

Computer Applications Lab Lab 5 Programming in Matlab

Chapter 4
Sections 1,2,3,4

Dr. Iyad Jafar

Adapted from the publisher slides

Outline

- Program design and development
- Relational operators and logical variables
- Logical operators and functions
- Loops
- The switch statement
- Debugging Matlab programs

2

Program Design and Development

- **Algorithm:** an ordered sequence of precisely defined instructions that performs some task in a finite amount of time.
- **Ordered** means that the instructions can be numbered, but an algorithm must have the ability to alter the order of its instructions using a control structure.
- There are three categories of algorithmic operations:
 - **Sequential operations:** Instructions executed in order.
 - **Conditional operations:** Control structures that first ask a question to be answered with a true/false answer and then select the next instruction based on the answer.
 - **Iterative operations (loops):** Control structures that repeat the execution of a block of instructions.

3

Program Design and Development

Structured Programming

A technique for designing programs in which a hierarchy of modules is used, each having a single entry and a single exit point, and in which control is passed downward through the structure without unconditional branches to higher levels of the structure (the use of GOTO). In Matlab these modules can be built-in or user-defined functions.

4

Program Design and Development

Structured Programming

- **Advantages**

1. Structured programs are **easier** to write because the programmer can study the overall problem first and then deal with the details later.
2. Modules (functions) written for one application can be used for other applications (this is called **reusable code**).
3. Structured programs are **easier to debug** because each module is designed to perform just one task and thus it can be tested separately from the other modules.
4. Structured programming is **effective in a teamwork environment** because several people can work on a common program, each person developing one or more modules.
5. Structured programs are **easier to understand and modify**, especially if meaningful names are chosen for the modules and if the documentation clearly identifies the module's task.

5

Program Design and Development

Steps for creating computer solutions

1. State the problem concisely.
2. Specify the data to be used by the program. This is the **"input"**.
3. Specify the information to be generated by the program. This is the **"output"**.
4. Work through the solution steps by hand or with a calculator; use a simpler set of data if necessary.
5. Write and run the program.
6. Check the output of the program with your hand solution.
7. Run the program with your input data and perform a reality check on the output.
8. If you will use the program as a general tool in the future, test it by running it for a range of reasonable data values; perform a reality check on the results.

6

Program Design and Development

Documentation

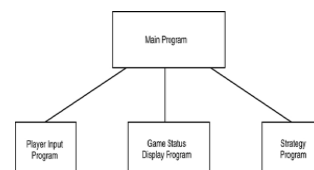
1. Proper selection of variable names to reflect the quantities they represent.
2. Use of comments within the program.
3. Use of structure charts.
4. Use of flowcharts.
5. A verbal description of the program, often in **pseudocode**.

7

Program Design and Development

Documenting with Charts

- Two types of charts aid in developing structured programs and in documenting them. These are **structure charts** and **flowcharts**.
- A **structure chart** is a graphical description showing how the different parts of the program are connected together.

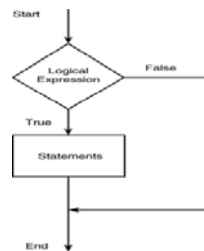


8

Program Design and Development

Documenting with Charts

- **Flowcharts** are useful for developing and documenting programs that contain conditional statements, because they can display the various paths (called “branches”) that a program can take, depending on how the conditional statements are executed.



9

Program Design and Development

Documenting with Pseudocode

- We can document with *pseudocode*, in which *natural language and mathematical expressions are used to construct statements that look like computer statements but without detailed syntax.*
- *Each pseudocode instruction may be numbered, but should be unambiguous and computable.*

10

Relational Operators

- Six operators that are used for comparison of variables (scalars or arrays) or expressions
- They have equal precedence (evaluated from left to right)

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

11

Relational Operators

- For example, suppose that $x = [6,3,9]$ and $y = [14,2,9]$.

The following Matlab session shows some examples.

```

>>z = (x < y)
z = 1 0 0
>>z = (x ~= y)
z = 1 1 0
>>z = (x > 8)
z = 0 0 1
  
```

Note
Comparison is performed
element-wise

12

Relational Operators

- The relational operators can be used for **array addressing**. For example, with $x = [6,3,9]$ and $y = [14,2,9]$, typing $z = x(x < y)$ finds all the elements in x that are less than the corresponding elements in y . The result is $z = 6$.
- The **arithmetic operators** $+$, $-$, $*$, $/$, and \backslash have **precedence over the relational operators**. Thus the statement $z = 5 > 2 + 7$ is equivalent to $z = 5 > (2+7)$ and returns the result $z = 0$.
- We can use parentheses to change the order of precedence; for example, $z = (5 > 2) + 7$ evaluates to $z = 8$.

13

The Logical Class

- When the relational operators are used, such as $x = (5 > 2)$ they create a **logical variable**, in this case, x .
- Prior to Matlab 6.5 **logical** was an attribute of any numeric data type. Now **logical** is a first-class data type and a Matlab class, and so **logical** is now equivalent to other first-class types such as character and cell arrays.
- Logical variables **may have only the values 1 (true) and 0 (false)**.

14

Logical Operators

Operator	Name	Definition
~	NOT	$\sim A$ returns an array the same dimension as A ; the new array has ones where A is zero and zeros where A is nonzero. $A=[1, 2, 0, 4, 0, 7]$; $C=\sim A$ $C=[0, 0, 1, 0, 1, 0]$
&&	AND	$A \&\& B$ returns an array the same dimension as A and B ; the new array has ones where both A and B have nonzero elements and zeros where either A or B is zero. $A=[1, 2, 0, 4, 0, 7]$ $B=[0, 1, 8, 2, 0, 3]$ $C=A\&\&B$ $C=[0, 1, 0, 1, 0, 1]$
	OR	$A B$ returns an array the same dimension as A and B ; the new array has ones where at least one element in A or B is nonzero and zeros where A and B are both zero. $A=[1, 2, 0, 4, 0, 7]$ $B=[0, 1, 8, 2, 0, 3]$ $C=A B$ $C=[1, 1, 1, 0, 1]$ (continued ...)

15

Logical Operators

Operator	Name	Definition
&&	Short-Circuit AND	Operator for scalar logical expressions. $A \&\& B$ returns true if both A and B evaluate to true, and false if they do not. If A equals zero, then the entire expression will evaluate to logical 0 (false), regardless of the value of B . Under these circumstances, there is no need to evaluate B because the result is already known. In this case, MATLAB short-circuits the statement by evaluating only the first term.
	Short-Circuit OR	Operator for scalar logical expressions. $A B$ returns true if either A or B or both evaluate to true, and false if they do not. This is similar to the case above. If the first term is true, then, regardless of the value of B , the statement will evaluate to true.

16

Precedence of Different Operators

Precedence Operator type

First	Parentheses; evaluated starting with the innermost pair.
Second	Arithmetic operators and logical NOT (-); evaluated from left to right.
Third	Relational operators; evaluated from left to right.
Fourth	Logical AND.
Fifth	Logical OR.

17

Logical Functions

Logical function Definition

<code>ischar (A)</code>	Returns a 1 if A is a character array and 0 otherwise.
<code>isempty (A)</code>	Returns a 1 if A is an empty matrix and 0 otherwise.
<code>isinf (A)</code>	Returns an array of the same dimension as A, with ones where A has 'inf' and zeros elsewhere.
<code>isnan (A)</code>	Returns an array of the same dimension as A with ones where A has 'NaN' and zeros elsewhere. ('NaN' stands for "not a number," which means an undefined result.)

18

Logical Functions

<code>isnumeric (A)</code>	Returns a 1 if A is a numeric array and 0 otherwise.
<code>isreal (A)</code>	Returns a 1 if A has no elements with imaginary parts and 0 otherwise.
<code>logical (A)</code>	Converts the elements of the array A into logical values. Given <code>A=[1,2,3,4,5,6,7,8,9]</code> , the statement <code>B=logical(eye(3))</code> returns the logical array <pre>B= 1 0 0 0 1 0 0 0 1</pre>

This can then be used in logical indexing that returns A's diagonal elements:

```
A (B)
ans=
1
5
9
```

Remember the `eye` function from Chapter 2? See page 105. The statement `eye (n)` creates an $n \times n$ identity matrix which is a square matrix whose diagonal elements are all equal to 1.

19

Logical Functions

- Use Matlab documentation to check the following logical functions.
 1. `isstr`
 2. `isnumeric`
 3. `isvector`
 4. `ishold`
 5. `isequal`
 6. `isscalar`

20

The Find Function

- The Matlab function **find** computes an array containing the indices of the *nonzero elements of the numeric array x*.

- **Example**

```
>> x = [-2, 0, 4];  
>> y = find(x)  
      y = 1 3
```

The resulting array $y = [1, 3]$ indicates that the first and third elements of x are nonzero.

- **[u,v,w] = find(A)**

Computes the arrays u and v containing the row and column indices of the nonzero elements of the array A and computes the array w containing the values of the nonzero elements. The array w may be omitted.

21

The find Function

- When the find function is used with arrays, the return value is the **linear indices** of the nonzero elements.
- With **linear indices**, we can refer to array elements with a single value $A(k)$, where k is the element location if the columns of the array are appended to each other.

- **Example**

$$A = \begin{bmatrix} -2 & -1 & 0 \\ 5 & 6 & 7 \\ 1 & 2 & 3 \end{bmatrix}$$

$A(2)$ is equivalent to $A(2,1)$ which is 5

$A(5)$ is equivalent to $A(2,2)$ which is 6

22

The Find Function

With logical operators

- The expression inside the function is evaluated first, then the search for nonzero elements is performed

- Consider the session

```
>> x = [5, -3, 0, 0, 8]; y = [2, 4, 0, 5, 7];  
>> z = find(x&y)  
      z = 1 2 5
```

Note that the find function **returns** the *indices*, and *not the values*.

- To extract the values, use array indexing $y = x(z)$

23

Conditional Statements

- The if statement's basic form is
if *logical expression*
statements
end
- Every *if* statement must have an accompanying **end** statement.
- The *end* statement marks the end of the statements that are to be executed if the logical expression is true.

24

Conditional Statements

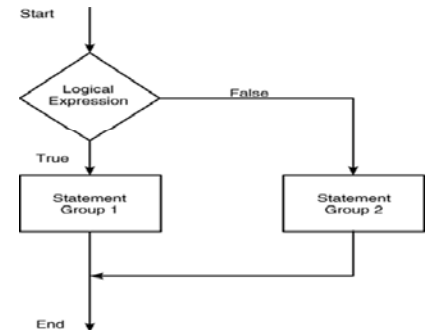
The else statement

- The basic structure for the use of the else statement is

```
if    logical expression
      statements group 1
else
      statement group 2
end
```

25

Conditional Statements



26

Conditional Statements

Example

```
x = [4,-9,25];
if x < 0
    disp('Some elements of x are negative.')
else
    y = sqrt(x)
end
```

Because the test `if x < 0` is false, when this program is run it gives the result

```
y = [2, +3.000i, 5]
```

27

Conditional Statements

Example

```
x = [4,-9,25];
if x >= 0
    y = sqrt(x)
else
    disp('Some elements of x are negative.')
end
```

When executed, it produces the following message: Some elements of x are negative. The test `if x < 0` is false, and the test `if x >= 0` also returns a false value because `x >= 0` returns the vector `[1,0,1]`.

28

Conditional Statements

The elseif Statement

- The general form of the if statement is

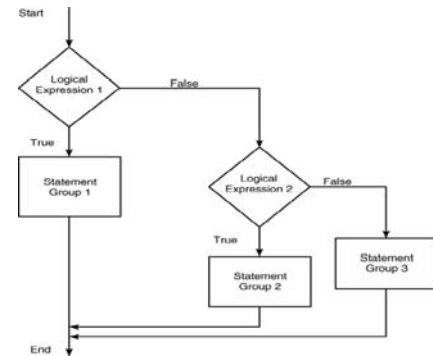

```

            if logical expression 1
              statement group 1
            elseif logical expression 2
              statement group 2
            else
              statement group 3
            end
            
```

The else and elseif statements may be omitted if not required. However, if both are used, the else statement must come after the elseif statement to take care of all conditions that might be unaccounted for.

29

Conditional Statements



30

Conditional Statements

- Example, suppose that $y = \begin{cases} \exp(x) - 1, & x < 0 \\ \sqrt{x}, & 0 \leq x \leq 10 \\ \log(x), & x > 10 \end{cases}$

- The following statements will compute y if x already has a scalar value

```

if x > 10
    y = log(x)
elseif x >= 0
    y = sqrt(x)
else
    y = exp(x) - 1
end
    
```

31

Loops

- For Loops.**
- Used to repeat a set of statements for specific number of times.

A simple example of a for loop is

```

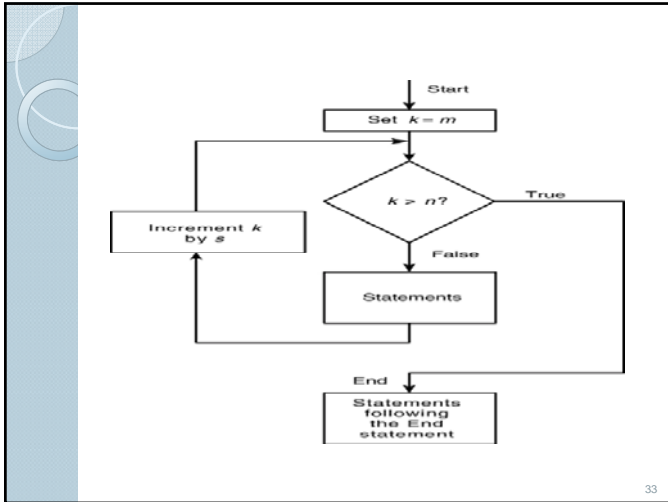
for k = 5:10:35
    x = k^2
end
    
```

start increment end

You do not need a semi-colon to suppress printing k

- The loop variable k is initially assigned the value 5, and x is calculated from $x = k^2$. Each successive pass through the loop increments k by 10 and calculates x until k exceeds 35. Thus k takes on the values 5, 15, 25, and 35, and x takes on the values 25, 225, 625, and 1225. The program then continues to execute any statements following the end statement.

32



Note the following rules when using *for* loops with the loop variable expression $k = m:s:n$

- The step value s may be **negative**.
Example: $k = 10:-2:4$ produces $k = 10, 8, 6, 4$.
- If s is omitted, the step value **defaults to 1**.
- If s is positive, the loop will not be executed if m is greater than n .
- If s is negative, the loop will not be executed if m is less than n .
- If m equals n , the loop will be executed **only once**.
- If the step value s is not an integer, round-off errors can cause the loop to execute a different number of passes than intended.

Loops – The While Statement

- The while loop is used when the looping process terminates because a specified condition is satisfied, and thus the **number of passes is not known in advance**.
- The structure of a while loop


```

while logical expression
statements
end
      
```
- For the while loop to function properly, the following two conditions must occur:
 1. The **loop variable** must have a **value** before the while statement is executed.
 2. The **loop variable** must be **changed** somehow by the statements.

Loops

While loop

- A simple example of a while loop is


```

x = 5;
while x < 25
    disp(x)
    x = 2*x - 1;
end
      
```
- The results displayed by the `disp` statement are 5, 9, and 17.

Loops

Infinite Loops

- You can create an infinite while loop which is a loop that never ends:

```
x = 8;
while x ~=0
    x = x -3;
end
```

- Within the loop the variable x takes on the values 5, 2, -1, -4....., and the condition $x \neq 0$ is always satisfied, so the loop never stops.

37

Loop Vectorization

- We can often avoid the use of loops and branching and thus create simpler and faster programs by using a logical array as a mask that selects elements of another array. This called **loop vectorization**.

- Example** Here is one way to compute the sine of 1001 values ranging from 0 to 10:

```
i = 0;
for t = 0:.01:10
    i = i + 1;
    y(i) = sin(t);
end
```

A vectorized version of the same code is

```
t = 0:0.01:10;
y = sin(t);
```

- The second example executes much faster than the first and is the way Matlab is meant to be use

38

Loop Vectorization

- Example:** for the array $A = [0, -1, 4, 9, -14, 25, -34, 49, 64]$, find all the elements that are greater than or equal to 0.

- Using loops

```
[m,n] = size(A) ;
C = zeros(m,n) ;
for k = 1:m*n
    if A(k) >= 0
        C(k) = A(k)
    end
end
```

- Using **array masking and relational operators** to vectorize the loop

```
C = A ;
C(C<0) = 0 ;
```

39

The Switch Structure

- The **switch structure** provides an alternative to using the if, elseif, and else commands. Anything programmed using switch can also be programmed using if structures. However, for some applications the switch structure is more readable than code using the if structure.

40

The Switch Statement

```
switch input expression
  case value1
    statement group 1
  case value2
    statement group 2
  .
  .
  .
  otherwise
    statement group n
end
```

41

The Switch Statement

- **Example**
- The following switch block displays the point on the compass that corresponds to that angle.

```
switch angle
  case 45
    disp('Northeast')
  case 135
    disp('Southeast')
  case 225
    disp('Southwest')
  case 315
    disp('Northwest')
  otherwise
    disp('DirectionUnknown')
end
```

42

Program Design and Development

Finding Bugs

- Debugging a program is the process of finding and removing the “bugs,” or errors, in a program. Such errors usually fall into one of the following categories.
 1. **Syntax errors** such as omitting a parenthesis or comma, or spelling a command name incorrectly. Matlab usually detects the more obvious errors and displays a message describing the error and its location.
 2. **Runtime errors**; These errors occur due to an **incorrect mathematical procedure**. They do not necessarily occur every time the program is executed; their occurrence often depends on the particular input data. A common example is division by zero.

43

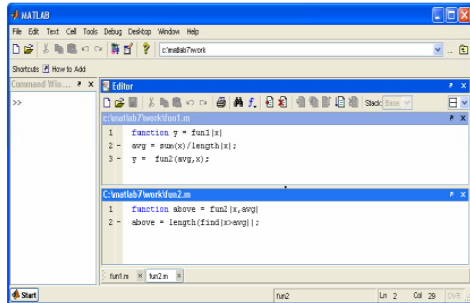
Program Design and Development

Locate Errors

1. Always test your program with a simple version of the problem, whose answers can be checked by hand calculations.
2. Display any intermediate calculations by removing semicolons at the end of statements.
3. To test user-defined functions, try commenting out the function line and running the file as a script.
4. Use the debugging features of the Editor/Debugger.

44

Matlab Editor/Debugger



- Menus: File, Edit, text, Tools, Debug, Desktop, Window, Help

45

The Debug Menu

- Breakpoints
 - Set/Clear Breakpoint
 - Step
 - Step In
 - Step Out
 - Continue
 - Go Until Cursor
 - Exit Debug Mode
 - Enable/Disable Breakpoint
 - Stop if Errors/Warnings

46

The Debug Menu

Example

Two functions that find the values of a vector **x** that are greater than the average values of all elements in **x**

```
% fun1.m
function y = fun1(x)
avg = sum(x) / length(x);
y = fun2(avg,x)
```

```
%fun2.m
function fun2(x,avg)
above = length(find(x>avg));
```

```
>> % Test the functions
>> x = [1,2,3,4,10];
>> y = fun1(x)
y =
     3
```

ERROR !!! Answer should be 1 !!!
Let's see how we can debug

47

Strings

- A *string* is a variable that contains characters.
- Strings are useful for creating input prompts and messages and for storing and operating on data such as names and addresses.
- To create a string variable, enclose the characters in single quotes.
- For example, the string variable name is created as follows:

```
>>name = 'Leslie Student'
name =
    Leslie Student
```

48

Strings and the Input Command

- The input function, whose syntax is

```
x = input('prompt', 's')
```

displays the string prompt on the screen, waits for input from the keyboard, and returns the entered value in the string variable x.

49

Concatenating Strings

- Given strings S1, S2, and S3, a string S that is composed of appending the three strings can be formulated by S = [S1 S2 S3] concatenates character

- **Example**

```
>> S1 = 'CPE' ;  
>> S2 = '0907311' ;  
>> S = [S1 S2]  
S =  
    CPE0907311
```

50

Strings Conversion

- Use the `num2str()` function to convert numeric values in strings.
- Use the `str2num()` function to convert numeric values that are stored as strings to numbers.

- **Example**

```
>> d = 1.23;  
>> class(d)  
ans = double  
>> f = num2str(d)  
f = 1.23  
>> class(f)  
ans = char
```

51