Computer Programming: Programming in MATLAB

Asst. Prof. Dr. Yalçın İşler
Izmir Katip Celebi University

Outline

- Introduction
- Relational and Logical Operators
- Conditional Statements: IF-ELSEIF-ELSE-END
- Switch-Case Statement
- Loops: FOR & WHILE
- Nested Loops and Conditionals
- Break and Continue

Introduction

- A computer program is a sequence of computer commands. In a simple program the commands are executed one after the other in the order that they are typed.
- MATLAB provides several tools that can be used to control the flow of a program.
- It should also be noted that function files (Chapter 6) can be used in programming.
- A function file is a subprogram. When the program reaches the command line that has the function, it provides input to the function, and “waits” for the results. The function carries out the calculations, transfers the results back to the program that “called” the function, which then continues to the next command.

Relational Operators

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

Relational Operators (cont'd)

- When two numbers or scalars are compared, the result is 1 (logical true) if the comparison, according to the relational operator, is true, and 0 (logical false) if the comparison is false.
- If two arrays are compared (only arrays with the same size can be compared), the comparison is done *element-by-element,* and the result is a logical array of the same size with 1's and 0's according to the outcome of the comparison at each address.

Relational Operators (cont'd)

```
>> 5>8
ans =
     0
>> a=5<10
a =
     1

>> y=(6<10)+(7>8)+(5*3==60/4)

y =
     2
```

Checks if 5 is larger than 8.

Since the comparison is false (5 is not larger than 8) the answer is 0.

Checks if 5 is smaller than 10, and assigns the answer to `a`.

Since the comparison is true (5 is smaller than 10) the number 1 is assigned to `a`.

Using relational operators in math expression.

Equal to 1 since 6 is smaller than 10.

Equal to 0 since 7 is not larger than 8.

Equal to 1 since 5*3 is equal to 60/4.

Relational Operators (cont'd)

```
>> b=[15 6 9 4 11 7 14]; c=[8 20 9 2 19 7 10];
>> d=c>=b
d =
     0     1     1     0
>> b == c
ans =
     0     0     1     0     0     1     0
>> b~=c
ans =
     1     1     0     1     1     0     1
>> f=b-c>0
f =
     1     0     0     1     0     0     1
```

Define vectors b and c.

Checks which c elements are larger or equal to b elements.

Assigns 1 where an element of c is larger than or equal to an element of b.

Checks which b elements are equal to c elements.

Checks which b elements are not equal to c elements.

Subtracts c from b and then checks which elements are larger than zero.

Relational Operators (cont'd)

```
>> A=[2 9 4; -3 5 2; 6 7 -1]
A =
     2     9     4
    -3     5     2
     6     7    -1
>> B=A<=2
B =
     1     0     0
     1     0     1
     0     0     1
```

Define a 3×3 matrix A.

Checks which elements in A are smaller or equal to 2. Assigns the results to matrix B.

Relational Operators (cont'd)

```
>> r = [8 12 9 4 23 19 10]
r =
     8    12     9     4    23    19    10
>> s=r<=10
s =
     1     0     1     1
>> t=r(s)
t =
     8     9     4    10
>> w=r(r<=10)
w =
     8     9     4    10
```

Define a vector `r`.

Checks which `r` elements are smaller than or equal to 10.

A logical vector `s` with 1's at positions where elements of `r` are smaller than or equal to 10.

Use `s` for addresses in vector `r` to create vector `t`.

Vector `t` consists of elements of `r` in positions where `s` has 1's.

The same can be done in one step.

Logical Operators

Logical Operator	Name	Description
& Example: A&B	AND	Operates on two operands (A and B). If both are true, the result is true (1), otherwise the result is false (0).
| Example: A|B	OR	Operates on two operands (A and B). If either one, or both are true, the result is true (1), otherwise (both are false) the result is false (0).
~ Example: ~A	NOT	Operates on one operand (A). Gives the opposite of the operand. True (1) if the operand is false, and false (0) if the operand is true.

Logical Operators (cont'd)

```
>> a=5|0
a =
     1
>> ~25
ans =
     0
>> t=25*((12&0)+(~0)+(0|5))
t =
    50
>> x=[9 3 0 11 0 15]; y=[2 0 13 -11 0 4];
>> x&y
ans =
     1     0     0     1
>> z=x|y
z =
     1     1     1     1
>> ~(x+y)
ans =
     0     0     0     1
```

5 OR 0 (assign to variable a).

1 is assigned to a since at least one number is true (nonzero).

NOT 25.

The outcome is 0 since 25 is true (nonzero) and the opposite is false.

Using logical operators in math expression.

Define two vectors x and y.

The outcome is a vector with 1 in every position that both x and y are true (nonzero elements), and 0's otherwise.

The outcome is a vector with 1 in every position that either or both x and y are true (nonzero elements), and 0's otherwise.

The outcome is a vector with 0 in every position that the vector x + y is true (nonzero elements), and 1 in every position that x + y is false (zero element).

Order of Precedence

Precedence	Operation
1 (highest)	Parentheses (If nested parentheses exist, inner have precedence).
2	Exponentiation.
3	Logical NOT (~).
4	Multiplication, division.
5	Addition, subtraction.
6	Relational operators (>, <, >=, <=, ==, ~=).
7	Logical AND (&).
8 (lowest)	Logical OR (|).

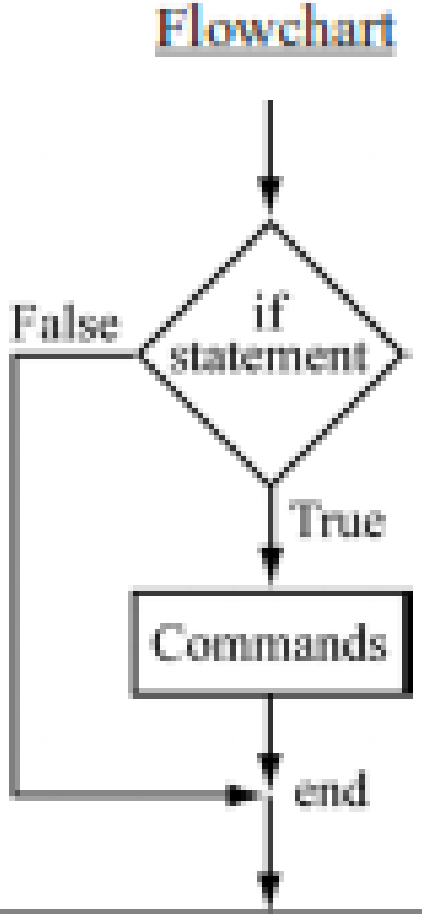
Function	Description	Example

`all(A)`	Returns 1 (true) if all elements in a vector A are true (nonzero). Returns 0 (false) if one or more elements are false (zero). If A is a matrix, treats columns of A as vectors, returns a vector with 1's and 0's.	`>> A=[6 2 15 9 7 11];`
		`>> all(A)`
		`ans =`
		`1`
		`>> B=[6 2 15 9 0 11];`
		`>> all(B)`
		`ans =`
		`0`

`any(A)`	Returns 1 (true) if any element in a vector A is true (nonzero). Returns 0 (false) if all elements are false (zero). If A is a matrix, treats columns of A as vectors, returns a vector with 1's and 0's.	`>> A=[6 0 15 0 0 11];`
		`>> any(A)`
		`ans =`
		`1`
		`>> B = [0 0 0 0 0 0];`
		`>> any(B)`
		`ans =`
		`0`

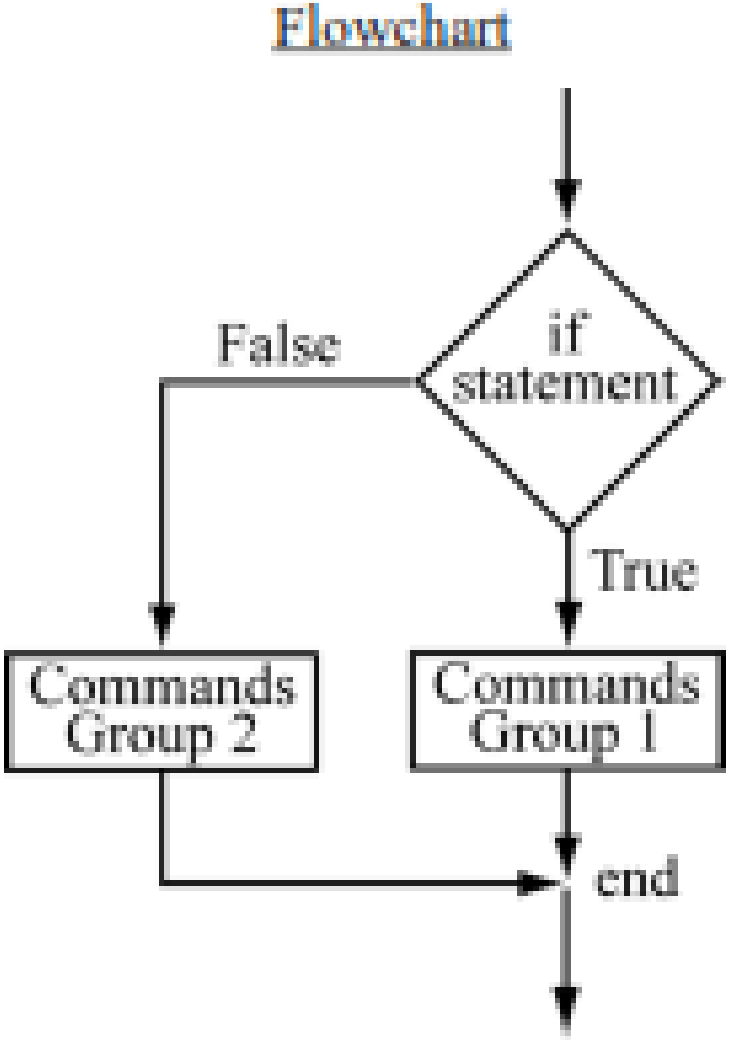
`find(A)`
 	
`find(A>d)`
	If A is a vector, returns the indices of the nonzero elements.
	If A is a vector, returns the address of the elements that are larger than d (any relational operator can be used).
		`>> A=[0 9 4 3 7 0 0 1 8];`
		`>> find(A)`
		`ans =`
		`2 3 4 5 8 9`
		`>> find(A>4)`
		`ans =`
		`2 5 9`

Conditional Statements (1)



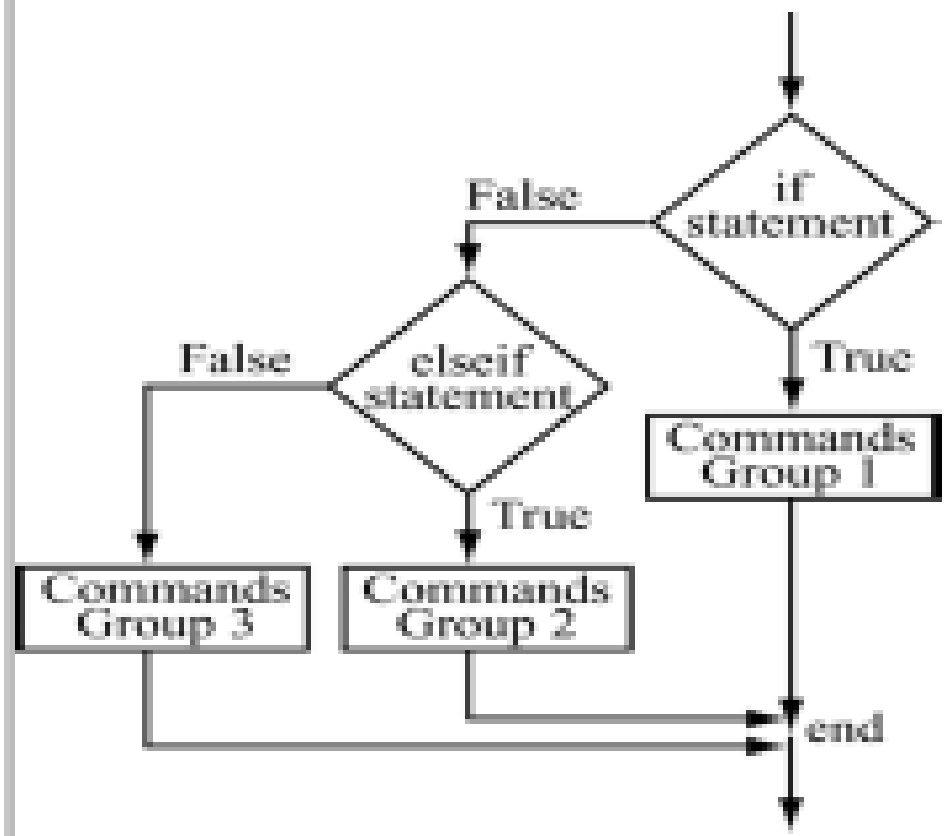

```
.....
.....  MATLAB program.
.....
if  conditional expression
        .........
        .........        A group of
        .........        MATLAB commands.
end
.....
.....  MATLAB program.
.....
```

Conditional Statements (2)



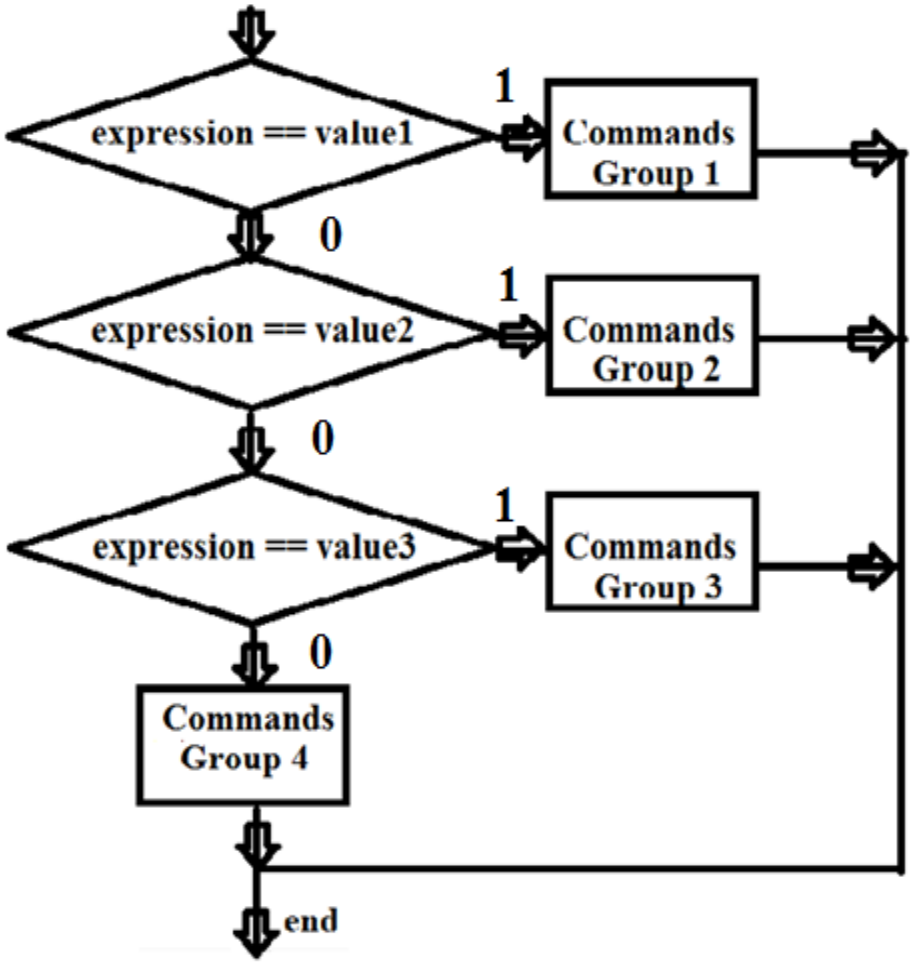

```
.....
.....   MATLAB program.
.....

if  conditional expression
        ........
        ........   Group 1 of
        ........   MATLAB commands.
else
        ........
        ........   Group 2 of
        ........   MATLAB commands.
end
.....
.....   MATLAB program.
.....
```

Conditional Statements (3)




```
.....
.....     MATLAB program.
.....

if conditional expression
        ........
        ........    Group 1 of
        ........    MATLAB commands.
elseif conditional expression
        ........
        ........    Group 2 of
        ........    MATLAB commands.
else
        ........
        ........    Group 3 of
        ........    MATLAB commands.
end
.....
.....     MATLAB program.
.....
```

SWITCH-CASE

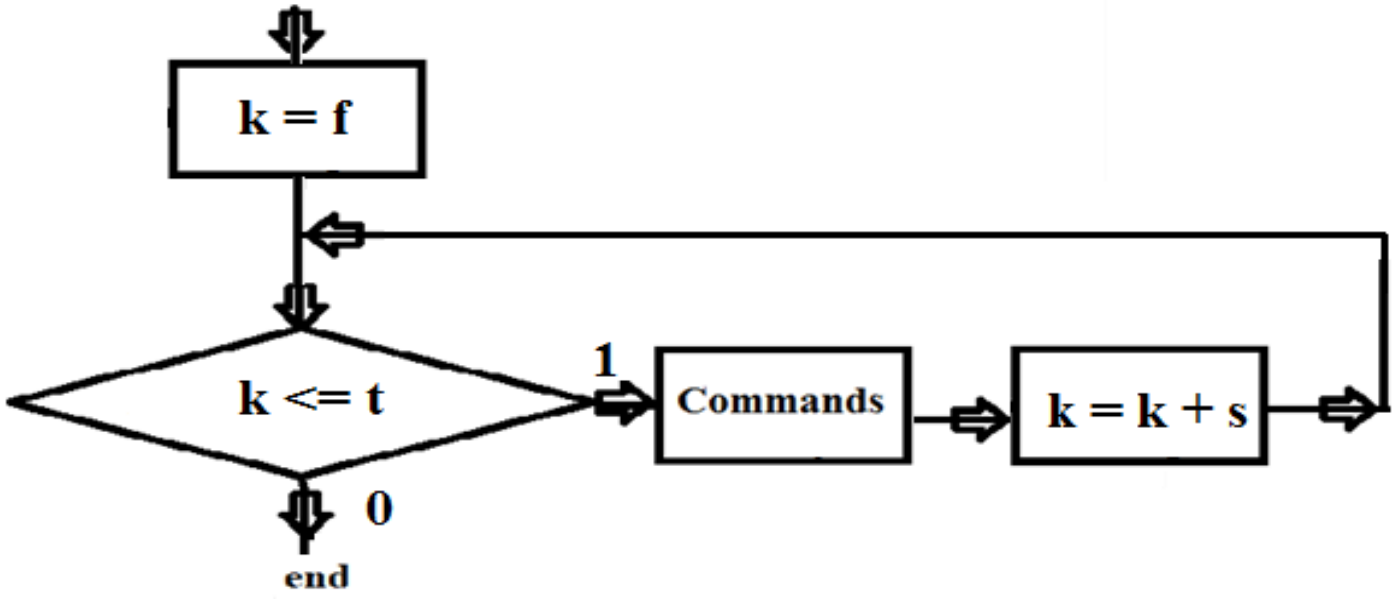


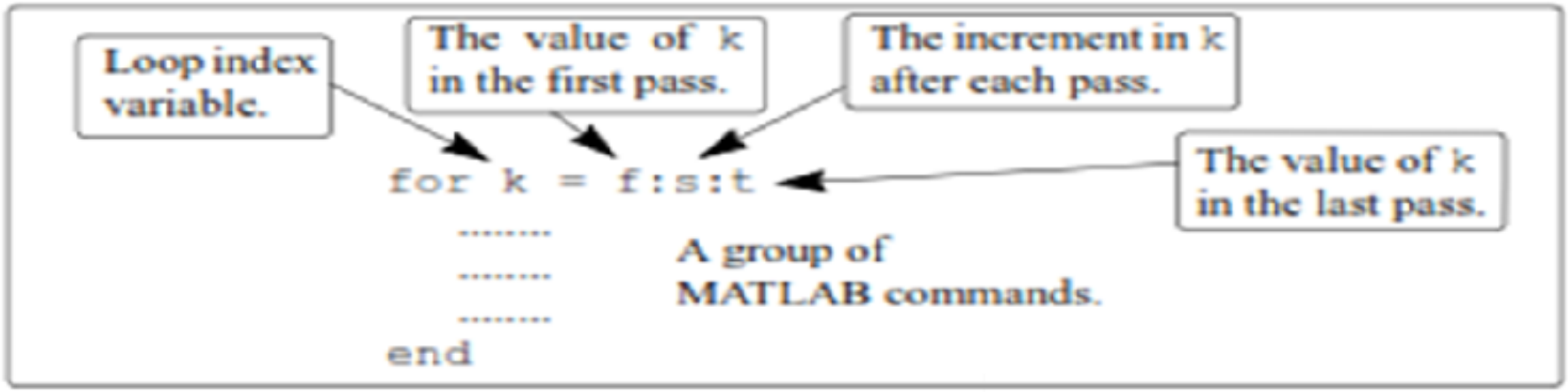

```
.....
      MATLAB program.
.....

switch expression
    case value1
        .....
        .....      ] Group 1 of commands.
    case value2
        .....
        .....      ] Group 2 of commands.
    case value3
        .....
        .....      ] Group 3 of commands.
    otherwise
        .....
        .....      ] Group 4 of commands.
end
.....
      MATLAB program.
.....
```

LOOPS

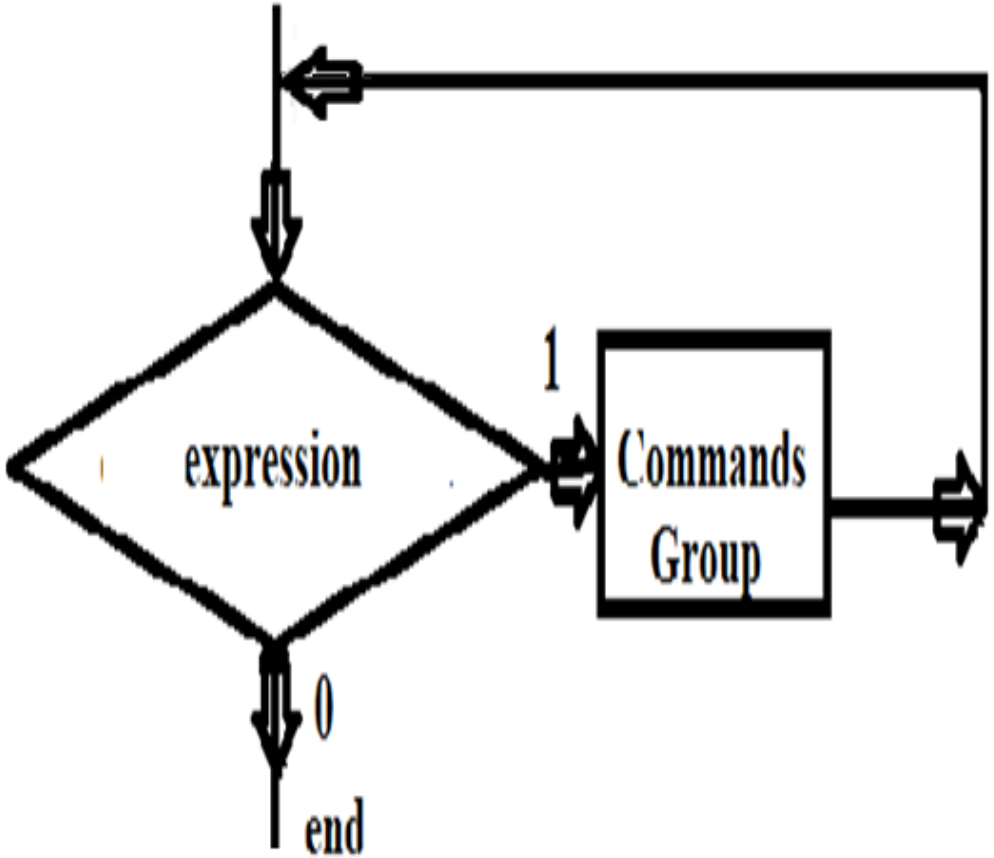
- In for-end loops the execution of a command, or a group of commands, is repeated a predetermined number of times.
- In while-end loops the execution of a command, or a group of commands, is repeated if the conditional expression is true (1); otherwise, MATLAB skips to the end statement and continues with the program. This looping process continues until the conditional expression is false.

LOOPS: FOR


k = f
k <= t
1
Commands
k = k + s
0
end

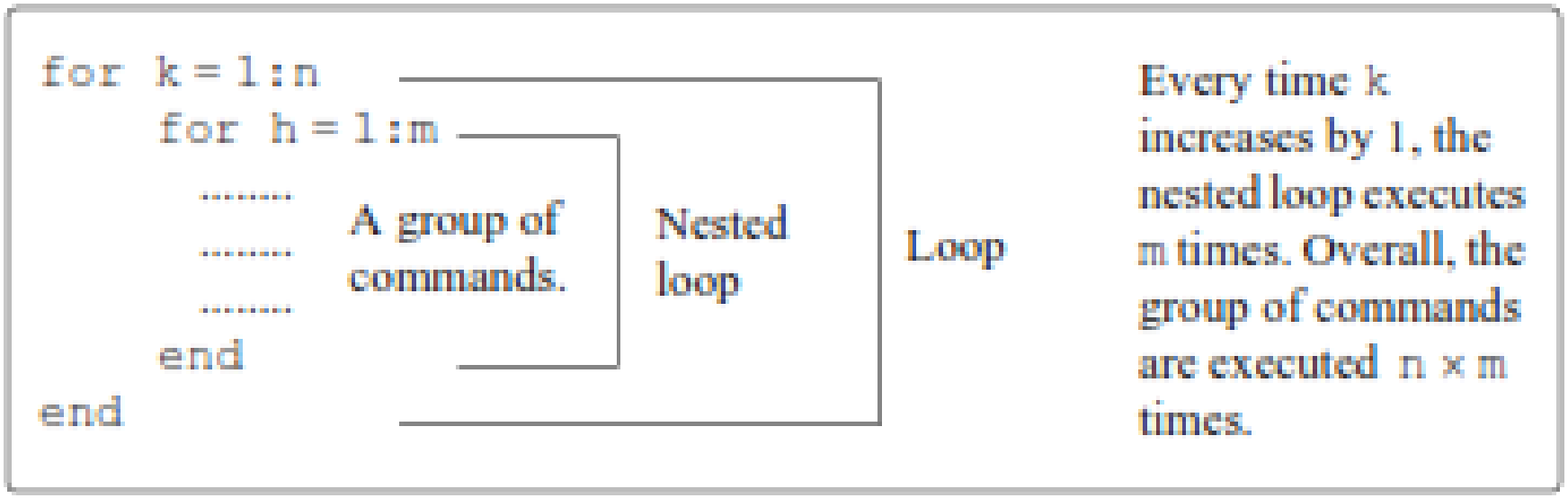

Loop index variable.
The value of k in the first pass.
The increment in k after each pass.
The value of k in the last pass.
for k = f:s:t
.....
.....
.....
A group of MATLAB commands.
end

LOOPS: WHILE




```
while conditional expression
    .......
    .......        A group of
    .......        MATLAB commands.
    .......
end
```

NESTED LOOPS & CONDITIONALS



This nested structure also available for IF conditionals.

BREAK

- When inside a loop (for and while), the break command terminates the execution of the loop (the whole loop, not just the last pass). When the break command appears in a loop, MATLAB jumps to the end command of the loop and continues with the next command (does not go back to the for command of that loop).
- If the break command is inside a nested loop, only the nested loop is terminated.
- When a break command appears outside a loop in a script, or function file, it terminates the execution of the file.
- The break command is usually used within a conditional statement. In loops it provides a method to terminate the looping process if some condition is met.

CONTINUE

- The continue command can be used inside a loop (for and while) to stop the present pass and start the next pass in the looping process.
- The continue command is usually a part of a conditional statement. When MATLAB reaches the continue command, it does not execute the remaining commands in the loop, but skips to the end command of the loop and then starts a new pass.

1. **BREAK means exit immediately from the last loop you started to run in which this break command.**
2. **CONTINUE means do not execute remaining commands in the loop, but do not exit from the last loop in which this continue command.**

Laboratory Session

Do both examples and sample applications in Chapter 7 of the textbook (the first 2 hours).

Homework #9

Not later than the next week:

Solve problems 7, 14, 19, and 26 from the Chapter 7 of the textbook using Matlab.