Bil 108

Introduction to the Scientific and Engineering Computing with MATLAB

Lecture 3

F.Aylin Konuklar Lale T. Ergene

Numbers on Computers

- The way in which the numbers are presented in the computer is a source of an error.

Ex:

>>1-5*0.2=0;

>>format long

1-0.2-0.2-0.2-0.2-0.2=5.551115123125783e-017

- **WHY?**

Bits, Bytes, and Words

base 10	conversion	base 2
1	$1 = 2^0$	0000 0001
2	$2 = 2^1$	0000 0010
4	$4 = 2^2$	0000 0100
8	$8 = 2^3$	0000 1000
9	$8 + 1 = 2^3 + 2^0$	0000 1001
10	$8 + 2 = 2^3 + 2^1$	0000 1010
27	$16 + 8 + 2 + 1 = 2^4 + 2^3 + 2^1 + 2^0$	$\underbrace{\text{0001 1011}}_{\text{one byte}}$

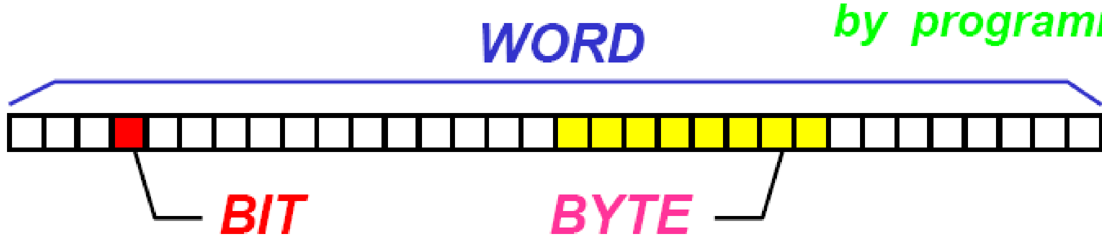
VARIABLES ARE REPRESENTED BY WORDS, COMPOSED OF BYTES, COMPOSED OF BITS

BIT = elemental circuit, ON (1) / OFF (0)

BYTE = string of 8 BITS

WORD = string of N BYTES

partially controllable by programmer



Digital Storage of Integers

- Integers can be exactly represented by base 2
- Typical size is 16 bits
- 32 bit and larger integers are available

- **Note:** All standard mathematical calculations in Matlab use floating point numbers.

Flaoting point numbers

- Numeric values with non-zero fractional parts are stored as **floating point numbers**.
- All floating point values are represented with a normalized scientific notation.

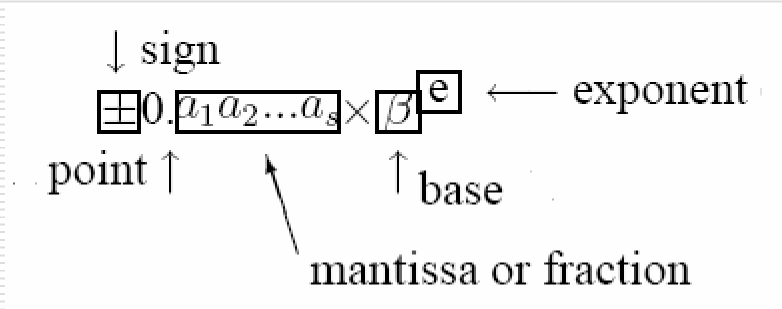
Example:

$$12.3792 = 0.123792 \times 10^{2} \text{ (base 10)}$$

$$-0.056 = -0.56 \times 10^{-1} \text{ (base 10)}$$

$$(110.01)_2 = 0.11001 \times 2^{3} \text{ (base 2)}$$

General form:



$$\boxed{\pm}0.\boxed{a_1 a_2 \ldots a_s} \times \boxed{\beta}^{\boxed{e}}$$

Digital Storage of Non-integer Numbers

- Floating point values have a fixed number of bits allocated for storage of the mantissa and a fixed number of bits allocated for storage of the exponent.

- Two common precisions are provided in numerical computing: single precision and double precision

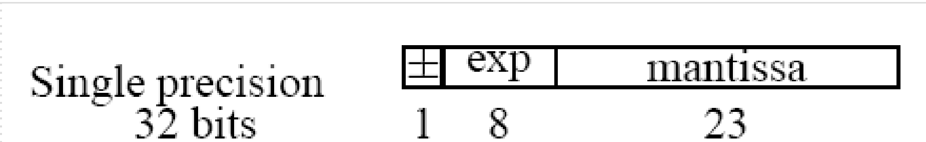
- Fixed number of bits are allocated to each number

single precision uses 32 bits per floating point number
double precision uses 64 bits per floating point number

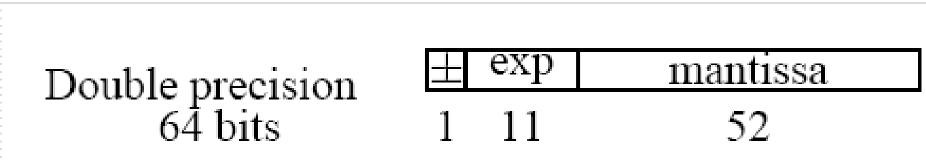
IEEE Standard

- Total number of bits are split into separate storage for the mantissa and exponent

single precision: 1 sign bit, 8 bit exponent,23 bit mantissa



double precision: 1 sign bit, 11 bit exponent, 52 bit mantissa



Consequences

- Limiting the number of bits allocated for storage of the exponent means that there are upper and lower limits on the magnitude of floating point numbers
- Limiting the number of bits allocated for storage of the mantissa means that there is a limit to the precision (number of significant digits) for any floating point number.

Errors

These are defined by:

$$\text{Absolute error} = \text{Approximate value} - \text{True value} \tag{1}$$

$$\text{Relative error} = \frac{\text{Absolute error}}{\text{True value}} \tag{2}$$

Often the relative error is represented as a percentage. A useful way to think of relative error is via the expression

$$\text{Approximate value} = \text{True value} \times (1 + \text{Relative error}) \tag{3}$$

Absolute and Relative Error (2)

Example: *Approximating* $\sin(x)$ *for small* x

Since

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots$$

we can approximate $\sin(x)$ with

$$\sin(x) \approx x$$

for small enough $x < 1$

The absolute error in this approximation is

$$E_{\mathrm{abs}} = x - \sin(x) = \frac{x^3}{3!} - \frac{x^5}{5!} + \ldots$$

And the relative error is

$$E_{\mathrm{abs}} = \frac{x - \sin(x)}{\sin(x)} = \frac{x}{\sin(x)} - 1$$

Errors in computer

- Round-off errors
- Truncation (or chopping) errors
- Other errors (data uncertainty, blunders, model errors)

Round-off errors in computing

- Finite-precision leads round-off in individual calculations
- Effects of round-off accumulate slowly
- The round-off errors are inevitable, solution is to create better algorithms
- Subtracting nearly equal may lead to severe loss of precision

Roundoff errors in computing (1)

Example 4 Compute $r = x^2 - y^2$, where $x = 4.005$ and $y = 4.004$ with a 4-digit precision.

By application of direct scheme we have

$$r = x^2 - y^2 = 16.04(0025) - 16.03(2016) = 0.01$$

The true value of r is 0.008009. This leads to the relative error

$$\boxed{E_{\text{rel}} = \frac{0.008009 - 0.01}{0.008009} \cdot 100\% \cong -24.859\%!!!}$$

From the other hand, applying the scheme $r = (x - y)(x + y)$ we have:

$$r = (4.005 - 4.004)(4.004 + 4.005) = 0.001 \cdot 8.009 = 0.008009$$

The result is exact and the relative error

$$\boxed{E_{\text{rel}} = 0\%!!!}$$

Machine Precision (1)

The magnitude of roundoff errors is quantified by *machine precision* ε_m.

There is a number, ε_m such that

$$1 + \delta = 1$$

whenever $\delta < \varepsilon_m$.

In exact arithmetic, ε_m is identically zero.

MATLAB uses double precision (64 bit) arithmetic. The built-in variable eps stores the value of ε_m.

$$\text{eps} = 2.2204 \times 10^{-16}$$

Truncation error

Consider the series for $\sin(x)$

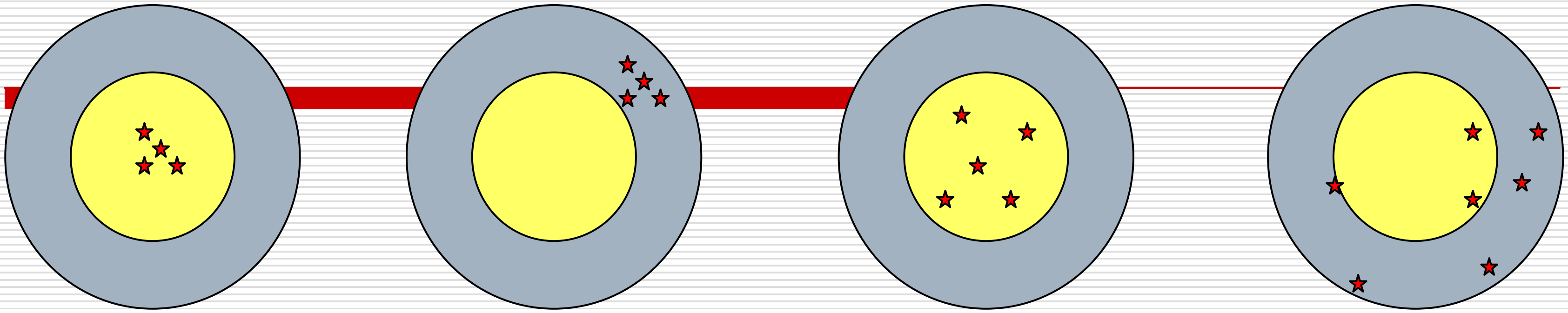
$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots$$

For small x, only a few terms are needed to get a accurate approximation to $sin(x)$. The higher order terms are 'truncated'

$$f_{\text{true}} = f_{\text{sum}} + \text{truncation error}$$

The size of truncation error depends on x and the number of terms included in f_{sum}.

Numbers: precision and accuracy



Good Accuracy
Good Precision

Good Precision
Poor Accuracy

Good Accuracy
Poor Precision

Poor Accuracy
Poor Precision

Numbers: precision and accuracy

- Low precision: $\pi = 3.14$
- High precision: $\pi = 3.140101011$
- Low accuracy: $\pi = 3.10212$
- High accuracy: $\pi = 3.14159$
- High accuracy & precision: $\pi = 3.141592653$

- **Precision**

The smallest difference that can be represented on the computer (help eps)

- **Accuracy**

How close your answer is to the “actual” or “real” answer.

- **Recognize:**

MATLAB (and other programs that use IEEE doubles) give you 15-16 “good” digits

What is an algorithm?

… a well-defined procedure that allows an agent to solve a problem.

Note: often the agent is a computer or a robot…

Example algorithms

- Cooking a dish
- Shampooing hair
- Making a pie

Algorithms

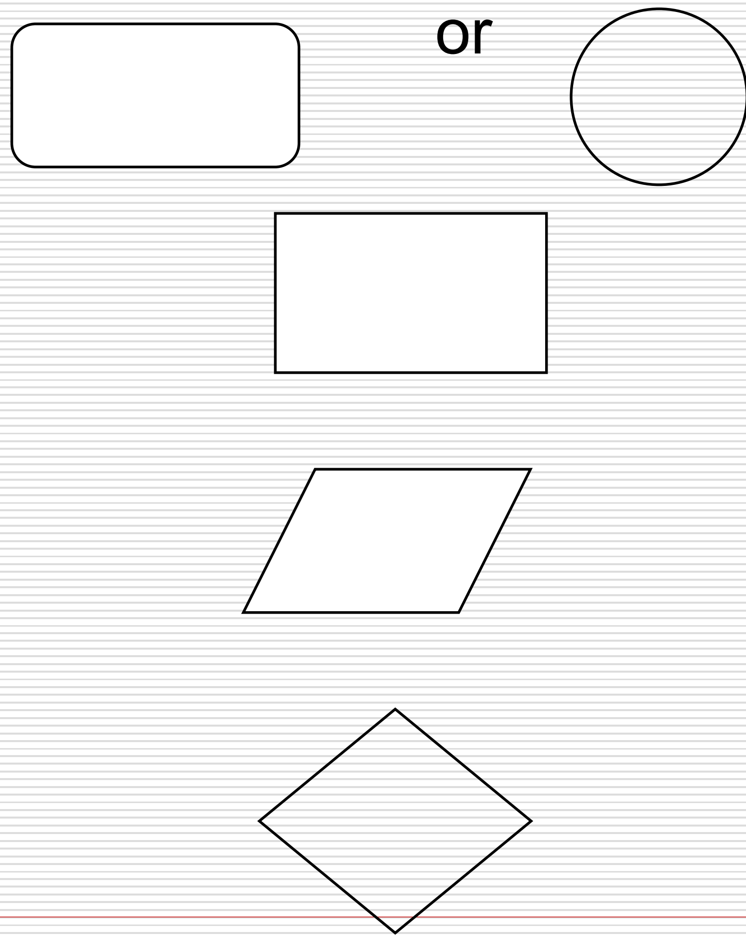
An algorithm must:

1. Be well-ordered
2. Each operation must be effectively computable
3. Terminate.

Flowcharting

- Flowcharting is another technique used in designing and representing algorithms.
- A flowchart is a graph consisting of geometric shapes that are connected by flow lines.
- From the flowchart one can write the program code.

Symbols (I)



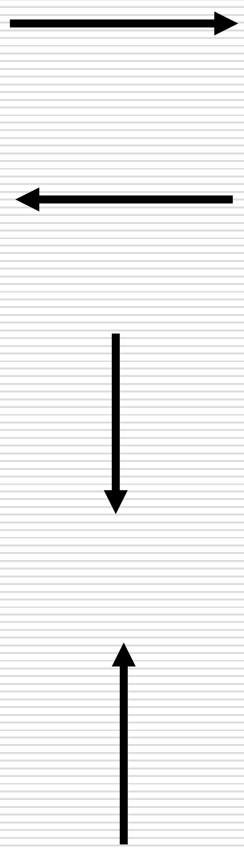
Terminal symbol

Process symbol

Input-Output symbol

Decision symbol

Symbols (II)



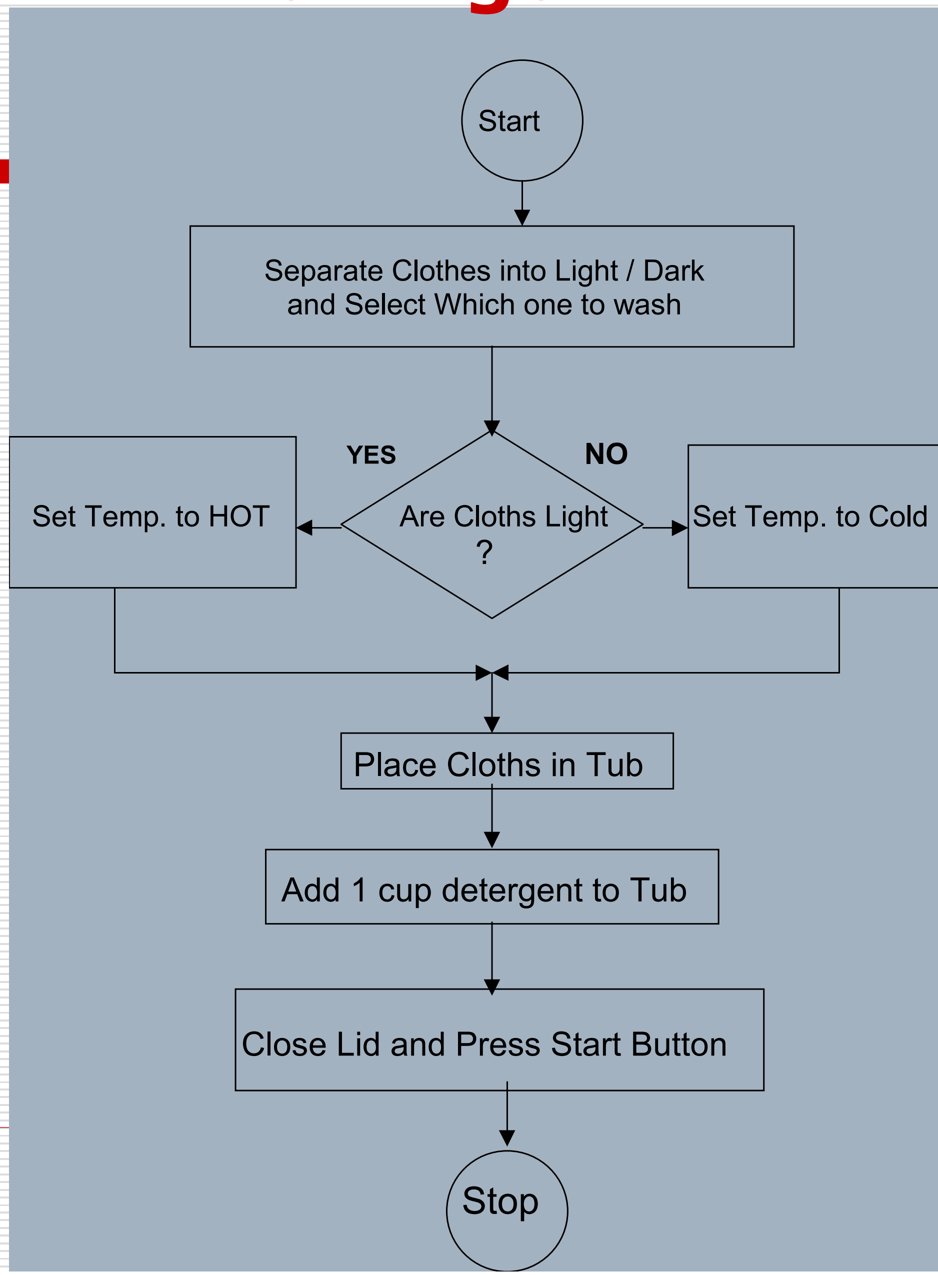
Flow Lines indicating the logical sequence of statements

One must follow the flow of the arrows direction, one cannot go in the opposite direction of the arrows flow.

Washing Machine Instructions

- Separate clothes into white clothes and colored clothes.
- For white clothes:
  - Set water temperature knob to HOT.
  - Place white laundry in tub.
- For colored clothes:
  - Set water temperature knob to COLD.
  - Place colored laundry in tub.
- Add 1 cup of powdered laundry detergent to tub.
- Close lid and press the start button.

Flow Chart for Algorithm



Logical Operators

Logical operators are used to combine logical expressions (with
"and" or "or"), or to change a logical value with "not"

Operator Meaning

& and

| or

~ not

INPUT		OUTPUT				
A	B	A&B	A|B	~A	~B	
false	false	false	false	true	true	
false	true	false	true	true	false	
true	false	false	true	false	true	
true	true	true	true	false	false	

Logical and Relational Operators

- Relational operators involve comparison of two values.
- The result of a relational operation is a logical (True/False) value.
- Logical operators combine (or negate) logical values to produce another logical value.
- There is always more than one way to express the same comparison

if Constructs

- **Syntax:**
- if *expression*
  - *block of statements*
- end

- The *block of statements* is executed only if the *expression* is true.

- **Example:**

```
if a < 0
disp('a is negative');
end
```

- *One line* format uses comma after if *expression*

```
if a < 0, disp('a is negative'); end
```

if. . . else

Multiple choices are allowed with if. . . else and if. . . elseif constructs

```
if x < 0

disp(' x is negative; sqrt(x) is imaginary ');

else

r = sqrt(x);

r

end
```

if. . . elseif

It’s a good idea to include a default **else** to catch cases that don’t match preceding **if** and **elseif** blocks

```
if x > 0
disp('x is positive');
elseif x < 0
disp('x is negative');
else
disp('x is exactly zero');
end
```

Calculating worker's pay

- A worker is paid according to his hourly wage up to 40 hours, and 50% more for overtime. Write a program in a m-file that calculates the pay to a worker. The program asks the user to enter the number of hours and the hourly wage. The program then displays the pay.

The switch Construct

- A switch construct is useful when a test value can take on discrete values that are either integers or strings.

- **Syntax:**

switch *expression*

case *value1*,

block of statements

case *value2*,

block of statements

...

otherwise,

block of statements

end

Flow Control
Repetition or Looping

- A sequence of calculations is repeated until *either*

1. All elements in a vector or matrix have been processed

OR

2. The calculations have produced a result that meets a predetermined termination criterion

- Looping is achieved with for loops and while loops.

for loops

- for loops are most often used when each element in a vector or matrix is to be processed.

- **Syntax:**

for index = *expression*

block of statements

end

- **Example:** *Sum of elements in a vector*

```
x = 1:5; % create a row vector
for k = 1:length(x)
sumx = sumx + x(k)
end
sumx
```

for loop variations

- **Example:** *A loop with an index incremented by two*

```
for k = 1:2:n
...
end
```

- **Example:** *A loop with an index that counts down*

```
for k = n:-1:1
...
end
```

Modify vector elements

- A vector is give by: V=[5, 17,-3,8,0,-1,12,15,20,-6, 6,4, -7,16]. Write a program that doubles the elements that are positive and are divisible by 3 or 5, and raise to the power 3 the elements that are negative but greater than -5.

while loops

- while loops are most often used when an iteration is repeated until some termination criterion is met.

- **Syntax:**

while *expression*

block of statements

end

- The *block of statements* is executed as long as *expression* is true.

- For a while-end loop to execute properly;
  - The conditional expression in the while command must include at least one variable;
  - The variables in the conditional expression must have assigned when MATLAB executes the while command fot the first time;
  - At least one of the variables in the conditional execution must be assigned a new value in the commands that are between the while and the end. Otherwise once the looping starts it will never stop since the conditional expression will remain true.

```
x=1
while x<=15
x=2*x
end
```

PS: Nobody is perfect ☺ In case of execution of an indefinite loop, you may stop it by pressing the CTRL+C or CTRL+Break keys.

The **break** and **return** statements provide an alternative way to
exit from a loop construct.

break and **return** may be applied to for loops or while loops.

break is used to escape from an enclosing while or for loop.
Execution continues at the end of the enclosing loop construct.

return is used to force an exit from a **function**. This can have the effect of escaping from a loop. Any statements following the loop that are in the function body are skipped.

Comparison of break and return

break is used to escape the current `while` or `for` loop.

return is used to escape the current function.

```
function k = demoBreak(n)

...

while k<=n
   if x(k)>0.8
     break;
   end
   k = k + 1;
end
```

jump to end of enclosing "`while ... end`" block

```
function k = demoReturn(n)

...

while k<=n
   if x(k)>0.8
     return;
   end
   k = k + 1;
end
```

return to calling function

data analysis functions

- max(x) Determines the largest value in x
- min(x)   Determines the smallest value in x
- sum(x)   Determines the sum of the lemetns in x
- prod(x) Determines the product of the elements in x

PS: Chapter 3 pp.91-96 Please check it!!!!

Mean and Median

- mean(x) Computes the mean(average value) of the elements of the vector x.
- median(x) Determines the median value of the elements in the vector x

$$\mu = \frac{\sum_{k=1}^{N} x_k}{N}$$

$$where \sum_{k=1}^{N} x_k = x_1 + x_2 + \ldots\ldots + x_N$$

- sort (x) Returns a vector with the values of x in ascending order.

Variance and Standard Deviation

- By simply, the variance of the values from the mean

- The standard deviation is deifned as the square root of the variance

- std(x) Computes the standard deviation of the values in x

$$\sigma^2 = \frac{\sum_{k=1}^{N} (x_k - \mu)^2}{N-1}$$