- Ankara Üniversitesi BLM bölümü

BLM433-1 Modelleme Yaklaşım Kesme hataları

BLM433-1

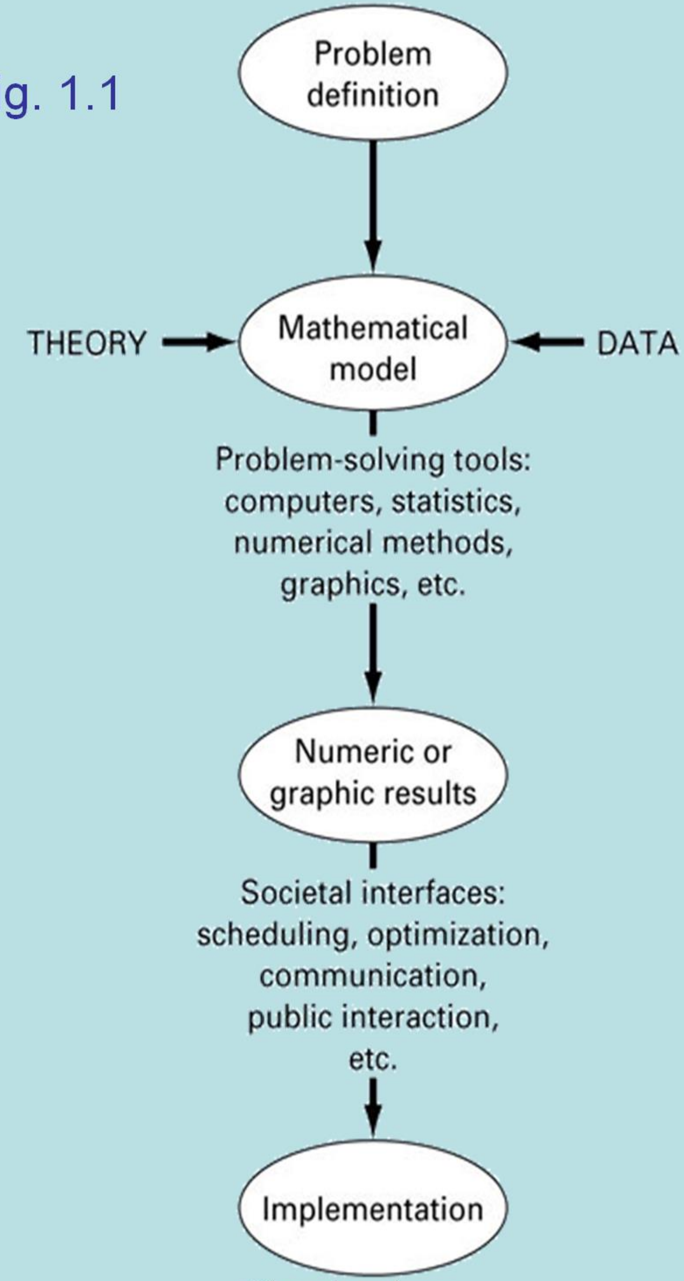
- Modelleme , yaklaşım, kesme hataları
- Modelleme ve simülasyon çok önemli
- Temel bilimler
- Uygulamalı bilimler
- Sosial bilimler

Mathematical Modeling and Engineering Problem solving

Chapter 1

- Requires understanding of engineering systems
  - By observation and experiment
  - Theoretical analysis and generalization
- Computers are great tools, however, without fundamental understanding of engineering problems, they will be useless.

Fig. 1.1


Problem definition
THEORY
Mathematical model
DATA
Problem-solving tools: computers, statistics, numerical methods, graphics, etc.
Numeric or graphic results
Societal interfaces: scheduling, optimization, communication, public interaction, etc.
Implementation

- A mathematical model is represented as a functional relationship of the form

$$\text{Dependent Variable} = f\left(\text{independent variables},\ \text{parameters},\ \text{forcing functions}\right)$$

- *Dependent variable*: Characteristic that usually reflects the state of the system
- *Independent variables*: Dimensions such as time ans space along which the systems behavior is being determined
- *Parameters*: reflect the system's properties or composition
- *Forcing functions*: external influences acting upon the system

Newton's 2nd law of Motion

- States that "*the time rate change of momentum of a body is equal to the resulting force acting on it.*"
- The model is formulated as

$$\mathbf{F = m\ a} \qquad (1.2)$$

F=net force acting on the body (N)

m=mass of the object (kg)

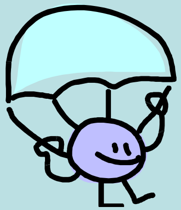
a=its acceleration (m/s^2)

- Formulation of Newton’s 2nd law has several characteristics that are typical of mathematical models of the physical world:
  - It describes a natural process or system in mathematical terms
  - It represents an idealization and simplification of reality
  - Finally, it yields reproducible results, consequently, can be used for predictive purposes.

- Some mathematical models of physical phenomena may be much more complex.

- Complex models may not be solved exactly or require more sophisticated mathematical techniques than simple algebra for their solution

  - Example, modeling of a falling parachutist:



$$\frac{dv}{dt} = \frac{F}{m}$$

$$F = F_D + F_U$$

$$F_D = mg$$

$$F_U = -cv$$

$$\frac{dv}{dt} = \frac{mg - cv}{m}$$

$$\frac{dv}{dt} = g - \frac{c}{m} v$$

- This is a differential equation and is written in terms of the differential rate of change dv/dt of the variable that we are interested in predicting.
- If the parachutist is initially at rest (v=0 at t=0), using calculus

$$v(t) = \frac{gm}{c}\left(1 - e^{-(c/m)t}\right)$$

Independent variable

Dependent variable

Forcing function

Parameters

Conservation Laws and Engineering

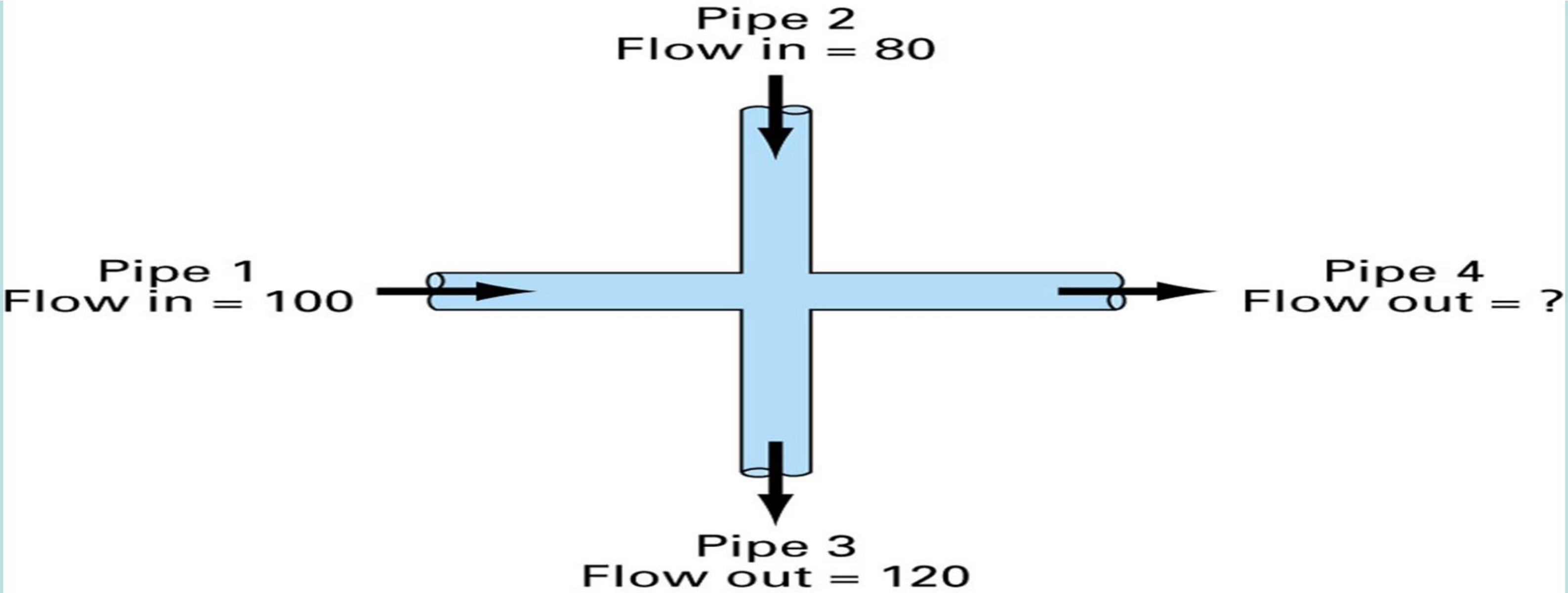
- Conservation laws are the most important and fundamental laws that are used in engineering.

$$\text{Change} = \text{increases} - \text{decreases} \quad (1.13)$$

- Change implies changes with time (transient). If the change is nonexistent (steady-state), Eq. 1.13 becomes

$$\text{Increases} = \text{Decreases}$$

Fig 1.6



- For steady-state incompressible fluid flow in pipes:

$$\text{Flow in} = \text{Flow out}$$

or

$$100 + 80 = 120 + \text{Flow}_4$$

$$\boxed{\text{Flow}_4 = 60}$$

Refer to Table 1.1

Programming and Software

Objective is how to use the computer as a tool to obtain numerical solutions to a given engineering model. There are two ways in using computers:

Use available software

Or, write computer programs to extend the capabilities of available software, such as Excel and Matlab.

Engineers should not be tool limited, it is important that they should be able to do both!

Programming and software

- Computer programs are set of instructions that direct the computer to perform a certain task.
- To be able to perform engineering-oriented numerical calculations, you should be familiar with the following programming topics:
- Simple information representation (constants, variables, and type declaration)
- Advanced information representation (data structure, arrays, and records)
- Mathematical formulas (assignment, priority rules, and intrinsic functions)
- Input/Output
- Logical representation (sequence, selection, and repetition)
- Modular programming (functions and subroutines)
- We will focus the last two topics, assuming that you have some prior exposure to programming.

Structured programming is a set of rules that prescribe god style habits for programmer.

An organized, well structured code

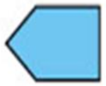
Easily sharable

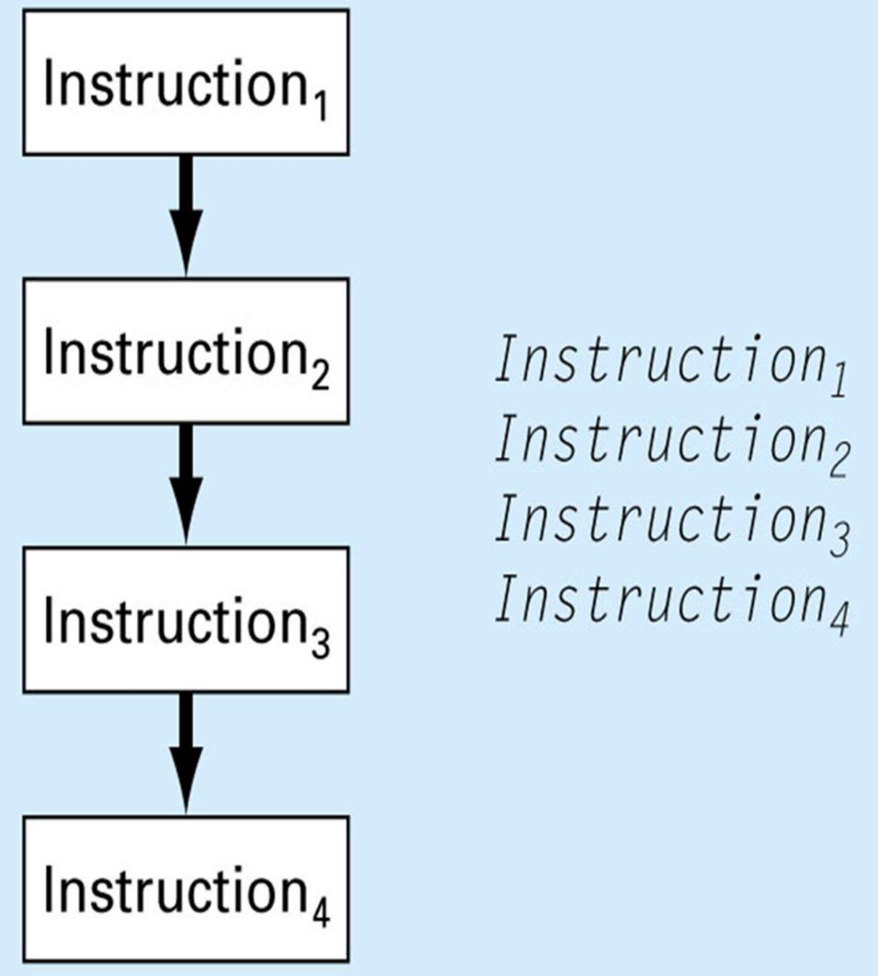
Easy to debug and test

Requires shorter time to develop, test, and update

The key idea is that any numerical algorithm can be composed of using the three fundamental structures:

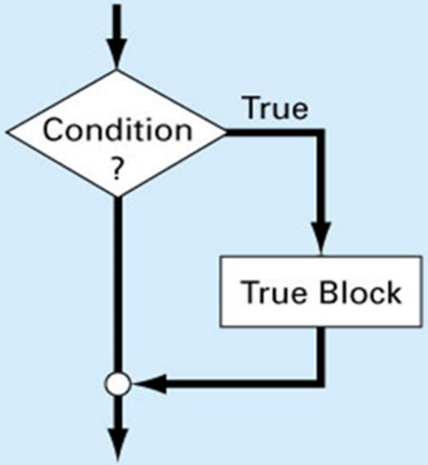
Sequence, selection, and repetition

SYMBOL	NAME	FUNCTION
	Terminal	Represents the beginning or end of a program.
	Flowlines	Represents the flow of logic. The humps on the horizontal arrow indicate that it passes over and does not connect with the vertical flowlines.
	Process	Represents calculations or data manipulations.
	Input/output	Represents inputs or outputs of data and information.
	Decision	Represents a comparison, question, or decision that determines alternative paths to be followed.
	Junction	Represents the confluence of flowlines.
	Off-page connector	Represents a break that is continued on another page.
	Count-controlled loop	Used for loops which repeat a prespecified number of iterations.



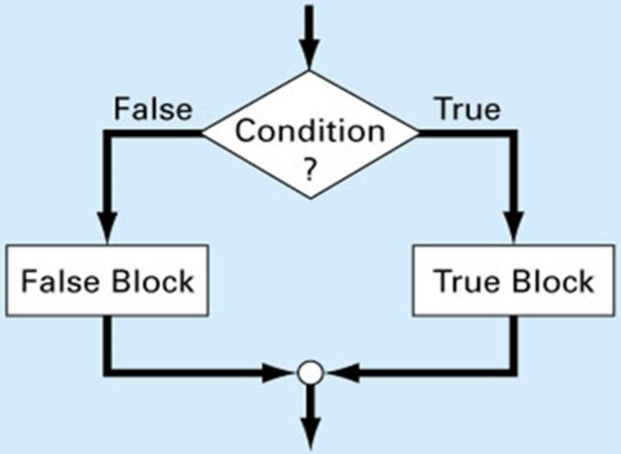
(*a*) Flowchart (*b*) Pseudocode

Flowchart | Pseudocode




```
IF condition THEN
  True block
ENDIF
```

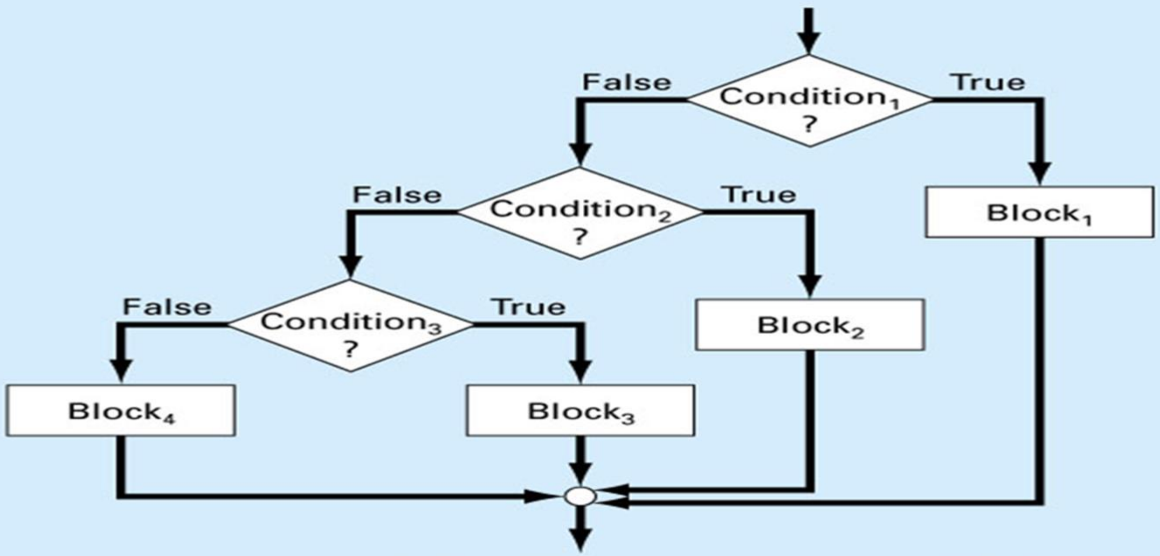
(*a*) Single-alternative structure (IF/THEN)




```
IF condition THEN
  True block
ELSE
  False block
ENDIF
```

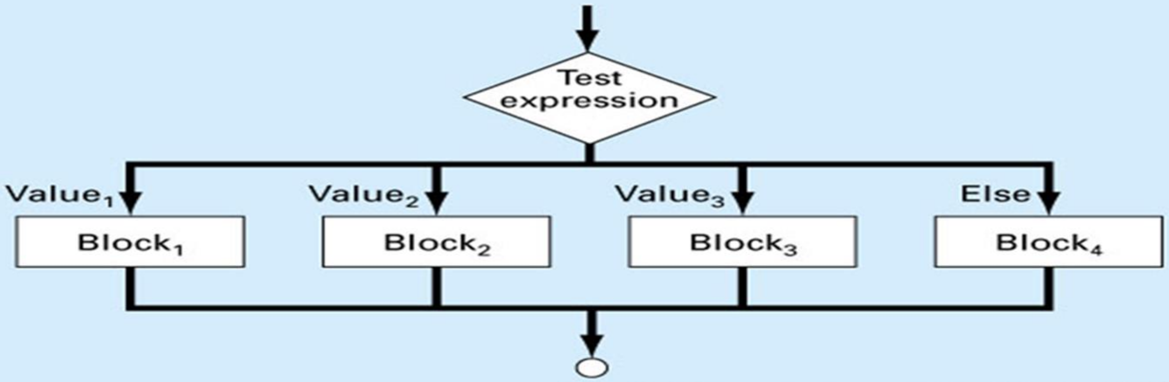
(*b*) Double-alternative structure (IF/THEN/ELSE)

Flowchart | Pseudocode




```
IF condition1 THEN
   Block1
ELSEIF condition2
   Block2
ELSEIF condition3
   Block3
ELSE
   Block4
ENDIF
```

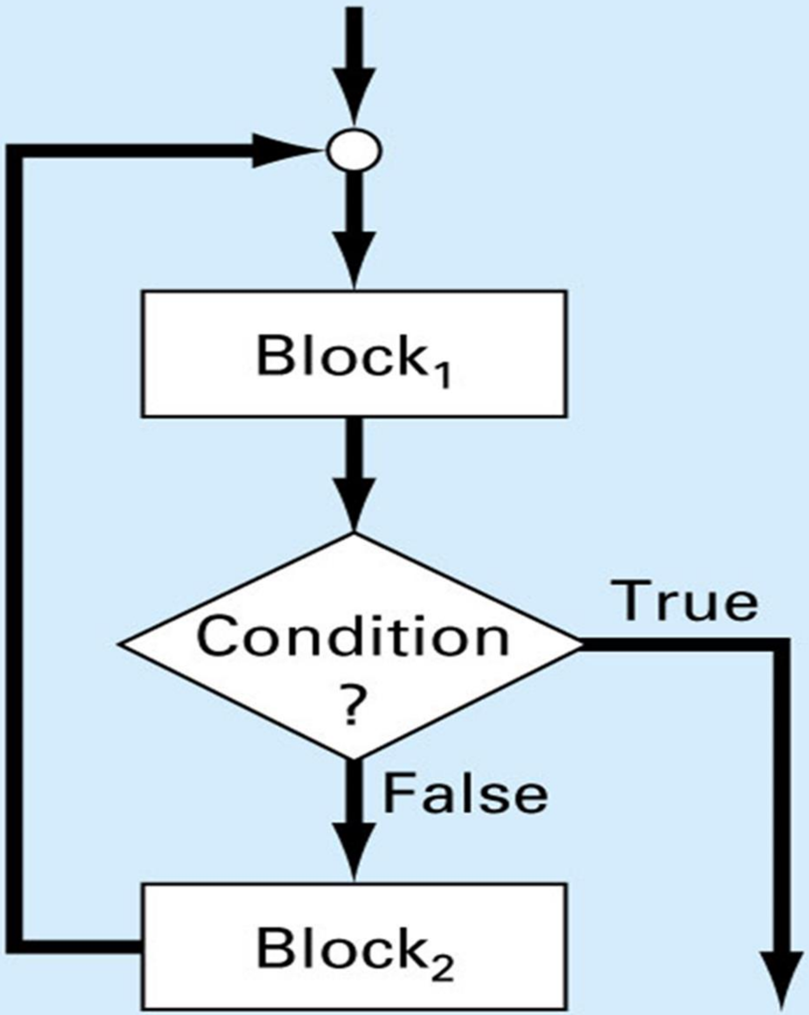
(a) Multialternative structure (IF/THEN/ELSEIF)




```
SELECT CASE Test Expression
   CASE Value1
      Block1
   CASE Value2
      Block2
   CASE Value3
      Block3
   CASE ELSE
      Block4
END SELECT
```

(b) CASE structure (SELECT or SWITCH)

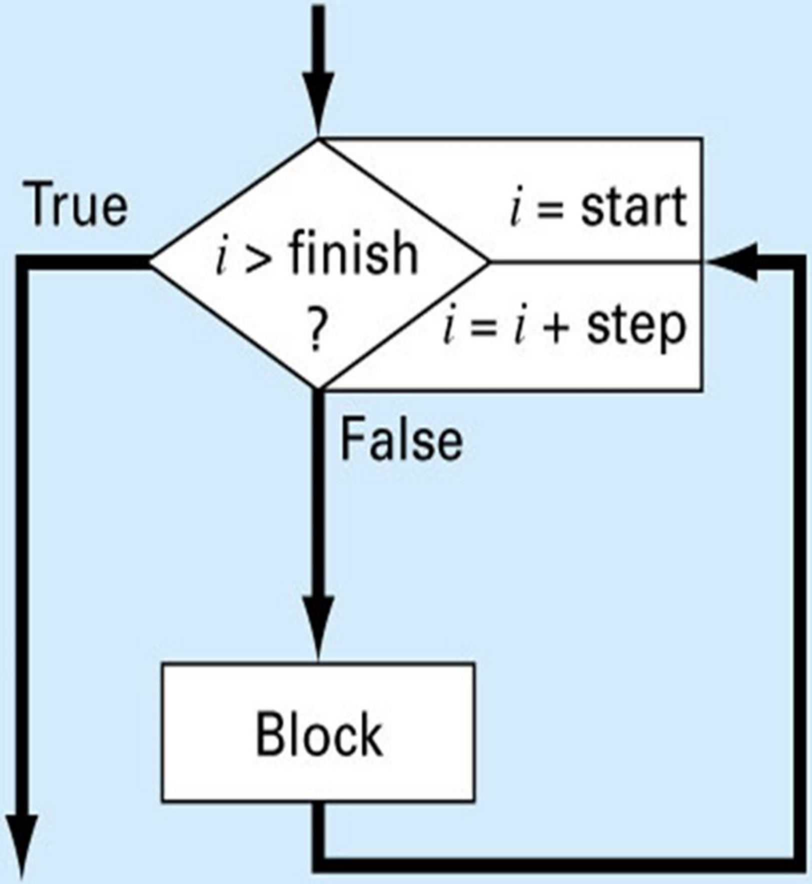
Flowchart



Pseudocode

```
DO
  Block1
  IF condition EXIT
  Block2
ENDDO
```

Flowchart



Pseudocode

```
DOFOR i = start, finish, step
  Block
ENDDO
```

The computer programs can be divided into subprograms, or modules, that can be developed and tested separately.

Modules should be as independent and self contained as possible.

Advantages to modular design are:

It is easier to understand the underlying logic of smaller modules

They are easier to debug and test

Facilitate program maintenance and modification

Allow you to maintain your own library of modules for later use

```
FUNCTION Euler(dt, ti, tf, yi)
t = ti
y = yi
h = dt
DO
  IF t + dt > tf THEN
    h = tf − t
  ENDIF
  dydt = dy(t, y)
  y = y + dydt * h
  t = t + h
  IF t ≥ tf EXIT
ENDDO
Euler = y
END
```

(*a*) **Pseudocode**	(*b*) **Excel VBA**

IF/THEN:

```
IF condition THEN
  True block
ENDIF
```

```
If b <> 0 Then
  r1 = -c / b
End If
```

IF/THEN/ELSE:

```
IF condition THEN
  True block
ELSE
  False block
ENDIF
```

```
If a < 0 Then
  b = Sqr(Abs(a))
Else
  b = Sqr(a)
End If
```

IF/THEN/ELSEIF:

```
IF condition1 THEN
  Block1
ELSEIF condition2
  Block2
ELSEIF condition3
  Block3
ELSE
  Block4
ENDIF
```

```
If class = 1 Then
  x = x + 8
ElseIf class < 1 Then
  x = x - 8
ElseIf class < 10 Then
  x = x - 32
Else
  x = x - 64
End If
```

CASE:

```
SELECT CASE Test Expression
  CASE Value1
    Block1
  CASE Value2
    Block2
  CASE Value3
    Block3
  CASE ELSE
    Block4
END SELECT
```

```
Select Case a + b
  Case Is < -50
    x = -5
  Case Is < 0
    x = -5 - (a + b) / 10
  Case Is < 50
    x = (a + b) / 10
  Case Else
    x = 5
End Select
```

DOEXIT:

```
DO
  Block1
  IF condition EXIT
  Block2
ENDIF
```

```
Do
  i = i + 1
  If i >= 10 Then Exit Do
  j = i*x
Loop
```

MATLAB

Is a flagship software which was originally developed as a matrix laboratory. A variety of numerical functions, symbolic computations, and visualization tools have been added to the matrix manipulations.

MATLAB is closely related to programming

Other languages

Fortran 90 (IMSL)
C++

Approximations and round of errors

For many engineering problems, we cannot obtain analytical solutions.

Numerical methods yield approximate results, results that are close to the exact analytical solution. We cannot exactly compute the errors associated with numerical methods.

Only rarely given data are exact, since they originate from measurements. Therefore there is probably error in the input information.

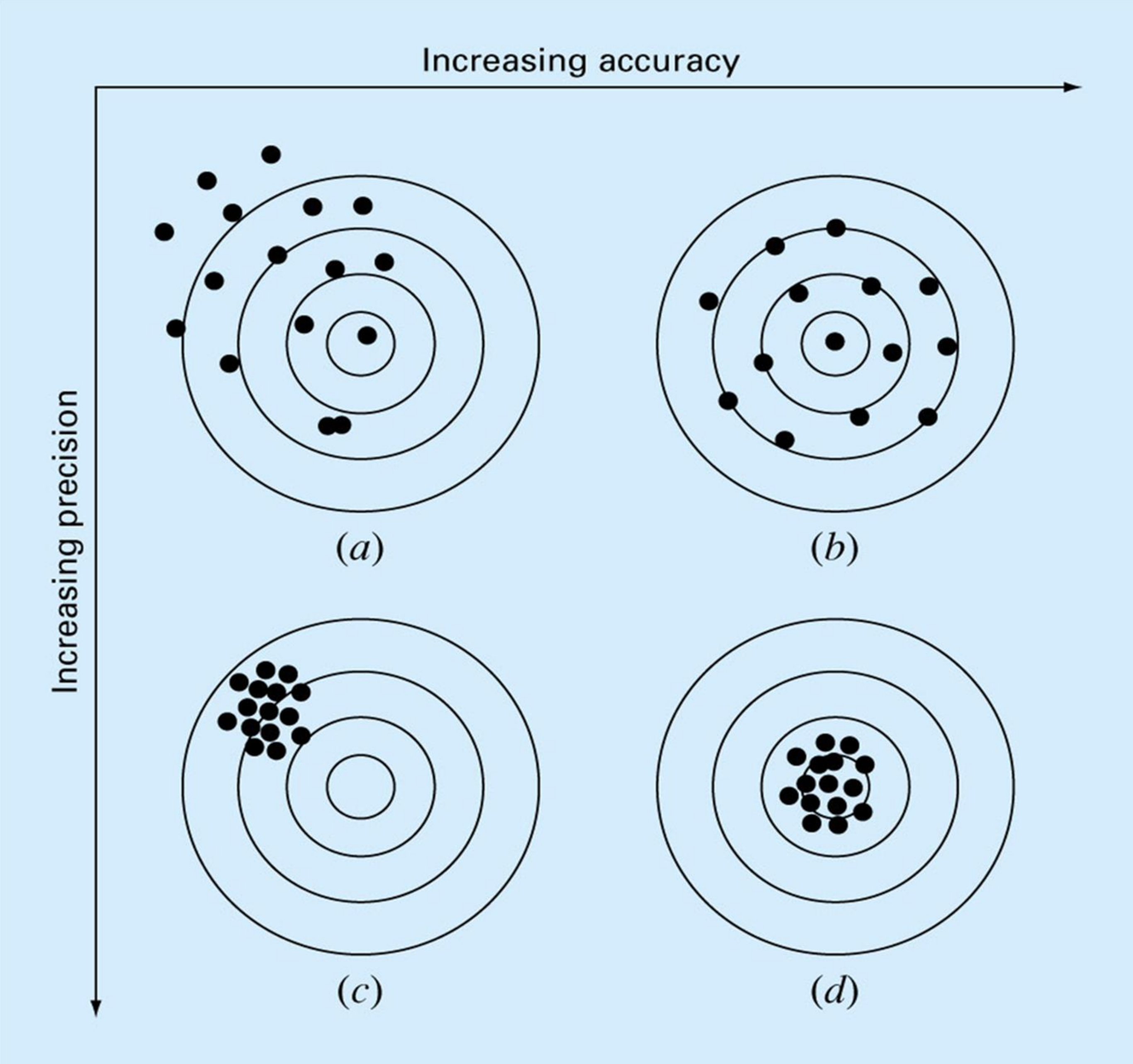
Algorithm itself usually introduces errors as well, e.g., unavoidable round-offs, etc …

The output information will then contain error from both of these sources.

How confident we are in our approximate result?

The question is "how much error is present in our calculation and is it tolerable?"

1. Accuracy. How close is a computed or measured value to the true value
2. Precision (or reproducibility). How close is a computed or measured value to previously computed or measured values.
3. Inaccuracy (or bias). A systematic deviation from the actual value.
4. Imprecision (or uncertainty). Magnitude of scatter.


Increasing accuracy
Increasing precision
(a)
(b)
(c)
(d)

Error definition

True Value = Approximation + Error

Et = True value – Approximation (+/-)

$$\text{True fractional relative error} = \frac{\text{true error}}{\text{true value}}$$

$$\text{True percent relative error, } \varepsilon_t = \frac{\text{true error}}{\text{true value}} \times 100\%$$

Error definition

For numerical methods, the true value will be known only when we deal with functions that can be solved analytically (simple systems). In real world applications, we usually not know the answer a priori. Then

$$\varepsilon_a = \frac{\text{Approximate error}}{\text{Approximation}} \times 100\%$$

Iterative approach, example Newton's method

$$\varepsilon_a = \frac{\text{Current approximation - Previous approximation}}{\text{Current approximation}} \times 100\%$$

you can be sure that the result is correct to at least n significant figures.

$$\left|\varepsilon_a\right| < \varepsilon_s$$

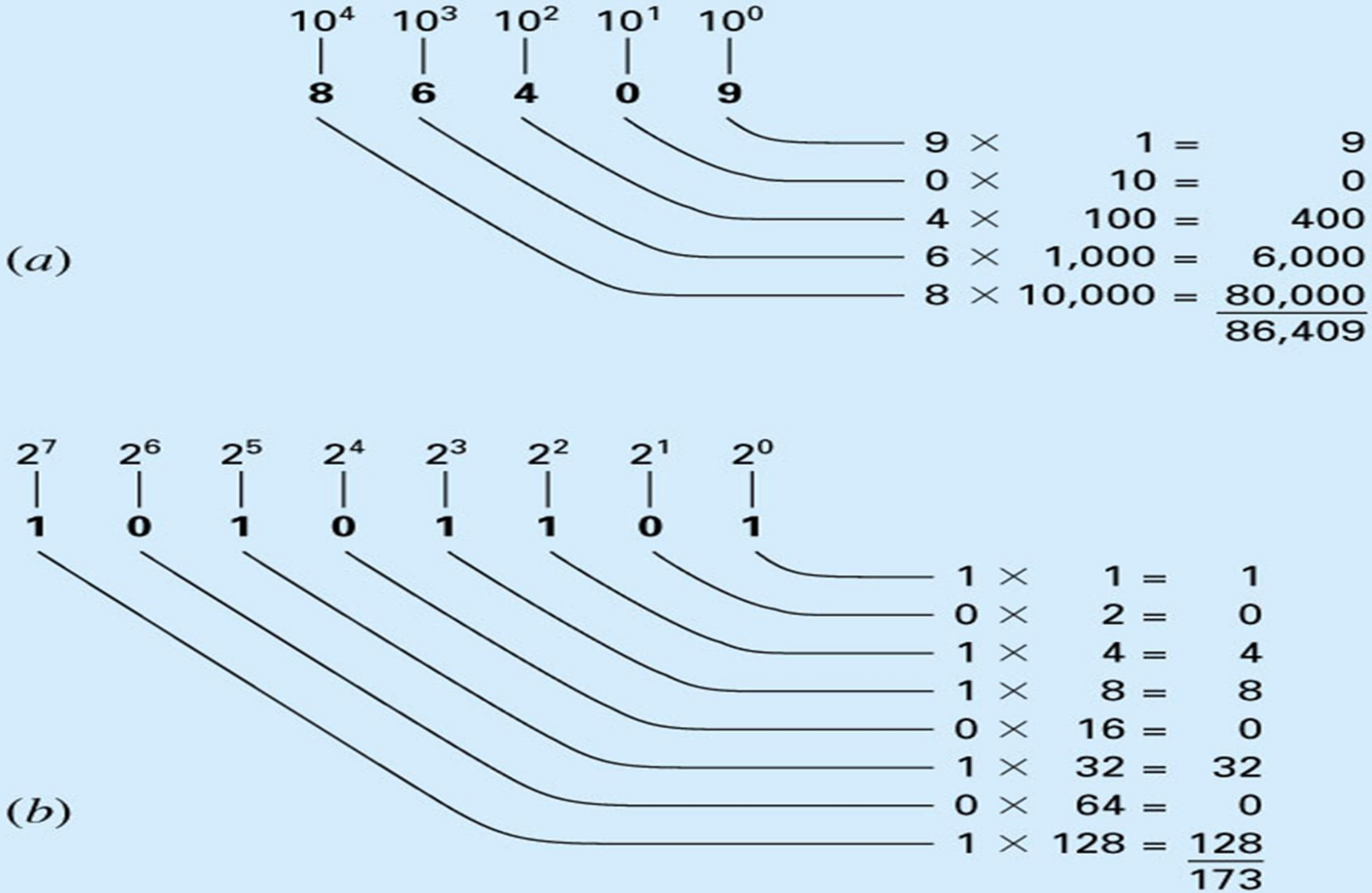
$$\varepsilon_s = \left(0.5 \times 10^{(2-n)}\right)\%$$

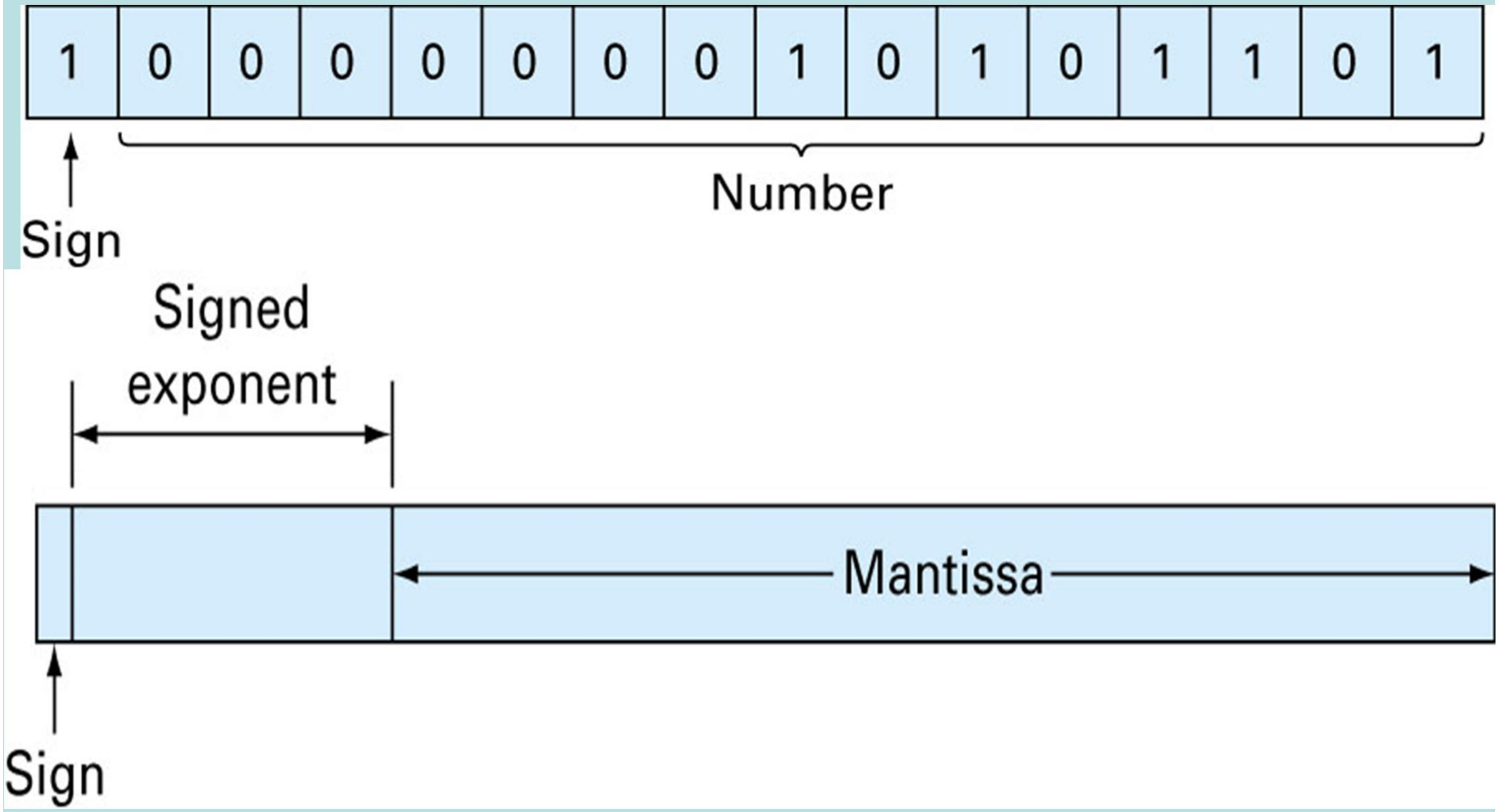
Round off errors

Numbers such as p, e, or cannot be expressed by a fixed number of significant figures.

Computers use a base-2 representation, they cannot precisely represent certain exact base-10 numbers.

Fractional quantities are typically represented in computer using "floating point" form, e.g.,


10⁴ 10³ 10² 10¹ 10⁰
8 6 4 0 9
9 × 1 = 9
0 × 10 = 0
4 × 100 = 400
6 × 1,000 = 6,000
8 × 10,000 = 80,000
86,409
(a)
2⁷ 2⁶ 2⁵ 2⁴ 2³ 2² 2¹ 2⁰
1 0 1 0 1 1 0 1
1 × 1 = 1
0 × 2 = 0
1 × 4 = 4
1 × 8 = 8
0 × 16 = 0
1 × 32 = 32
0 × 64 = 0
1 × 128 = 128
173
(b)


1 0 0 0 0 0 0 0 1 0 1 0 1 1 0 1
Number
Sign
Signed
exponent
Mantissa
Sign

Chopping

Example:

p=3.14159265358 to be stored on a base-10 system carrying 7 significant digits.

p=3.141592 chopping error et=0.00000065

If rounded

p=3.141593 et=0.00000035

Some machines use chopping, because rounding adds to the computational overhead. Since number of significant figures is large enough, resulting chopping error is negligible.

Therefore

for a base-10 system $0.1 \leq m < 1$

for a base-2 system $0.5 \leq m < 1$

Floating point representation allows both fractions and very large numbers to be expressed on the computer. However, Floating point numbers take up more room.

Take longer to process than integer numbers.

Round-off errors are introduced because mantissa holds only a finite number of significant figures