2num(tr)
val =
   25

>> isnumeric(val)

ans =

    1
```

Conversion between arrays

- **`mat2str(a)`** Convert a 2-D matrix to a string in MATLAB syntax

```
>> val=mat2str(rand(2))
val =
[0.63235924622541 0.278498218867048;0.0975404049994095 0.546881519204984]
>> ischar(val)
ans =
    1
>> isnumeric(val)
ans =
    0
```

- **`int2str(a)`** Convert integer to string.

```
>> a=154.411
a =
  154.4110
>> val=int2str(a)
val =

154
```

Conversion between arrays

- `char(a)` Create character array (string)

```
>> val{1,1}='7'
val =
    '7'
>> val{1,2}='8'
val =
    '7'    '8'
>> val{2,1}='5'
val =
    '7'    '8'
    '5'     []
>> val{2,2}=['1' '2';'0' '3']
val =
    '7'    '8'
    '5'    [2x2 char]
>> search=char(val)
search =

7
5
8
12
03
```

Conversion between arrays

- `num2cell(a)` Convert numeric array into cell array.

```
>> a=2;
>> tr=num2cell(a)

tr =

   [2]
```

Conversion between arrays

Example: Assume that the result is a=10.234 .
to represent the expression (character), "The result obtained=10.234"

```
['The result obtained=' num2str(a)]
```

Both should be string!

Or; it can be written using **fprintf:**

- fprintf('The result obtained= %6.3f \n',a)

Trigonometric functions

sin(x)	Sine of argument in radians.
asin(x)	Inverse sine, result in radians.
cos(x)	Cosine of argument in radians
acos(x)	Inverse cosine, result in radians.
tan(x)	Tangent of argument in radians.
atan(x)	Inverse tangent, result in radians..
cot(x)	Cotangent of argument in radians.
acot(x)	Inverse cotangent, result in radian.
sec(x)	Secant of argument in radians.
asec(x)	Inverse secant, result in radians.
csc(x)	Cosecant of argument in radians.
acsc(x)	Inverse cosecant, result in radian.

MATLAB/Expressions in Programming

It is needed that a piece of code that executes a series of commands, if and only if some condition is met. MATLAB provides several built-in statements that allow for conditional behavior.

These are:

- `if/elseif/else`
- `switch, case`
- `try/catch`

MATLAB/if, else, elseif, end

- `if` (eğer) Execute statements if condition is true.

```
if expression           if expression            if expression
    statement               statement                statement
end                      else                     elseif
                            statement                statement
                         end                      elseif
                                                     statement
                                                  end
```

Example: If a number entered by user is negative, change the value of it with logarithmic value of itself:

```
a=input(' enter a number= ');
if a<0
   a=log(a);
else
a=a;
end
a
```

"otherwise" :

Here, it is a condition for a>0

Without using else

```
a=input(' enter a number= ');
if a<0
   a=log(a);
end
if a>0
   a=a;
end
a
```

Example: Enter a number from keyboard and take appropriate action depending on the number in three options.
First set min and max values.
If your number exceeds max. value, display a message mention that.
If your number is under min value, display a message mention that.
If your number is between the range, display a message mention that.

```
x = input('enter a number = ');
minVal = 3;
maxVal = 8;
if (x >= minVal) && (x <= maxVal)
disp('Value within specified range.')
elseif (x > maxVal)
disp('Value exceeds maximum value.')
else
disp('Value is below minimum value.')
end
```

MATLAB/switch,case

switch (değiştir) evaluates an expression and chooses to execute one of several groups of statements. Each choice is a case. The switch block tests each case until one of the case expressions is true.

```
switch switch_expression
    case case_expression
         statement
    case case_expression
         statement
    otherwise
         statement
end
```

Up to user

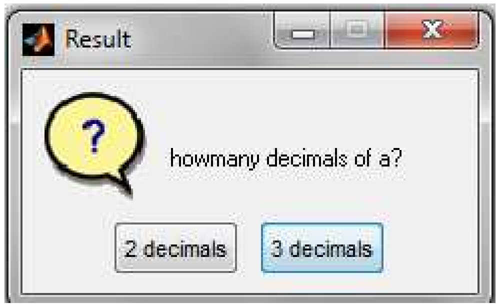
Example: for a variable namely day, decide whether it is working day or not;

```
clear,clc
   day=input('which day=', 's');
   switch lower(day)
        case {'monday', 'tuesday','wednesday','thursday','friday'}
        disp('working day')
        case {'saturday','sunday'}
        disp('HOLIDAY!')
   end
```

MATLAB/switch,case

- **Assume that a variable is accessed by user (a=10.2424542). Let us propose a GUI (questdlg), which decides to represent the result with 2 decimals or 3 decimals:**

```
a=10.2424542;
button=questdlg(‘howmany decimals of a?', ‘Result','2 decimals', '3 decimals','3 decimals');
switch button
       case {'2 decimals'}
       fprintf('%1.2f',a)
       case {'3 decimals'}
       fprintf('%1.3f',a), end
```



Click to “2 decimals” ,

`give the result as`

`10.24`

button = questdlg(qstring,title,str1,str2,default)

MATLAB/for,end

- **for,end for** loop to repeat specified number of times

```
for index = values   ➔ i=1:n     (i→ (integer))
    statements
end
```

Example: Design a loop for summing numbers from 1 to N

```
clear,clc
N=input('enter a number=');
count=0; %counter
for i=1:N
    count=count+i; %cumulative sum of numbers
end
count
```

MATLAB/while,end

- while,end loop to repeat when condition is true

```
done=0;
while done==0 (expression)
    statements
end
```

1. While, end loop can only be processed in case of done is 0.
2. To run while loop, varible done should be assigned as 0.

Example: Assume that we design a program with while, end to compute the sum of the numbers from 1 to N.

```
clear,clc
N=input(enter a number=');
count=0; i=0;done=0;
while done==0
    i=i+1;  %it corresponds to i (for,end) in the previous example.
    if i==N
    done=1;
    end
count=count+i;
end
count
```

When i is the last number (N), a number differs from 0 is assigned to variable done.
So, in the command line of while, while, end loop does not work (because done is not 0 at this situation).
The program continues running after the end command line of this loop.
(Here, variable count is represented in the command window)

MATLAB/
break

```
for i=1:n
     statement
if condition
   break;
end
end
statement
```

```
 done=0;
while done==0

       statement

if condition
   break;
end
end
statement
```

- **break** Terminate execution of for or while loop

MATLAB/

return

```
for i=1:n
    statement
if condition
   return;
end
end
program ends
here
```

```
done=0;
while done==0

       statement

if condition
   return;
end
end
program ends
here
```

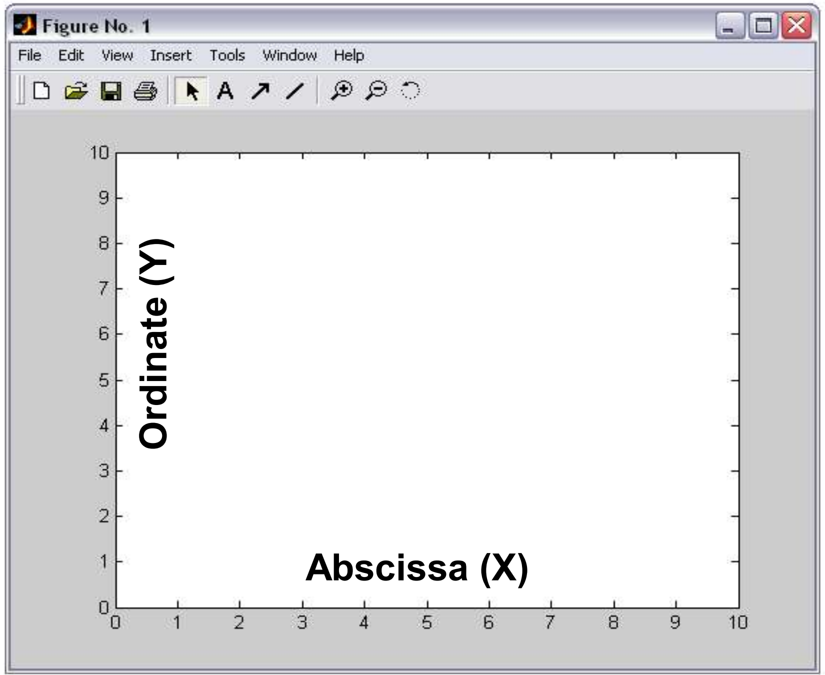
- **Return** control to invoking function

MATLAB/Graphics

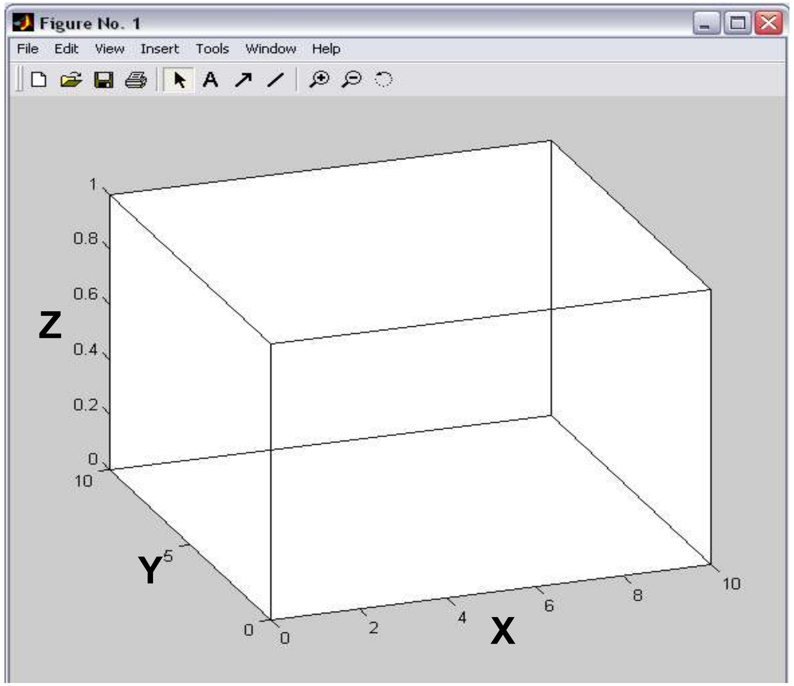
- **In Matlab, graphics are drawn in “figure” window.**

- **2D or 3D graphics are available. Also, graphics can be drawn in polar coordinate system (see, *polar*).**

2 Dimensional Coordinate System



3 Dimensional Coordinate System



2D Graphics

plot function

- The basic command for graphics is plot.

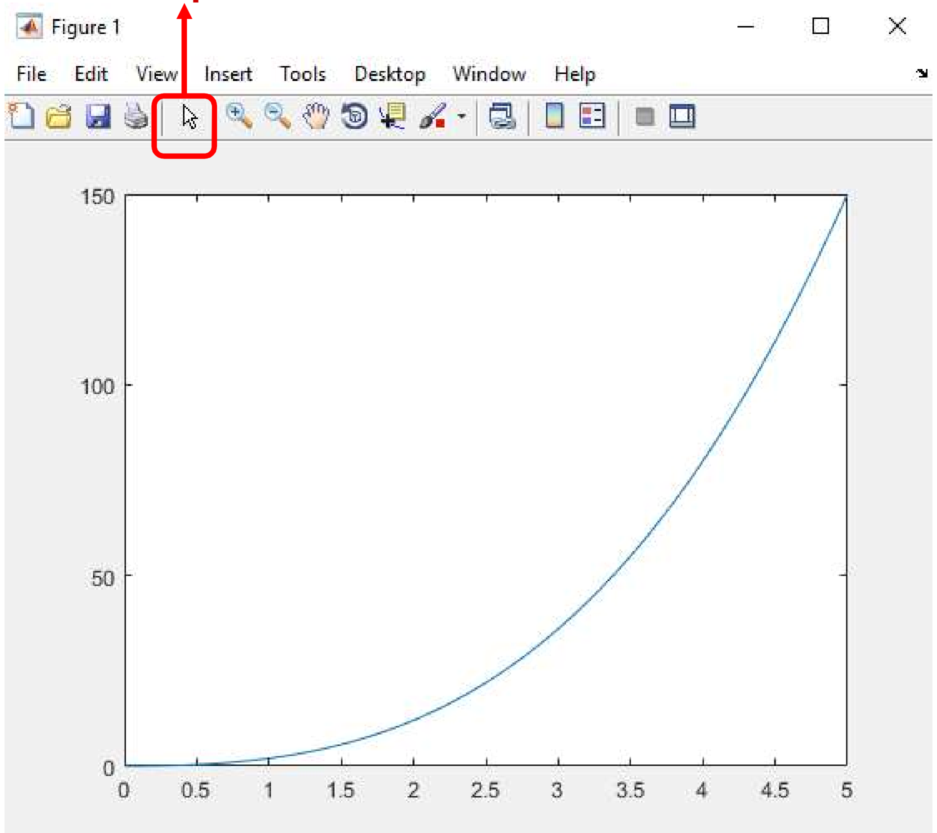
plot	2-D line plot
plot(X,Y) plot(X,Y,LineSpec) plot(X1,Y1,...,Xn,Yn) plot(X1,Y1,LineSpec1,...,Xn,Yn,LineSpecn) plot(Y) plot(Y,LineSpec)	
plot(X,Y) creates a 2-D line plot of the data in Y versus the corresponding values in X.	

For example:

Compute the values of y using the function (y=x.^3+x.^2) that correspond to x=0:0.1:5 (array vector).

To draw the graphic for x and y ➔ plot(x,y)

Edit plot

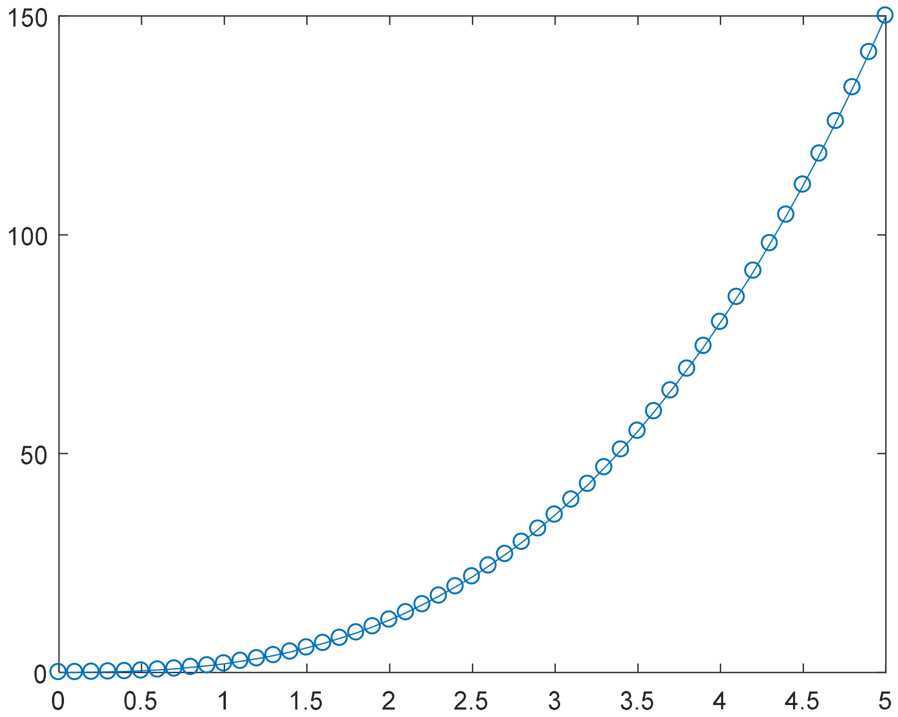


- You can edit the graphic.
- For editing, click the button of “Edit plot”.
- The related object (arc drawn, axes etc.) can be changed by double-clicking the related object to be edited via “Property Editor” window.
- Also, commands can be used to realize changes on the figure.

For example, `plot(x,y,'-o')` draw the figure both connecting the successive points and marked as “o” symbol.

FIGURE

- `plot(x,y,'-o'):`



- **Repeat plotting using below properties:**

```
plot(x,y,'-o')
plot(x,y,'-*')
plot(x,y,'-+')
plot(x,y,'-^')
plot(x,y,'-.')
```

- **Such symbols (o,*,+) on figure are called as marker.**

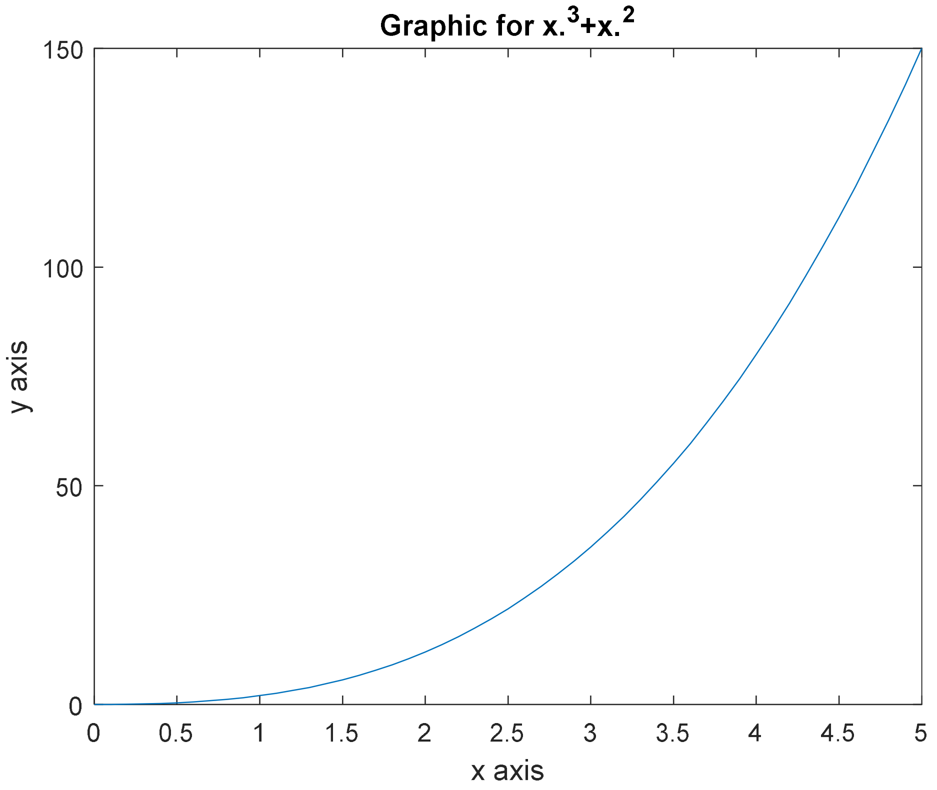
- **Also, the color of the graphic can be changed :**

```
plot(x,y,'r')  (red)
plot(x,y,'k')  (black)
plot(x,y,'b')  (blue)
plot(x,y,'g')  (green)
```

title, xlabel, ylabel

- **We can add graphic title and labels for axes. To represent them in a figure, we use "title", "xlabel" and "ylabel" functions.**

```
>> x=[0:0.1:5];
>> y=x.^3+x.^2;
>> plot(x,y)
>> title('Graphic for x.^3+x.^2')
>> xlabel('x axis')
>> ylabel('y axis')
```



axis

- **Matlab allows to change only specific configurations of plot. Such as:**

Function	*Description*
axis([xmin xmax ymin ymax])	Set axis limits and aspect ratios.
axis equal	Use the same length for the data units along each axis
axis square	Use axis lines with equal lengths. Adjust the increments between data units accordingly.
axis normal	Restore the default behavior.
axis off	Axis visibility is off
axis on	Axis visibility is on

scatter function

- **scatter(X,Y)** **Scatter/bubble plot.**

Example:

X=rand(100,1)*5;

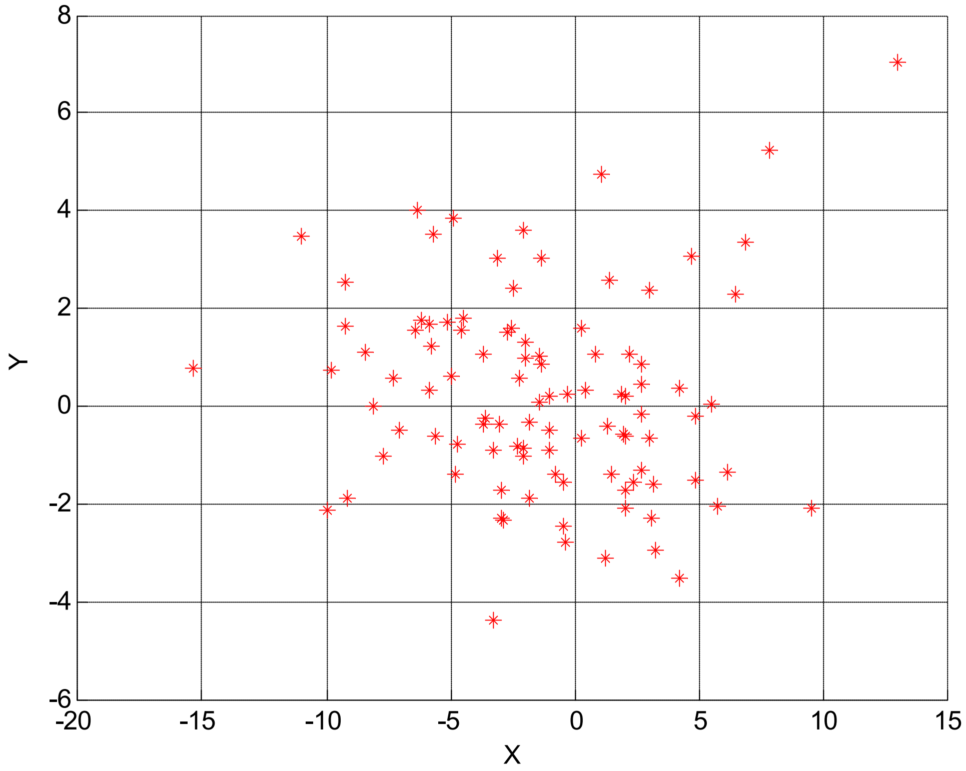
Y=rand(100,1)*2;

scatter(X,Y,'r*')

grid on

xlabel('X')

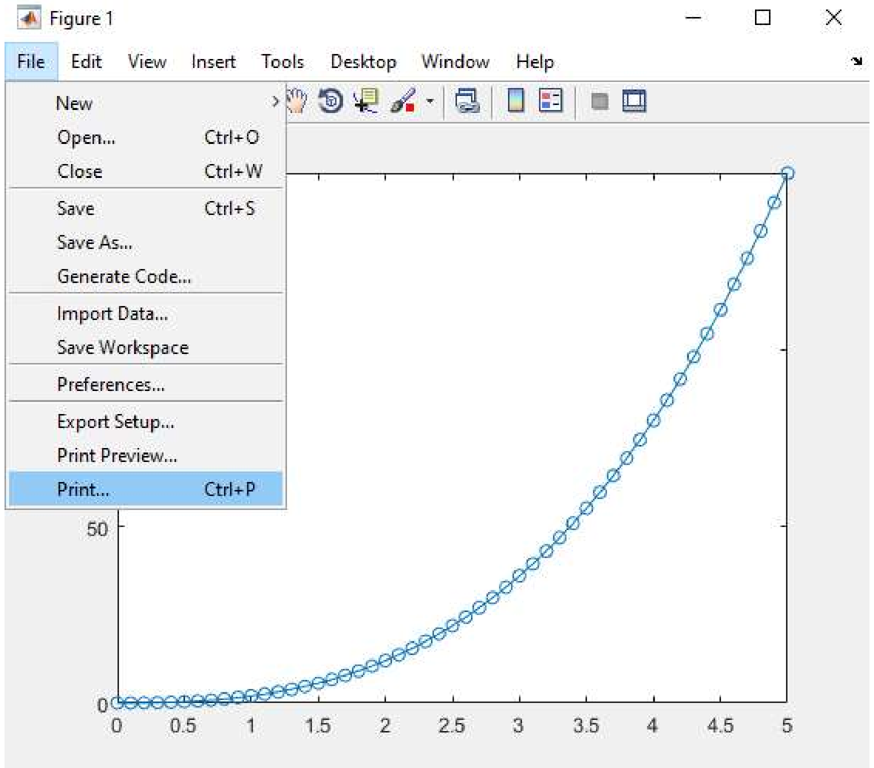
ylabel('Y')

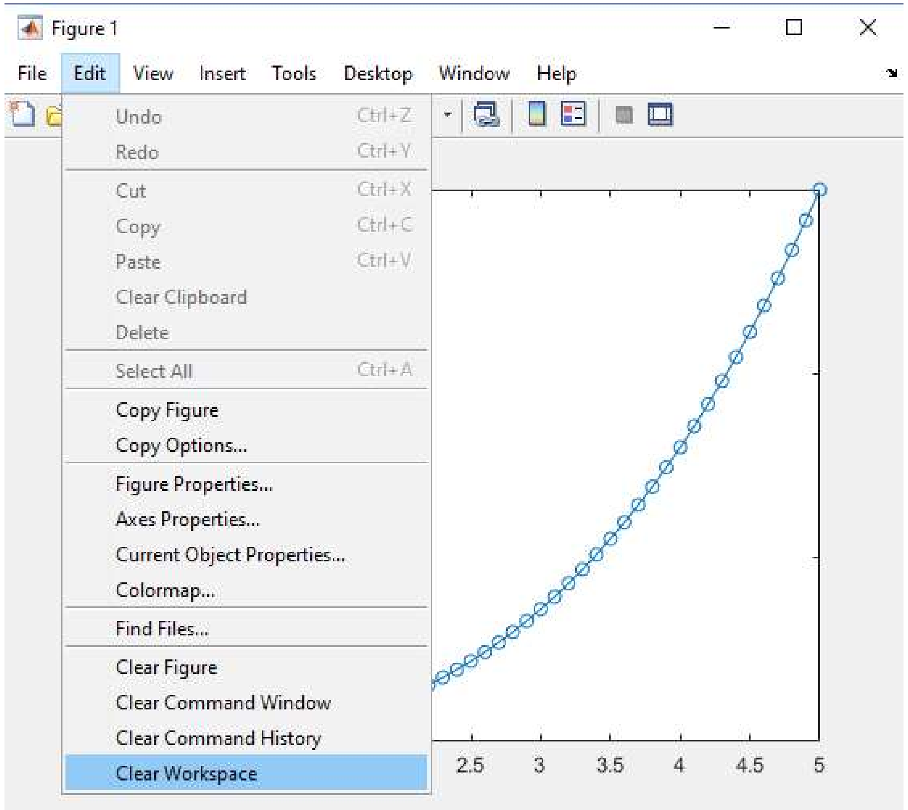


Save and Copy

- To save the graphics:
- On the Figure window, click “File” menu; Use “Save” or “Save As” options.
- The extension of graphics is “fig” .

- To transfer the graphics to another environment;
- Click “Edit” menu; Use “Copy Figure” option.
- (PS: To change the color of background, see “Copy Options” .)





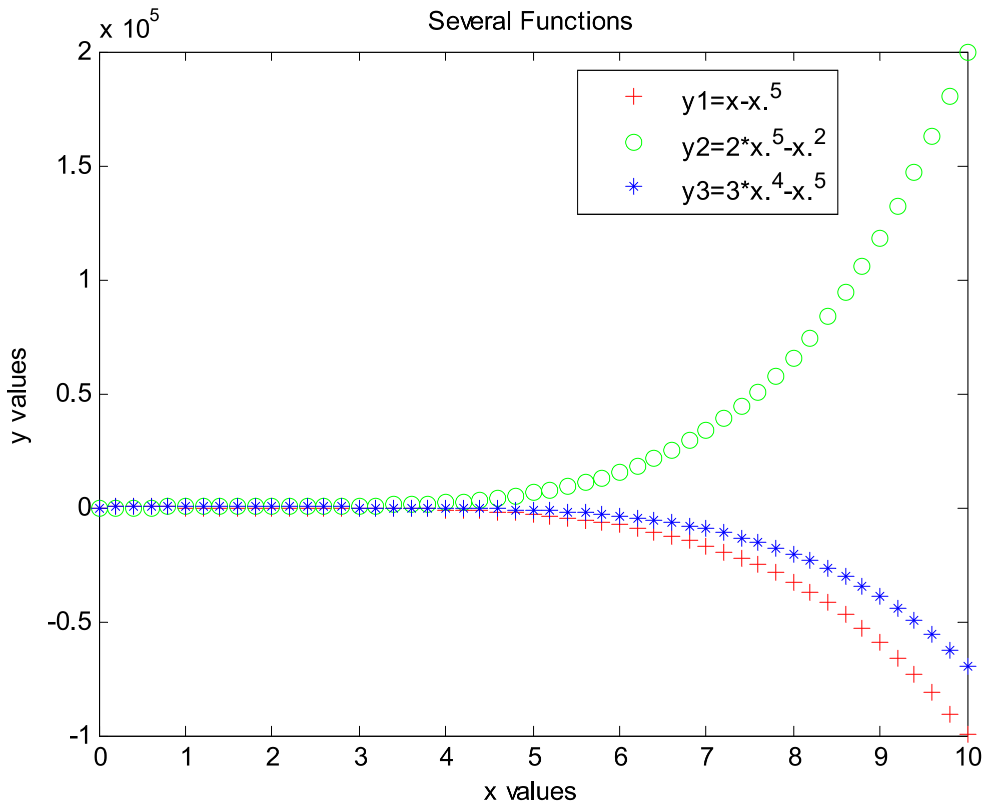
Example:

- x=[0:0.2:10]

- y1=x-x.^5; → Red +
- y2=2*x.^5-x.^2; → Green o
- y3=3*x.^4-x.^5; → Blue *

- Title: Several Functions
- X label: x values
- Y label: y values
- Legend

```
>> x=[0:0.2:10];
>> y1=x-x.^5;
>> y2=2*x.^5-x.^2;
>> y3=3*x.^4-x.^5;
>> plot(x,y1,'r+',x,y2,'go',x,y3,'b*')
>> title('Several Functions')
>> xlabel('x values')
>> ylabel('y values')
>> legend('y1','y2','y3')
```



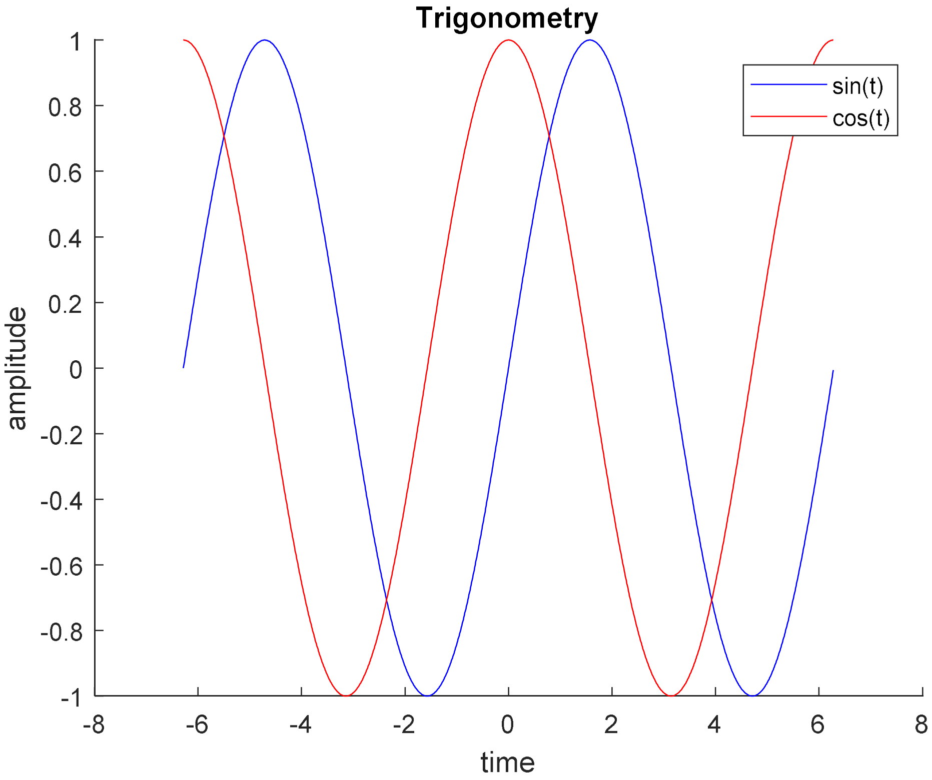
Example:

- t=[-2*pi:0.01:2*pi]

- x=sin(t);
- y=cos(t);

- Add
- Title: Trigonometry
- X label: time
- Y label: amplitude
- Legend

```
>> t=[-2*pi:0.01:2*pi];
>> x=sin(t);
>> y=cos(t);
>> hold on
>> plot(t,x,'b',t,y,'r')
>> title('Trigonometry')
>> xlabel('time')
>> ylabel('amplitude')
>> legend('sin(t)','cos(t)')
```



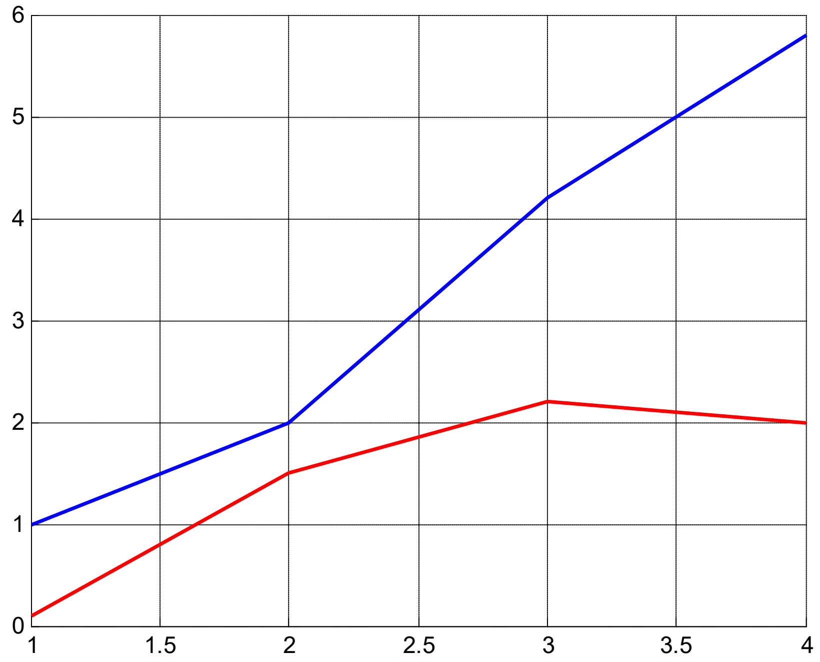
hold on – hold off
adding different graphics in a figure

For example: For two different observation data,

x=[1;2;3;4];
ya=[1;1.2;2.4;4.5]
yb=[0.5;0.8;1.8;0]

Draw the graphic corresponds to the x values

>> hold on
>> plot(ya,‘b')
>> plot(yb,'r')
>> hold off
>> grid on



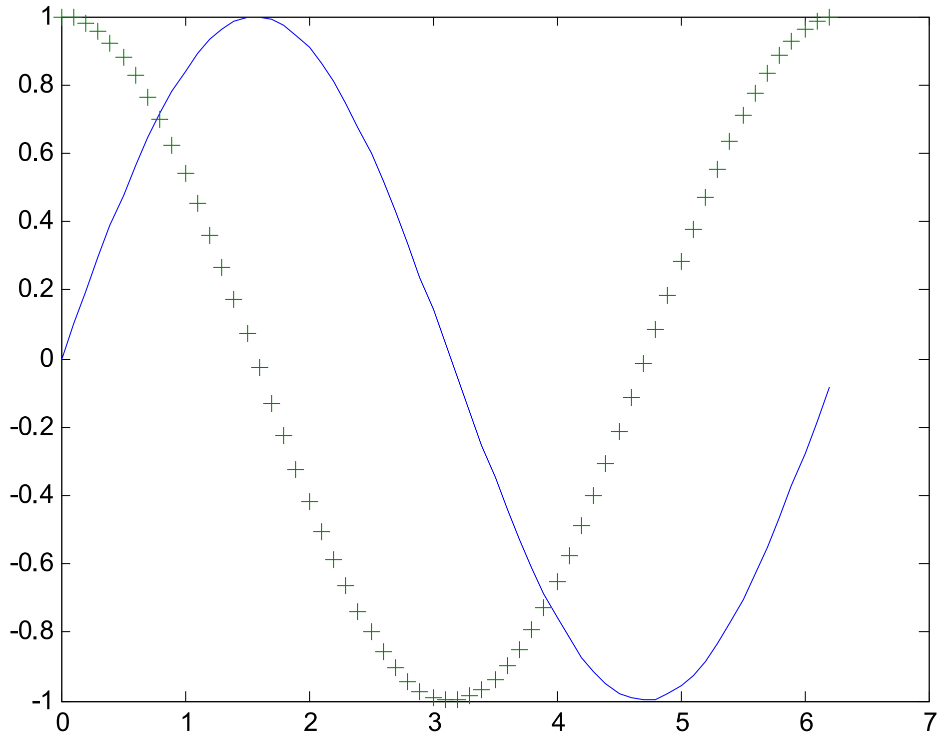
OR
plot(x,ya,x,yb)

plotyy(x,ya,x,yb)

Example

- **Plot y1 = sin(x) and y2 = cos(x) with x in [0; 2pi] on the same graph. Use a solid line for sin(x) and the symbol + for cos(x). The first step is to define a set of values for x at which the functions will be defined.**

```
>> x=[0:0.1:2*pi];
>> y1=sin(x);
>> y2=cos(x);
>> plot(x,y1,'-',x,y2,'+')
```



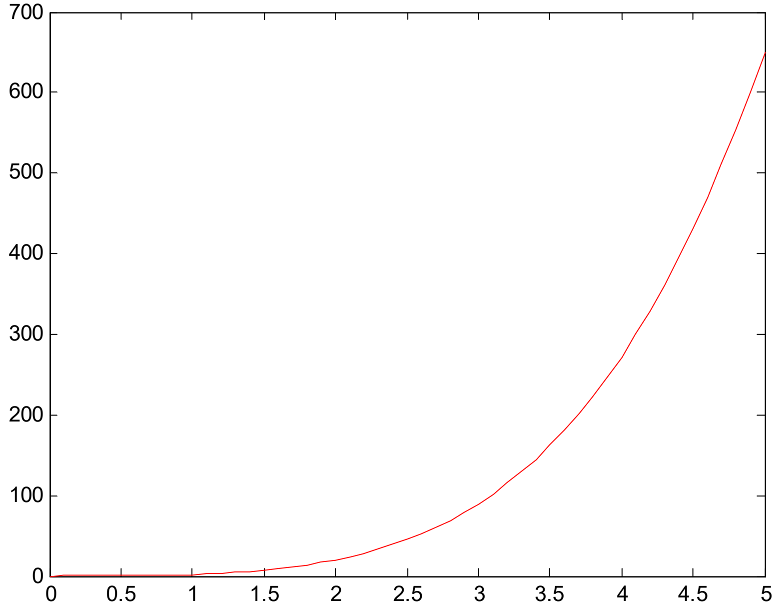
Example:

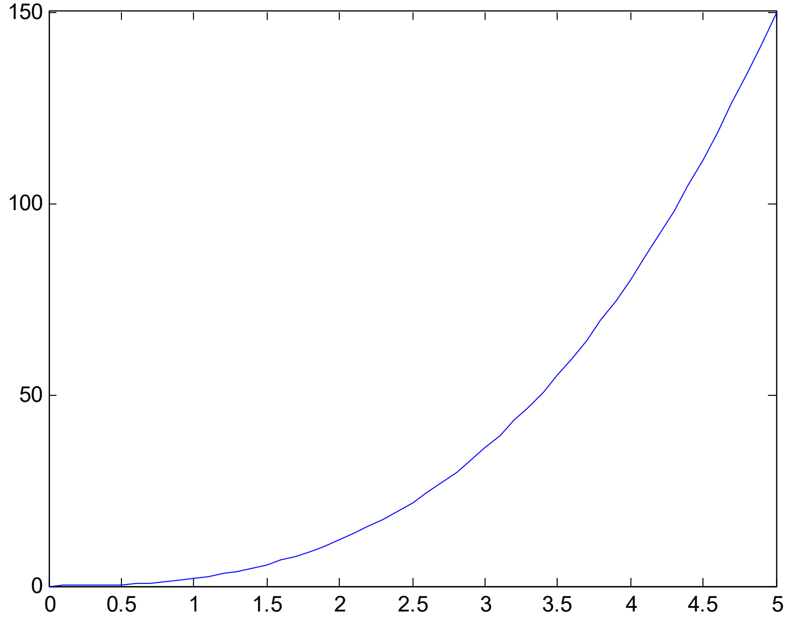
```
np=100
t=-1:2/(np*100):1;
r=(1-abs(t)).*(1+3*abs(t));
xx=r.*sin(t);
yy=r.*cos(t);
plot(xx,yy)
```

Draw multiple graphics

- **figure** function create figure window.

```
x=[0:0.1:5];
y1=x.^3+x.^2;
y2=x.^4+x.^2;
figure(1)
plot(x,y1)
figure(2)
plot(x,y2,'r')
```





Draw subplots

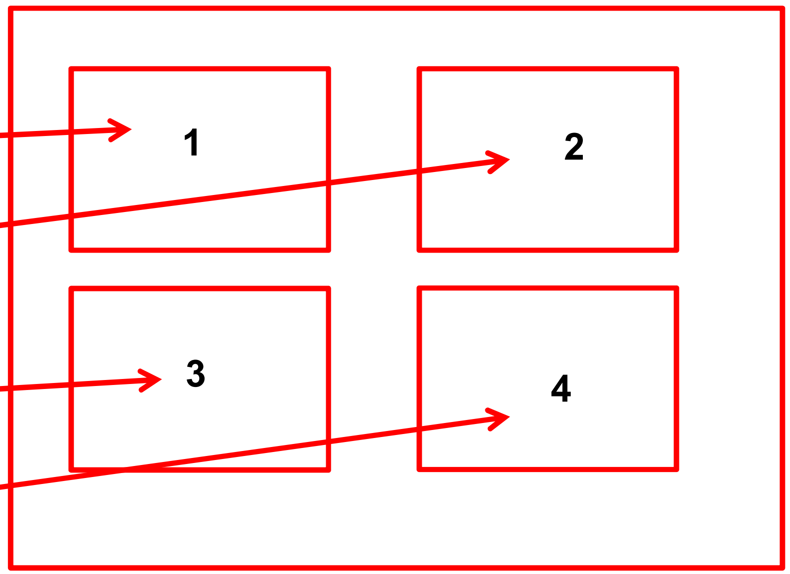
Display multiple plots in different sub regions of the same window using subplot function.

subplot(a,b,c)

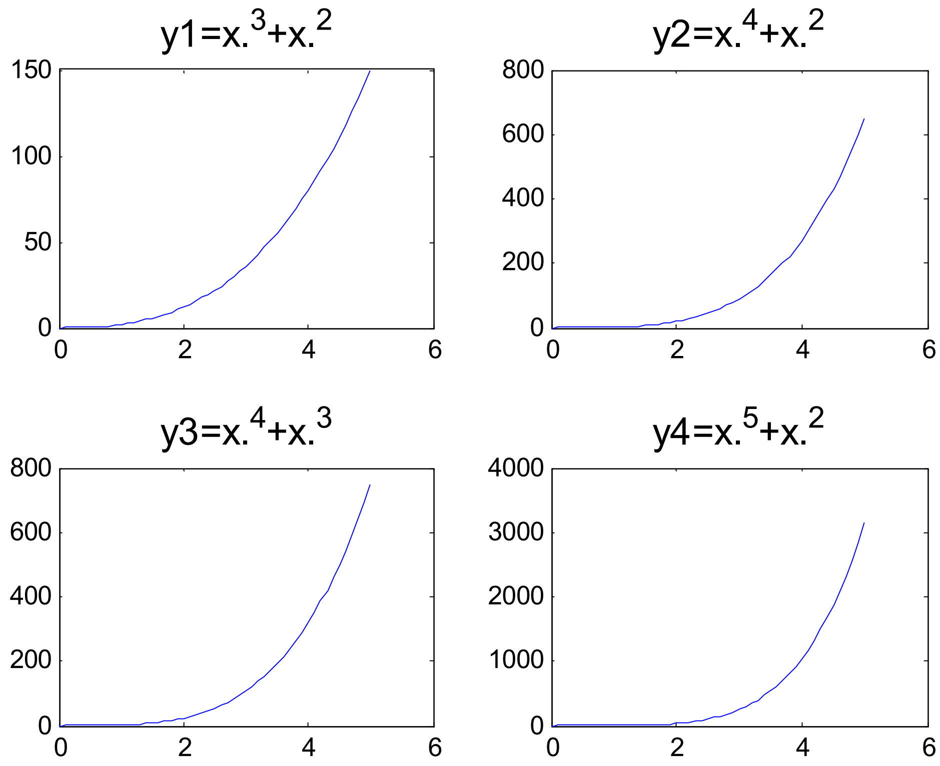
The size of graphic window: axb

The related graphic window: c

```
x=[0:0.1:5];
y1=x.^3+x.^2;
y2=x.^4+x.^2;
y3=x.^4+x.^3;
y4=x.^5+x.^2;
subplot(2,2,1)
plot(x,y1)
title('y1=x.^3+x.^2','fontsize',14)
subplot(2,2,2)
plot(x,y2)
title('y2=x.^4+x.^2','fontsize',14)
subplot(2,2,3)
plot(x,y3)
title('y3=x.^4+x.^3','fontsize',14)
subplot(2,2,4)
plot(x,y4)
title('y4=x.^5+x.^2','fontsize',14)
```



Draw subplots



DATA GENERATION

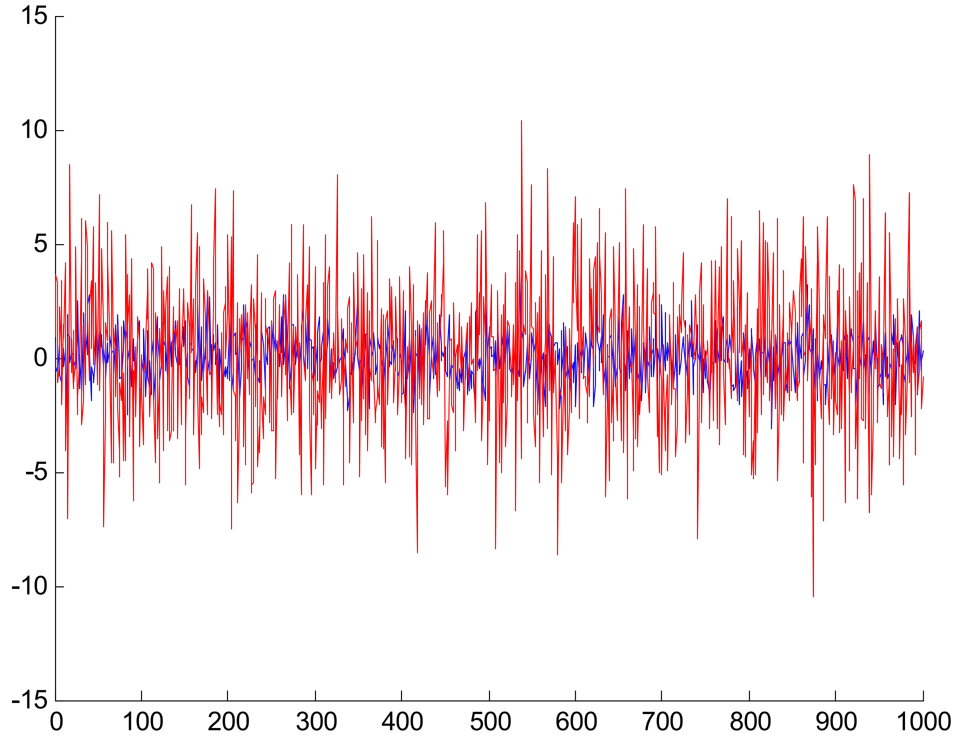
Example: Generate 2 dataset;

```
ya=randn(1000,1)
yb=randn(1000,1)*3
```

***Randn function generates data with a given standard deviation (1 and 3) and mean** 0.*

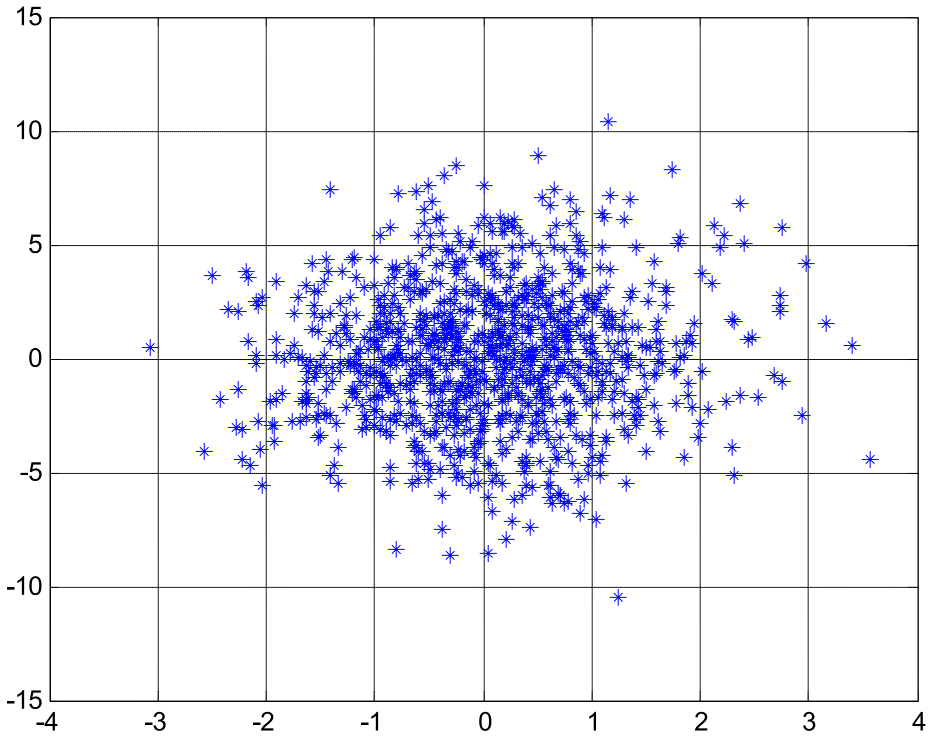
To see which dataset has lower standard deviation (more reliable), represent them on the same graphic:

>> ya=randn(1000,1);
>> yb=randn(1000,1)*3;
>> hold on
>> plot(ya)
>> plot(yb,'r')
>> hold off



- **To see the correlation between these dataset;**

```
plot(ya,yb,'*')
```

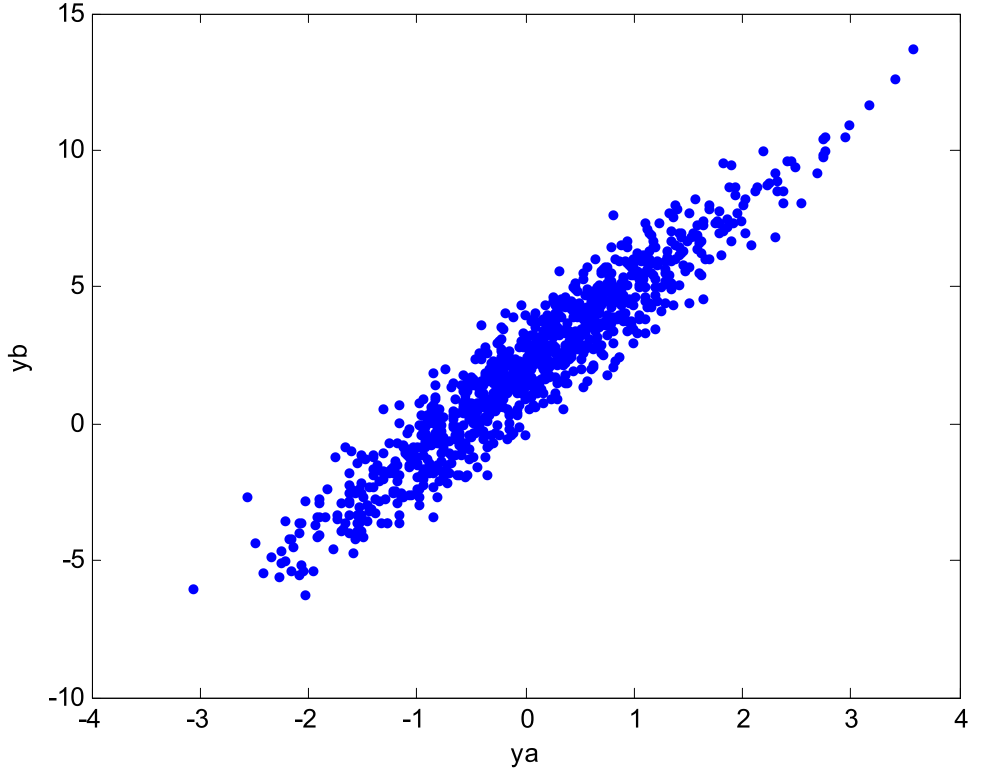


- From the related dataset, it can be seen that there are no reliable correlation. Because, as the mean value is 0 for both of them, they scatter regularly around 0.

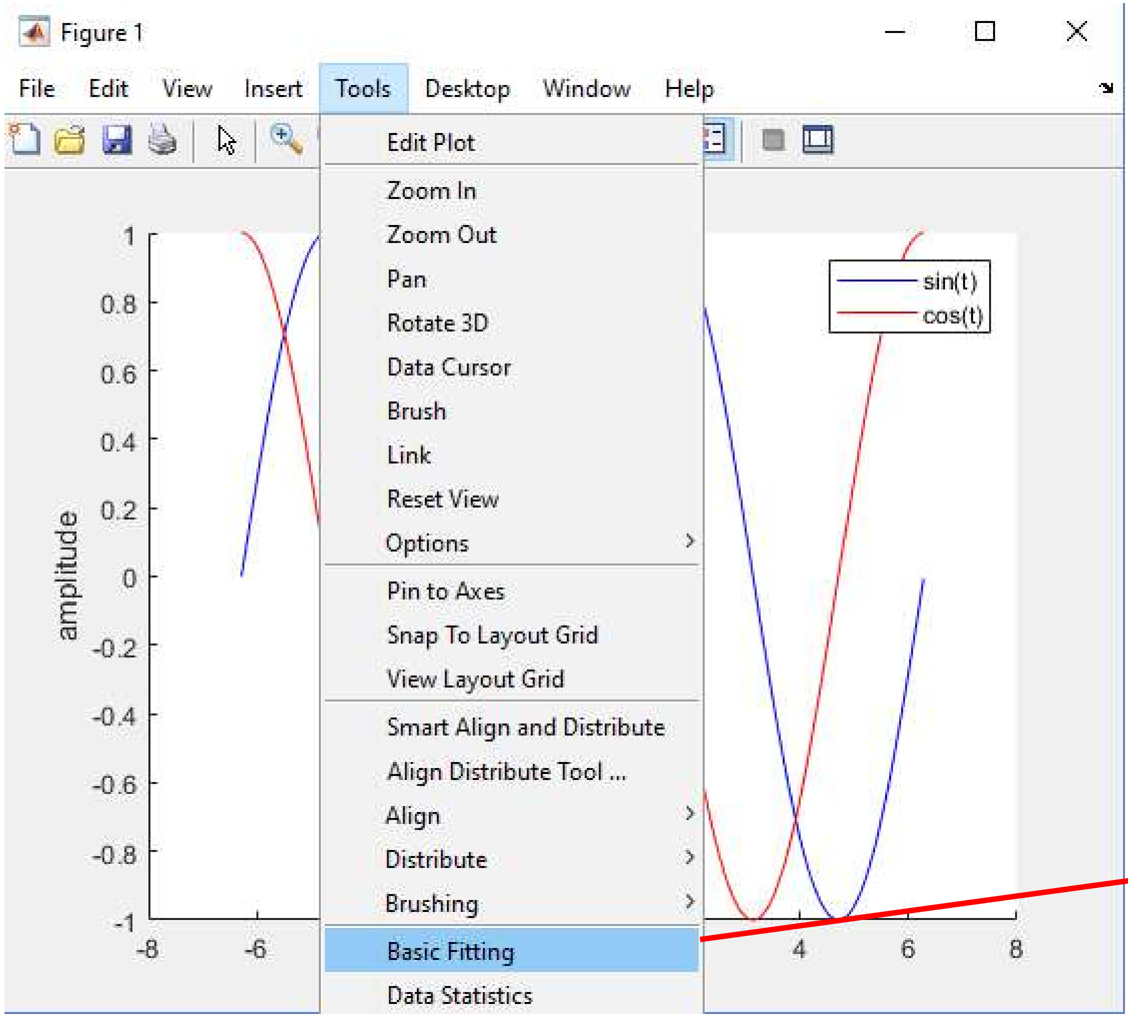
- To provide the correlation; these data should be around a straight line.

- **Let us generate the yb; according to ya values using below equation:**
- `yb=2+3*ya+randn(1000,1)*1`

```
plot(ya,yb,'.')
```

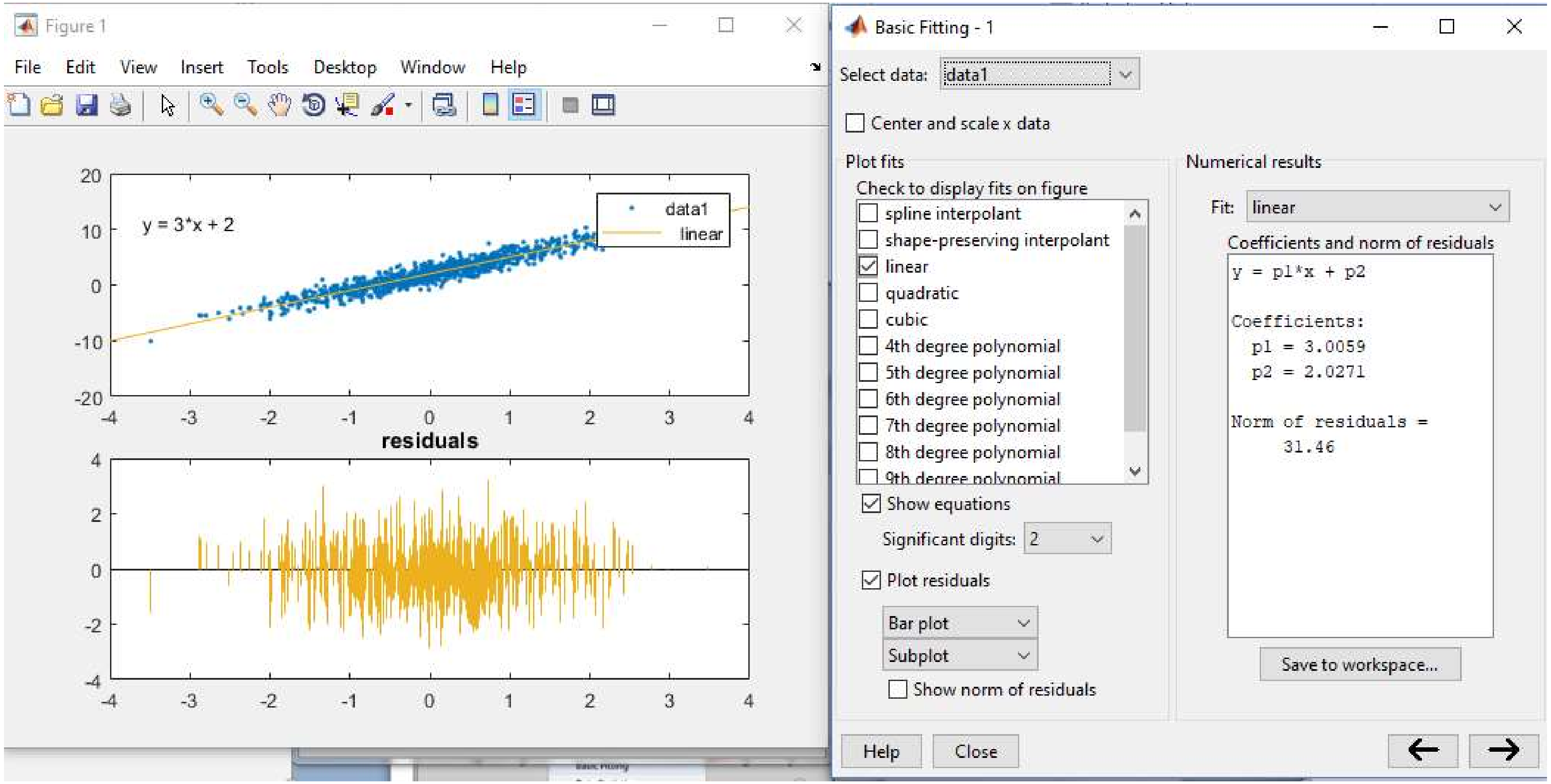


Basic Fitting Tool



Depending on x and y values; this define the function y=f(x) which fits the best.

- ya=randn(1000,1)
- yb=2+3*ya+randn(1000,1)*1
- plot(ya,yb,'.')



Example:

On the table given below, y values are given corresponding to time (x). Find the best fitting model to the observations using the function y=a+bx.

x	0	1	2	3	4	5
y	10.06	9.46	16.69	22.25	25.44	27.75

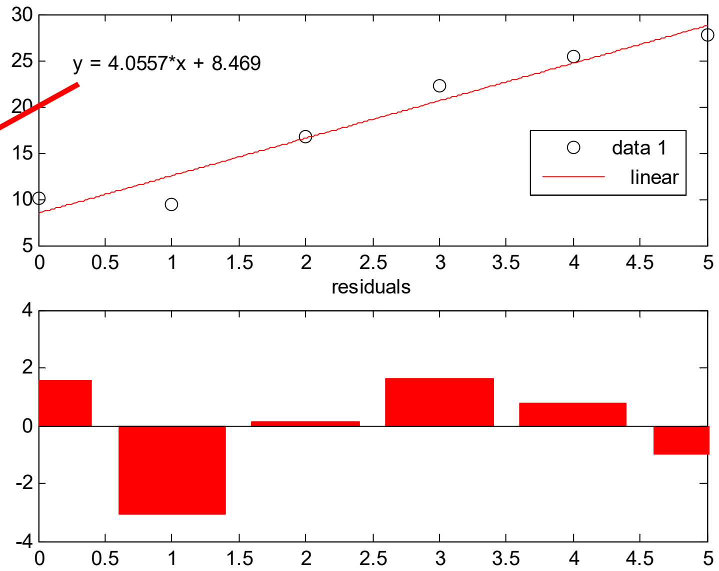
Solution:

Assign x and y values to arrays.

Draw graphics. plot(x,y,'o')

Use «Basic Fitting» window. Select “linear”, “show equation”, “plot residuals”

According to the Least Squares Method, this is the best fitting model to data



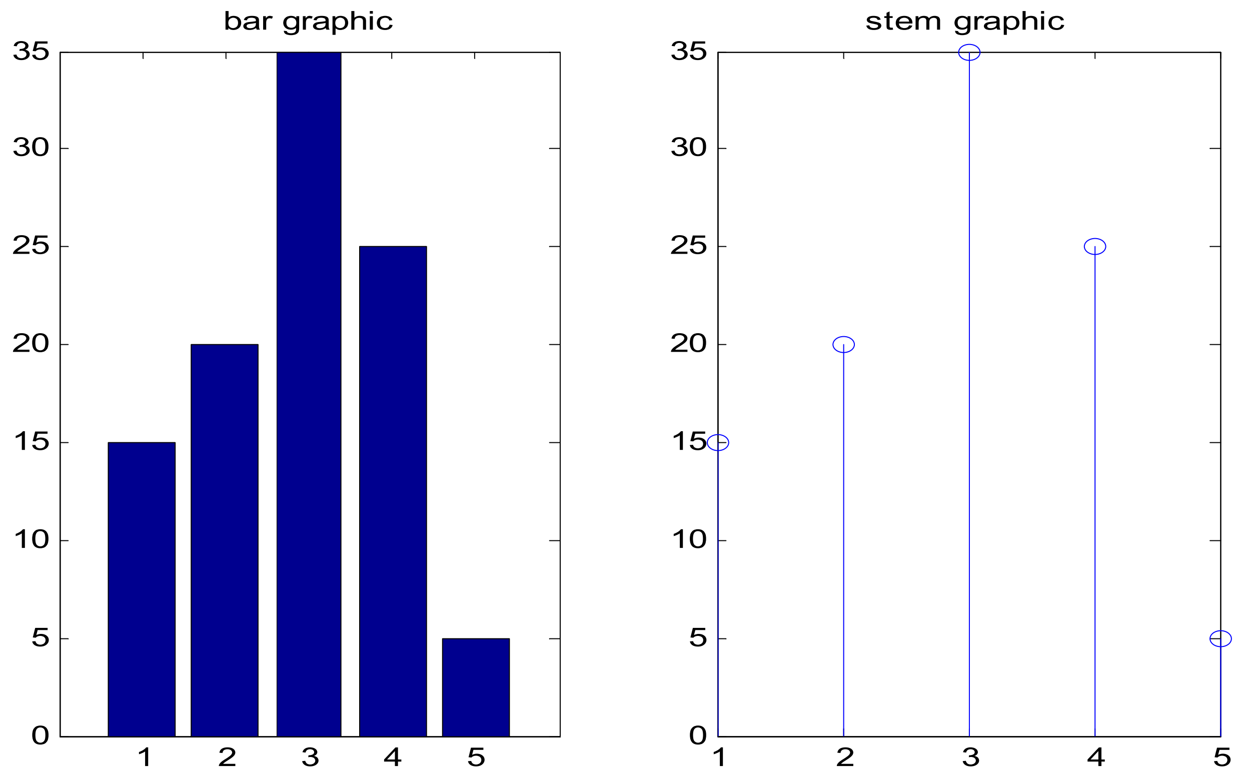
residuals

bar & stem graphics

- Example:

For `x=[15;20;25;25;5]`

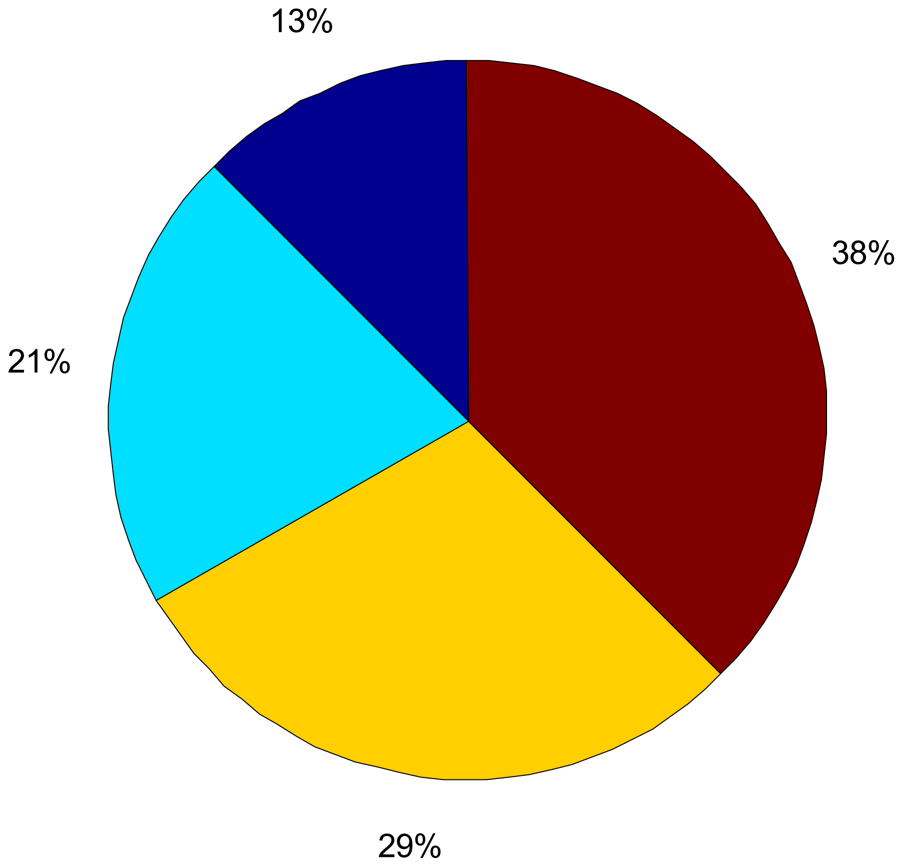
```
bar(x)
stem(x)
```



pie function

- **pie([array])** **draws a pie plot of the data in the vector X.**

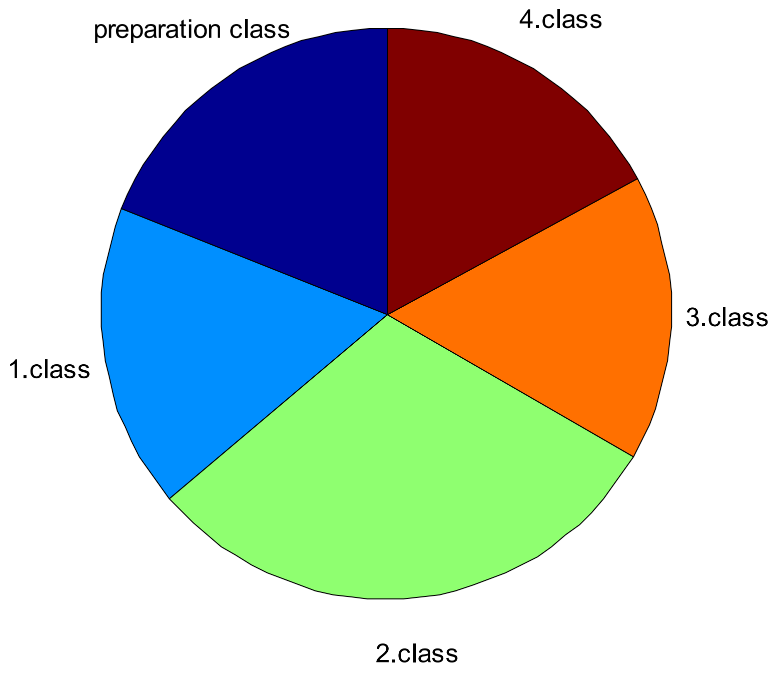
a=[15 25 35 45];
pie(a)



pie function

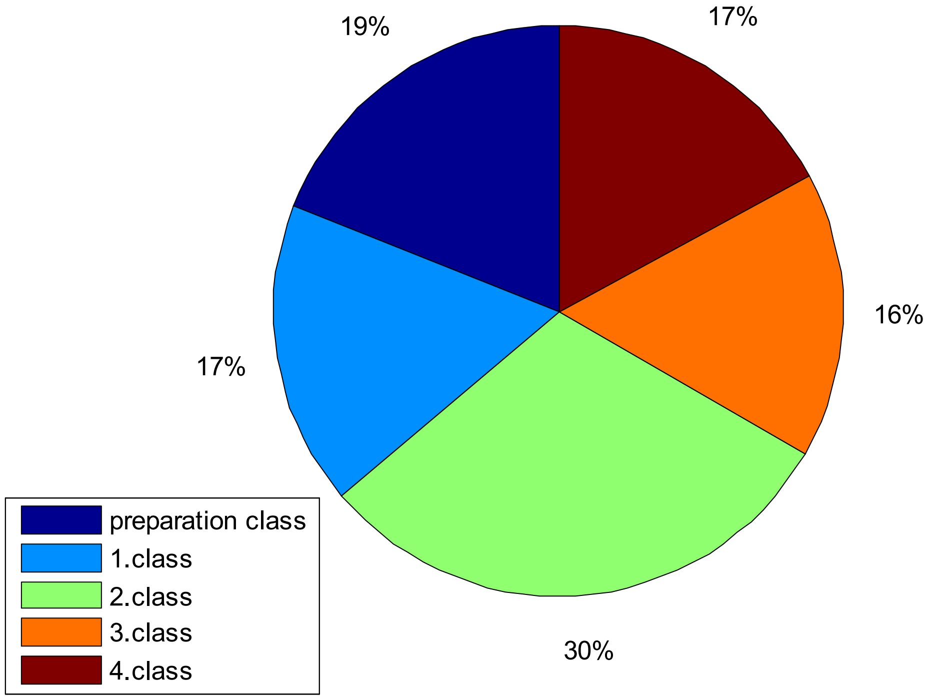
- Draw the percentage distribution of the students according to the classes.

```
clear
clc
a=[250, 225, 400, 212, 225];
b={‘preparation class','1.class', '2.class','3.class','4.class'};
pie(a,b)
```



pie function

```
clear
clc
a=[250, 225, 400, 212, 225];
pie(a)
legend(‘preparation class','1.class', '2.class','3.class','4.class');
```



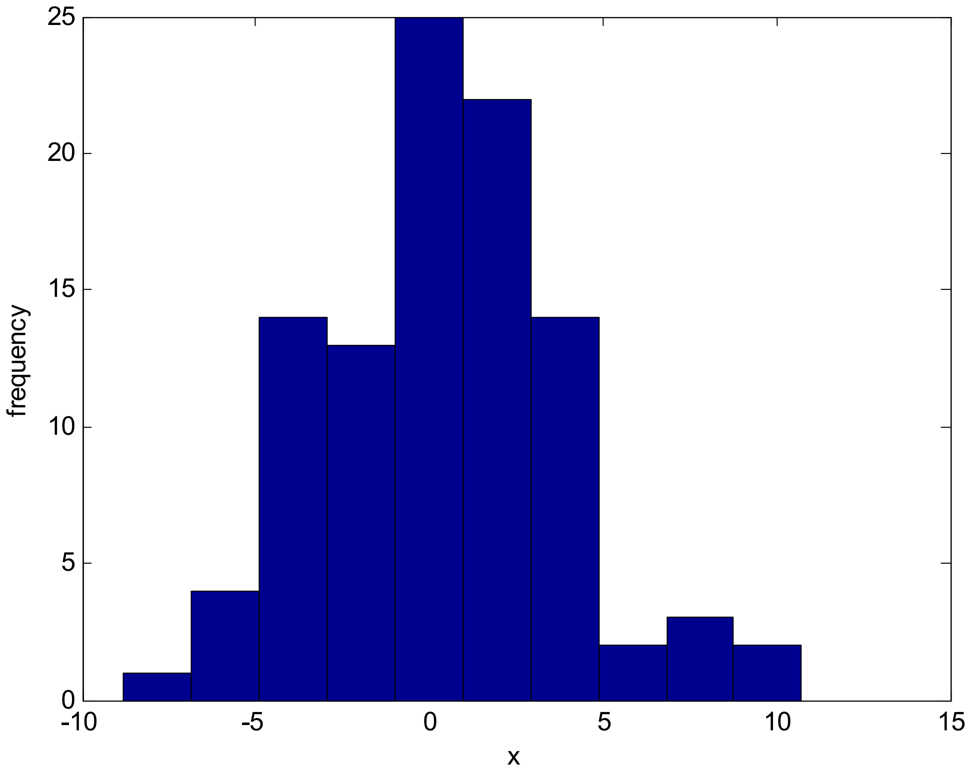
Histogram Plot

- To determine the statistical distribution of observations, the frequency values are computed and histogram graphic is plotted.

For example,

- **Generate a dataset with normal distribution using `x=randn(100,1)*3`**

- **Draw the histogram using `hist(x)`**



quiver function

- **quiver(X,Y,U,V)** plots velocity vectors as arrows with components (u,v) at the points (x,y). The matrices X,Y,U,V must all be the same size and contain corresponding position and velocity components (X and Y can also be vectors to specify a uniform grid). Quiver automatically scales the arrows to fit within the grid.

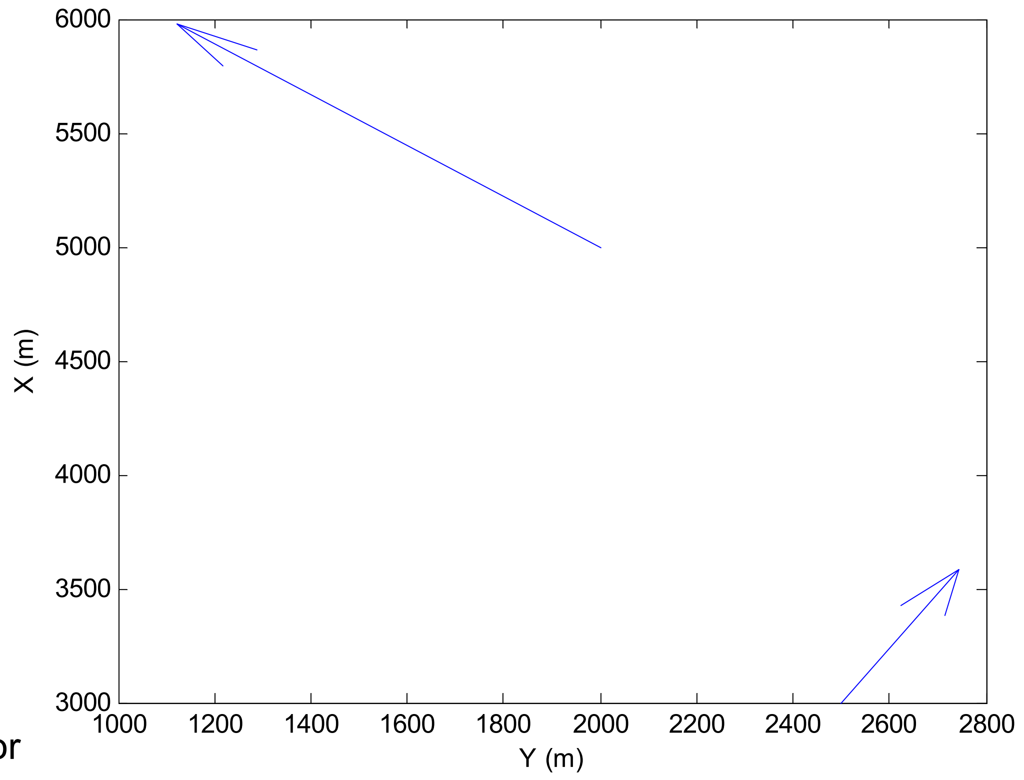
- Example: Assume that coordinates and displacements of two benchmarks are given in geodetic coordinate system.

Coordinates:

```
x=[5000;3000],
y=[2000;2500]
```

Displacements:

```
dx=[2;1.2]
dy=[-1.8;0.5]
```



- To scale vectors, add s as scale factor

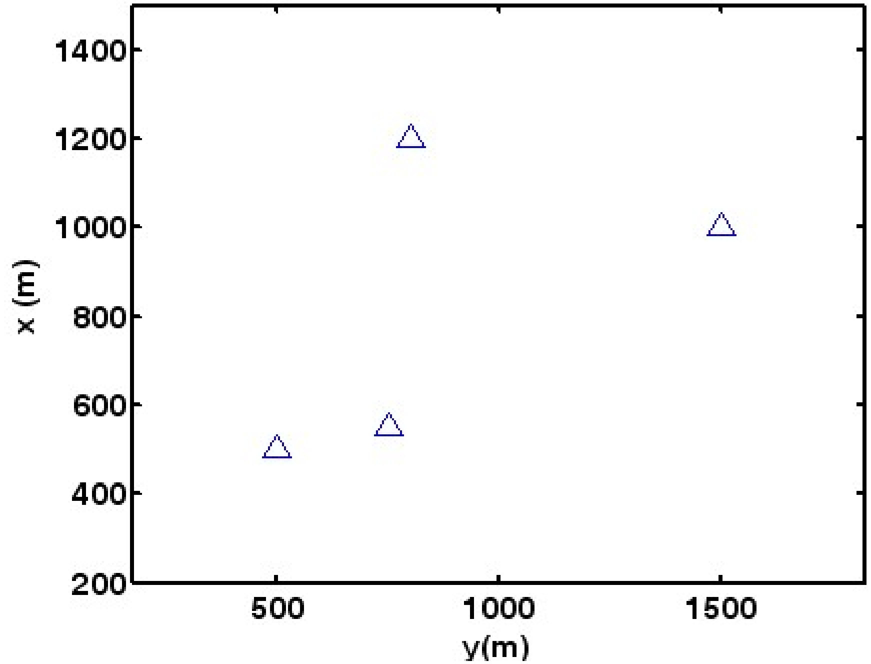
```
quiver(y,x,dy,dx,s)
```

Sketch Draw

- Draw the points according to given geodetic coordinates (x,y) with located triangle symbol.

Point	P1	P2	P3	P4
x (m)	500.00	550.00	1000.00	1200.00
y (m)	500.00	750.00	1500.00	800.00

```
plot(y,x,'^')
axis([200 1700 200 1500])
axis equal
```



- `axis([Xmin Xmax Ymin Ymax])` arrange the minimum and maximum values of axes,

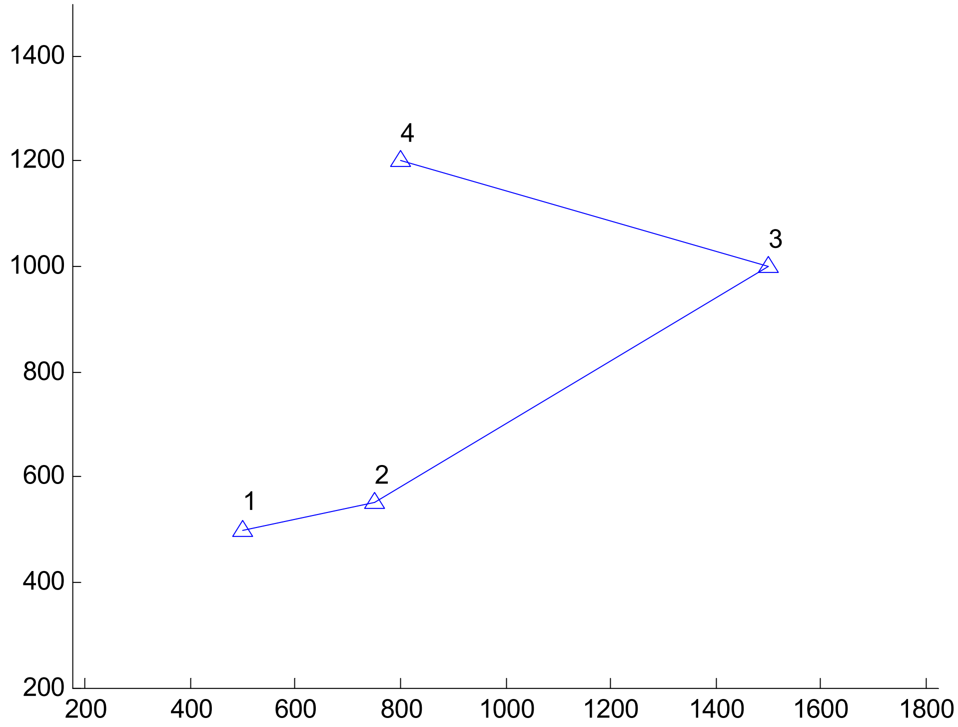
- `axis equal` Use the same length for the data units along each axis.

line Function

- Create line
- line([x1 x2],[y1 y2])

```
clear
clc
ad=[1 2 3 4];
x=[500 550 1000 1200];
y=[500 750 1500 800];
hold on
plot(y,x,'^')
axis([200 1700 200 1500])
axis equal
for i=2:length(x)
   line([y(i-1) y(i)],[x(i-1) x(i)]);
end
for i=1:length(ad)
  text(y(i),(x(i)+50),num2str(ad(i)))
end
hold off
```

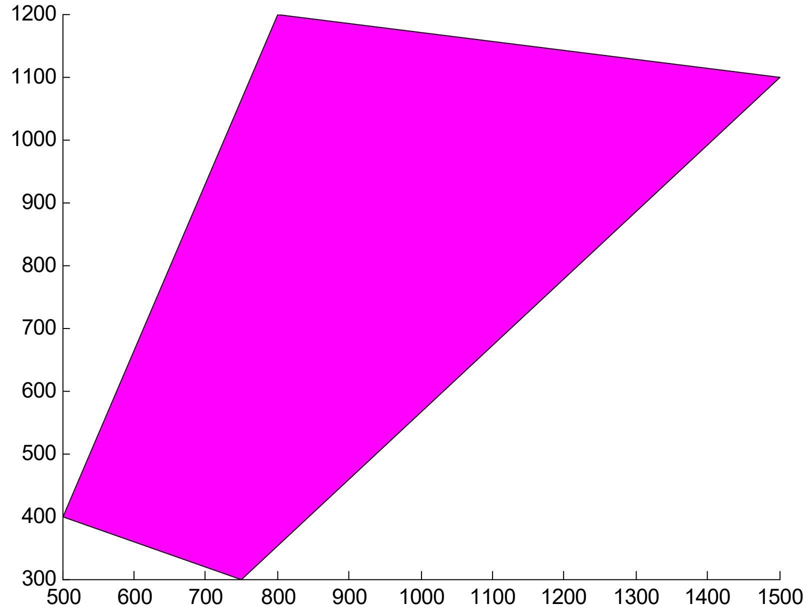
text(X,Y,'string')

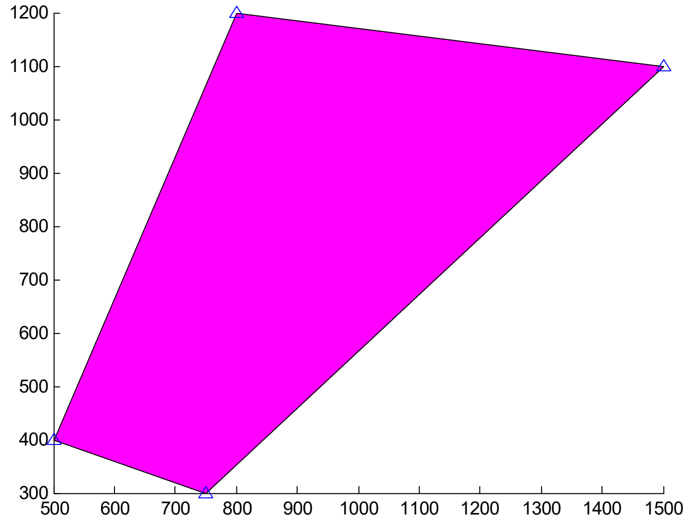


Patch Function

- Create patch
- filled 2-D polygon defined by vectors X and Y to the current axes
- patch(X,Y,C).
- C specifies the color of the face(s)

```
clear
clc
x=[400 300 1100 1200];
y=[500 750 1500 800];

patch(y,x,'m')
```





fplot Function

fplot('fonksiyon',[xmin xmax ymin ymax])

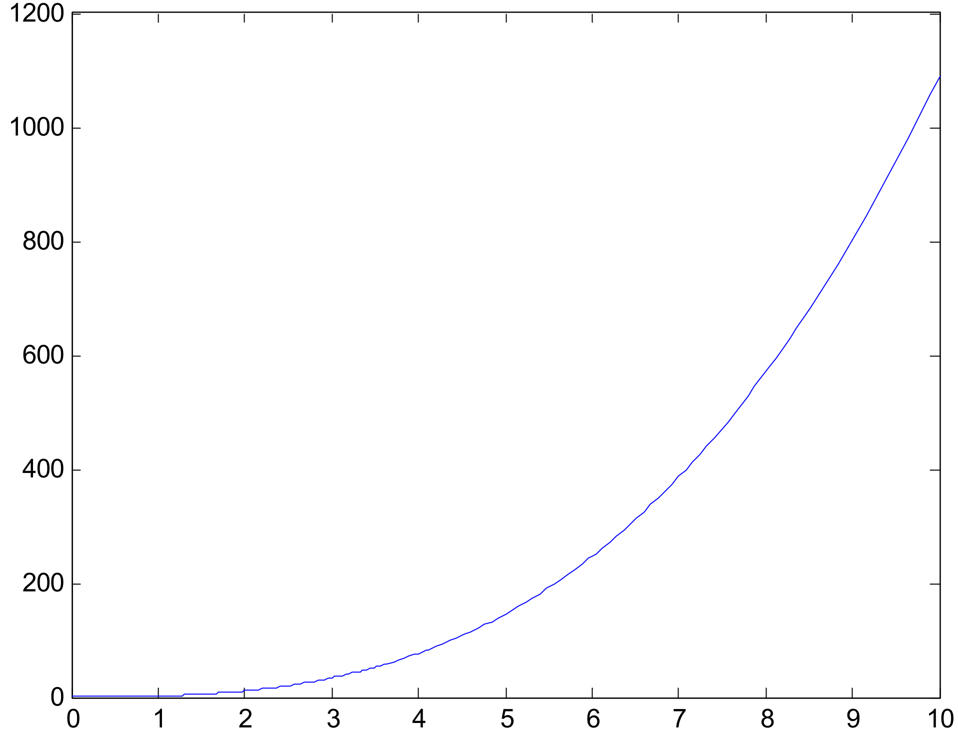
Plot function

fplot(FUN,LIMS) plots the function FUN between the x-axis limits specified by LIMS = [XMIN XMAX].

Using LIMS = [XMIN XMAX YMIN YMAX] also controls the y-axis limits. FUN(x) must return a row vector for each element of vector x

fplot('x^3+x^2-x+1',[0 10])

To define x axes for limitation is enough



3D Graphics

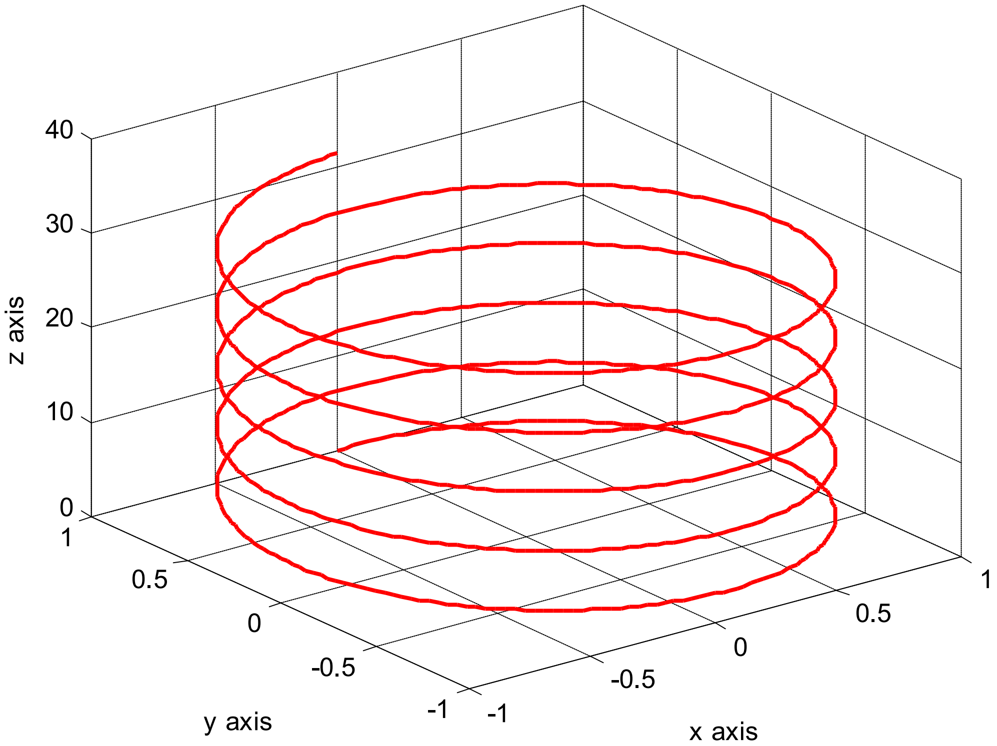
plot3
meshgrid
mesh
surf
contour

plot3 function

Plot lines and points in 3-D space.

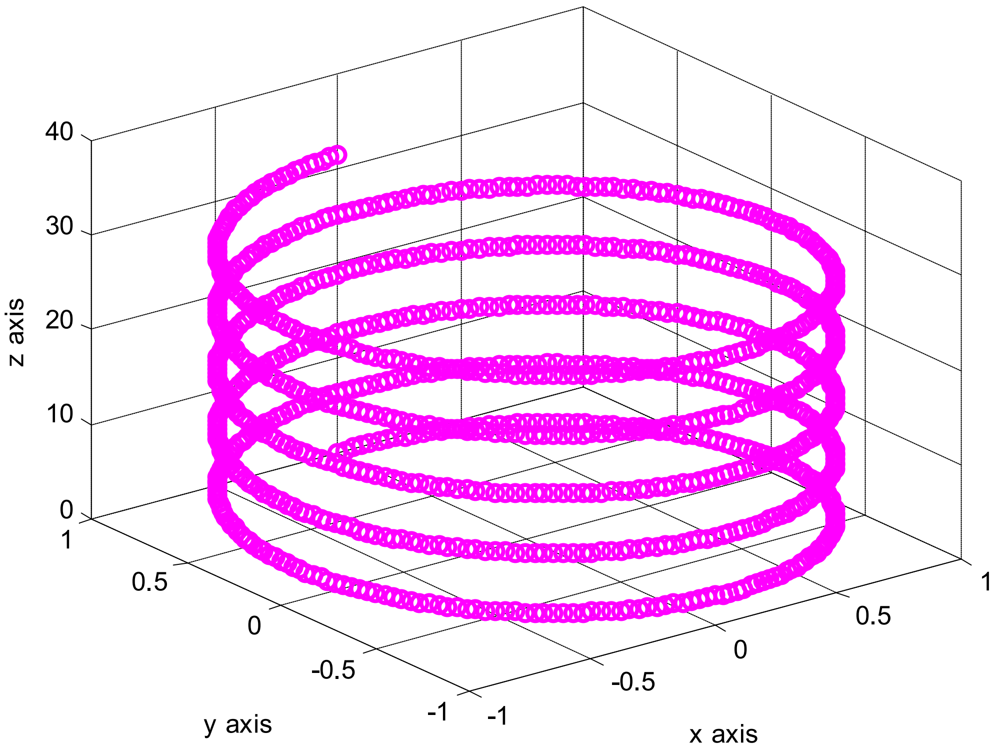
plot3(x,y,z), where x, y and z are three vectors of the *same length*, plots a line in 3-space through the points whose coordinates are the elements of x, y and z.

```
t = 0:pi/100:10*pi;
plot3(sin(t),cos(t),t,'r','LineWidth',2);
grid on
xlabel('x axis');
ylabel('y axis');
zlabel('z axis');
```



scatter3 function

```
t = 0:pi/100:10*pi;
scatter3(sin(t),cos(t),t, 'm','LineWidth',2);
grid on
xlabel('x axis');
ylabel('y axis');
zlabel('z axis');
```

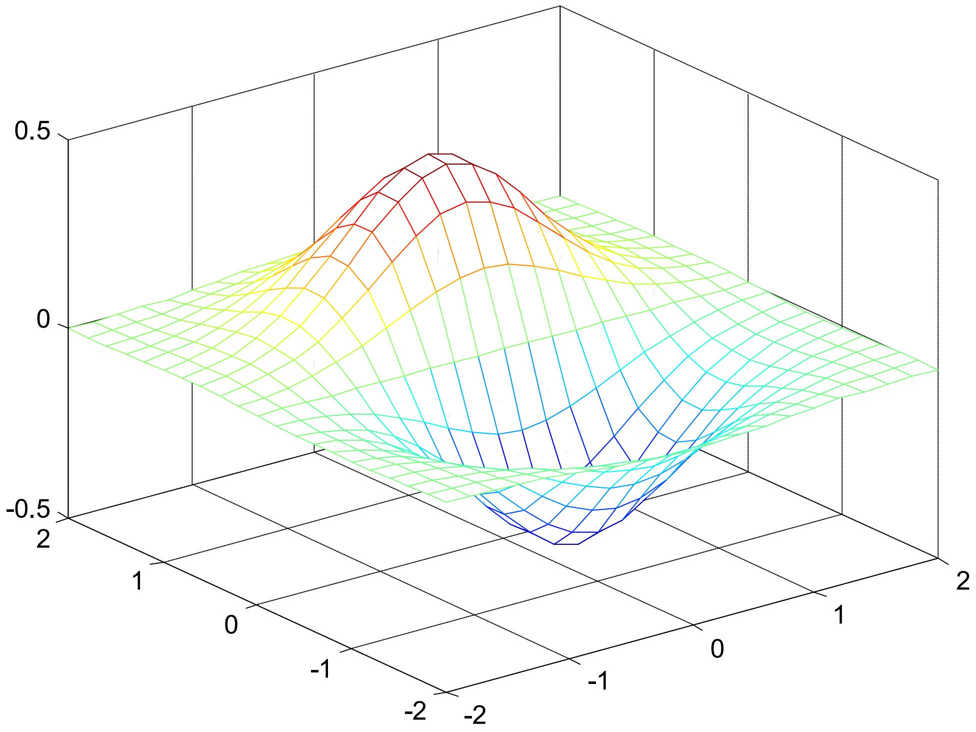


meshgrid & mesh functions

- meshgrid : replicates the grid vectors x and y to produce the coordinates of a rectangular grid (X, Y).
  [X,Y]=meshgrid(x,y)
- mesh: plots the colored parametric mesh defined by four matrix arguments.
  mesh(X,Y,Z,C)

For -2 < x < 2, -2 < y < 2

```
[X,Y] = meshgrid(-2:0.2:2, -2:0.2:2);
Z = Y .* exp(-X.^2 - Y.^2);
mesh(X,Y,Z)
```



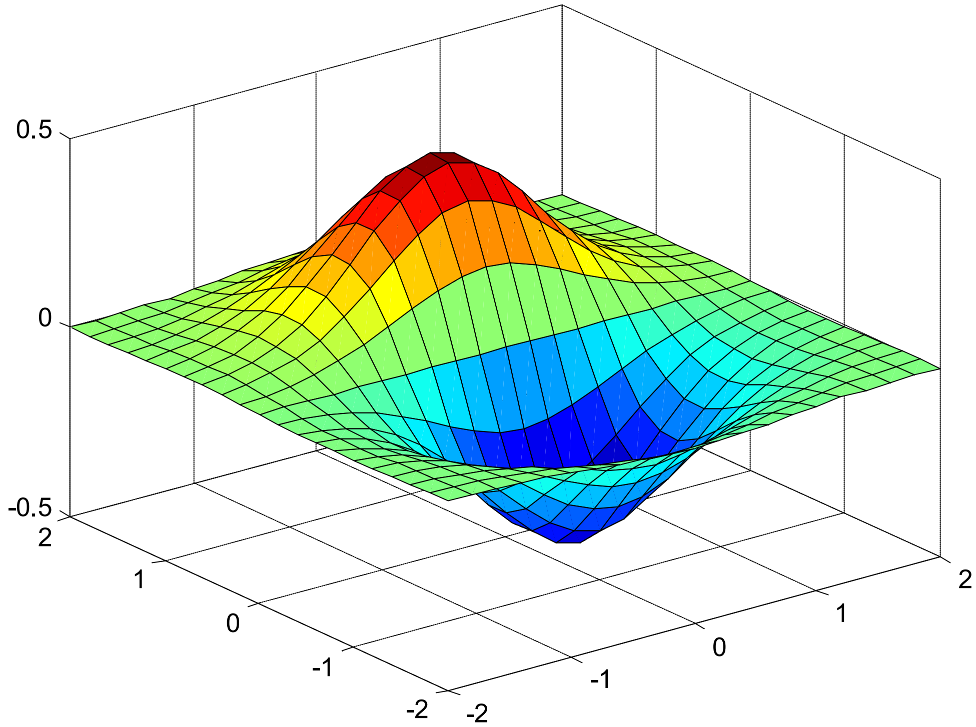
`meshc` and `meshz` functions!!!

surf function

- Surf: 3-D colored surface
  surf(X,Y,Z) or surf(Z)

For -2 < x < 2, -2 < y < 2

```
[X,Y] = meshgrid(-2:0.2:2, -2:0.2:2);
Z = Y .* exp(-X.^2 - Y.^2);
surf(X,Y,Z)
```

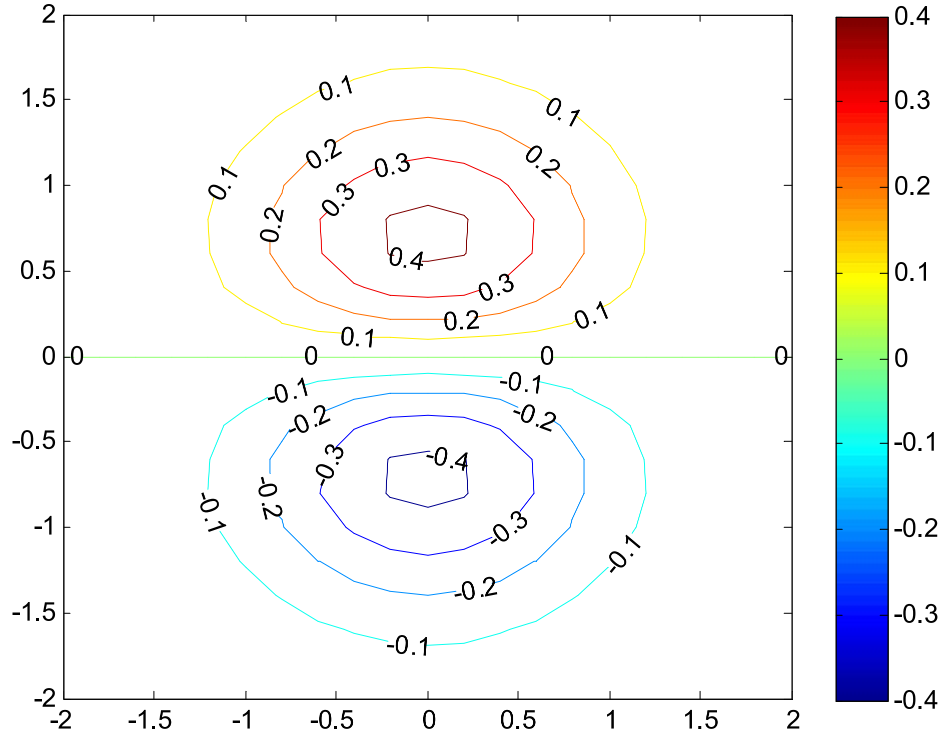


`surfl` **and** `surfc` **functions !!**

contour function

- contour(Z) is a contour plot of matrix Z treating the values in Z as heights above a plane.
- [C, H] = contour(...) returns contour matrix C as described in CONTOURC and a handle H to a contourgroup object. This handle can be used as input to CLABEL.
- clabel(C,H)

```
[X,Y] = meshgrid(-2:0.2:2, -2:0.2:2);
Z = Y .* exp(-X.^2 - Y.^2);
[C,H]=contour(X,Y,Z);
clabel(C,H)
colorbar
```



MATLAB- write & read

diary

Save Command Window text to file

Syntax

```
diary
diary('filename')
diary off
diary on
diary filename
```

- Write data to text file ➔ scan
- Read data from text file ➔ input data

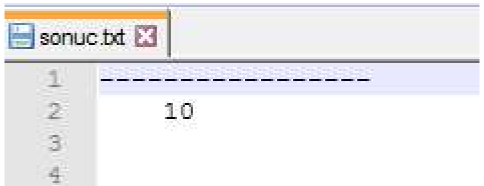
The basic function for writing data to the text file: `diary`

- The **diary** function creates a log of keyboard input and the resulting text output, with some exceptions.
- The output of **diary** is an ASCII file, suitable for searching in, printing, inclusion in most reports and other documents.
- If you do not specify **filename**, the MATLAB® software creates a file named **diary** in the current folder

MATLAB- write & read

```
a=10;
diary sonuc.txt
    disp('-----------------')
    disp(a)
diary end
```

- All of the info that will be written on command window between two diary commands, will be on sonuc.txt file. Matlab creates a file named **diary** in the current folder.
- Different file names and extensions instead of sonuc.txt can be used.
- If the file has already been created, then the outputs would be added after the text written on file.



MATLAB- write & read

- fopen: Open file, or obtain information about open files
- fprintf: Write data to text file
- fclose: Close one or all open files

While using these functions, <u>No need to show</u> the text to be written on ‘command window’.

Syntax

- fileID = fopen(filename)
- fprintf(fileID,formatSpec,A1,...,An)
- fclose(fileID)

For instance: Assume that a side, namely ‘a’ is computed by a program. For writing the value ‘a’ computed by this program on kenar.txt file; the following codes can be written:

```
a=150.0234234;
fid=fopen('kenar.txt','w');
fprintf(fid,'kenar uzunluğu=%1.4f',a);
fclose(fid);
```

w refers that it will be written on this file.

permission — File access type

'r' (default) | 'w' | 'a' | 'r+' | 'w+' | 'a+' | ...	
'r'	Open file for reading.
'w'	Open or create new file for writing. Discard existing contents, if any.
'a'	Open or create new file for writing. Append data to the end of the file.
'r+'	Open file for reading and writing.
'w+'	Open or create new file for reading and writing. Discard existing contents, if any.
'a+'	Open or create new file for reading and writing. Append data to the end of the file.

Examples

Ex.1: Write a program that writes the following matrix
a=[3.12356 4.12456 1;5.8463 6.45111 2;4 5 6]
with 4 digits of its elements on **mat.out** file.

```
a=[3.12356 4.12456 1;5.8463 6.45111 2;4 5 6]
fid = fopen('mat.out','w');
fprintf(fid,'%1.4f%10.4f%10.4f\n',a);
fclose(fid);
```

mat.out

```
3.1236    5.8463    4.0000
4.1246    6.4511    5.0000
1.0000    2.0000    6.0000
```

Ex.2: Write these two variables; side=1500.123 m & azimuth=103.3367 grad, to the **result.out** file one under the other.

```
side=1500.123;
azimuth=103.3367;
fid=fopen('result.out','w');
fprintf(fid,'side=%1.3f m\n',side);
fprintf(fid,'azimuth=%1.4f grad',azimuth);
fclose(fid)
```

result.out

```
side=1500.123 m
azimuth=103.3367 grad
```

Examples

Ex.3: Write a program that makes conversion between Fahrenheit and centigrade units for a given interval and writes the results on a file with extension '.txt'.

<u>TIP</u>: Fahrenheit=1.8*centigrade+32;

```
Tstart=input('Enter the initial temperature:');
Tend=input('Enter the final temperature:');
nTemp=input('How many values are required between initial and final temperatures:');

centigrade=linspace(Tstart,Tend,nTemp);

fahrenheit=1.8*centigrade+32;

fid=fopen('temp.txt','w+');
fprintf(fid,'Temperature Conversion Chart\n');
fprintf(fid,'--------------------------\n');
fprintf(fid,'Centigrade    Fahrenheit\n');
for k=1:nTemp
    fprintf(fid, '%f %f \n', centigrade(k), fahrenheit(k) );
end
fclose(fid);
```

temp.txt

```
Temperature Conversion Chart
--------------------------
Centigrade   Fahrenheit
5.000000 41.000000
7.222222 45.000000
9.444444 49.000000
11.666667 53.000000
13.888889 57.000000
16.111111 61.000000
18.333333 65.000000
20.555556 69.000000
22.777778 73.000000
25.000000 77.000000
```

MATLAB- write & read

fscanf: Read data from text file

A = fscanf(fileID,formatSpec)
A = fscanf(fileID,formatSpec,sizeA)
[A,count] = fscanf(___)

Exp. Use the same matrices produced at the previous example (mat.out) and read it in Matlab.

```
fid=fopen('mat.out','r+');
[dizi,sayi]=fscanf(fid,'%f',inf)
```

```
dizi =

   3.1236
   5.8463
   4.0000
   4.1246
   6.4511
   5.0000
   1.0000
   2.0000
   6.0000


sayi =

    9
```

```
fid=fopen('mat.out','r+');
[dizi,sayi]=fscanf(fid,'%f',[3 3])
```

```
dizi =


   3.1236    4.1246    1.0000
   5.8463    6.4511    2.0000
   4.0000    5.0000    6.0000


sayi =

    9
```

MATLAB- write & read

- `textread`: Read data from text file; write to multiple outputs
- **For example:** Read the data on below file, namely koordinat.txt.

```
P1   1000.1234 1300.23423
P2   1300.5673 1450.98563
P3   2000.1500 2000.11000
P4   3500.3100 1000.12000
```

Station Name **x coordinate** **y coordinate**

- To do this;

```
[nokta,x,y]=textread('koordinat.txt','%s%f%f')
```

The above function is used.
`nokta` is assigned as a cell containing station names;
`x` includes x coordinates' vector and,
`y` includes y coordinates' vector.

```
[a, b, c,…]=textread('dosya_adi', 'format')
```

```
nokta =

4×1 cell array

    'P1'
    'P2'
    'P3'
    'P4'

x =

  1000.1234
  1300.5673
    2000.15
    3500.31

y =

 1300.23423
 1450.98563
    2000.11
    1000.12
```

MATLAB- write & read

- Example: Use the below file (koordinat.txt) containing coordinates of the stations and assign the variables to the names of ‘nokta’, ‘x’, ‘y’, by using textread function.

```
Nirengi koordinatları
NN       x(m)        y(m)
 P1   1000.1234 1300.23423
 P2   1300.5673 1450.98563
 P3   2000.1500 2000.11000
 P4   3500.3100 1000.12000
```

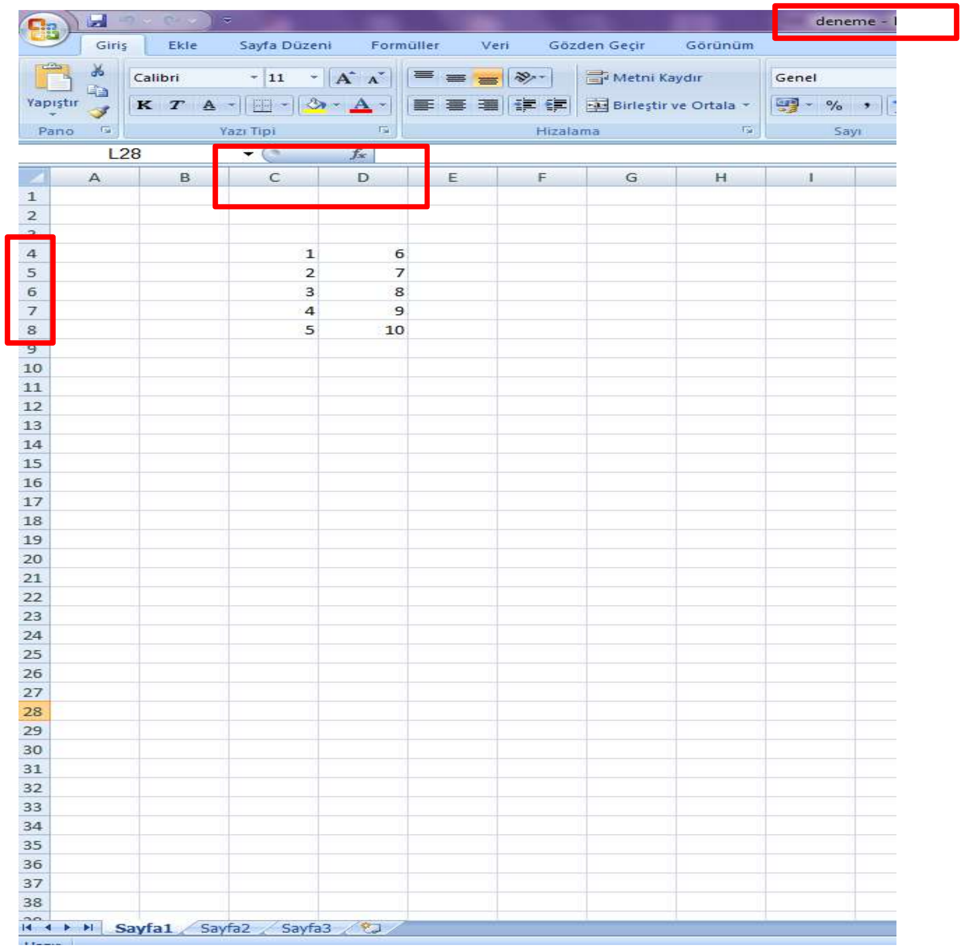
```
[nokta,x,y]=textread('koordinat.txt','%s%f%f','headerlines',2)
```

`'headerlines'` Ignores the specified number of lines at the beginning of the file.

✓ In **koordinat.txt** file, the first two rows are ignored.

MATLAB- write & read

- xlsread: Read Microsoft Excel spreadsheet file



num = xlsread('filename', sheet, 'range')

A = xlsread('deneme.xlsx', 1, 'C4:D7')

```
A =

    1    6
    2    7
    3    8
    4    9
```

FUNCTION

Declare function name, inputs, and outputs

- function [y1,...,yN] = myfun(x1,...,xM) declares a function named **myfun** that accepts inputs **x1,...,xM** and returns outputs **y1,...,yN**.
- This declaration statement must be the first executable line of the function.
- Valid function names begin with an alphabetic character, and can contain letters, numbers, or underscores.
- Function is stored in m-file and this file uses the same name with function.
- The advantages of use of function are;

✓ **Avoid code repetition if loops are required. (e.g., assume that there is a function for computing the azimuth angle namely, `azimuth`; in the related part of the program, if this function is defined, the function will compute the azimuth angles for given two points just coding `azimuth(X1,Y1,X2,Y2)`**

✓ **the variables given in functions are local variables, which means that they are not stored as global variables in workspace as the other program types.**

MATLAB/Function Files

- Example: Write a function, namely ‘kenar’, to compute the horizontal distance between given two points with x and y coordinates.

kenar.m

```
function S=kenar(X1,Y1,X2,Y2)

%this function computes horizontal distance
%between two points
%with given x and y coordinates

dx=X2-X1;
dy=Y2-Y1;
S=sqrt(dx^2+dy^2);
end
```

- The only difference for function from the other program is that;

  ```
  function output=func_name(input)
  ```

  It is start with the above statement, and is completed with `end`

- The next row after the function command is the explanation of the function (prompt).

- The file name and the function name should be the same.

MATLAB/Function Files

Example: Write a function, namely aci_kenar; to compute both azimuth angle and horizontal distance, for given any two points.

aci_kenar.m

```
function [ a,S ] = aci_kenar( X1,Y1,X2,Y2)
%[a,S]=aci_kenar(X1,Y1,X2,Y2)
%This function computes azimuth angle of (1-2)==>a
%and horizontal distance between Point1 and Point2==>S

 DX=X2-X1;DY=Y2-Y1;

if (DX~=0)&(DY~=0),a=atan(DY/DX);a=a*200/pi;
  if (DX>0)&(DY>0),a=a;end
  if (DX<0)&(DY>0),a=a+200;end
  if (DX<0)&(DY<0),a=a+200;end
  if (DX>0)&(DY<0),a=a+400;end
end

if (DX==0)&(DY>0),a=100;end
if (DX==0)&(DY<0),a=300;end
if (DX>=0)&(DY==0),a=0;end
if (DX<0)&(DY==0),a=200;end

S=sqrt(DX^2+DY^2);

 end
```

- There can be several outputs of the function.
- In this example, there are two, a and S.
- a, is the azimuth angle, S is the distance.
- [a,S]=aci_kenar(1500,5210,4521,5842)