MATLAB
-Loops, Branches, and Control Flow

PhD Course Work – 2014

Instructor: **Dr. G. L. Prajapati**
IET-DAVV

Flow Control Constructs

- Logic Control:
  - IF / ELSEIF / ELSE
  - SWITCH / CASE / OTHERWISE

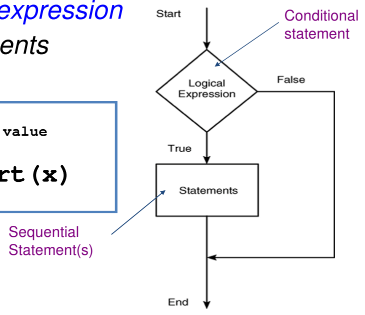
  Objective is to use these conditional statements to develop logical program flow.
- Iterative Loops:
  - FOR
  - WHILE

  Objective is to control repetitions of the execution of a particular group of statements until certain condition holds, or specified number of times.

Syntax of the `if` statement:

if *logical expression*
  statements
end

```
Example:
x = some given value
if x >= 0
   y = sqrt(x)
end
```

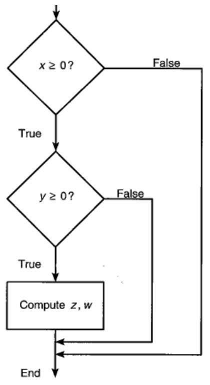


Nested "if" statements:

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

Nested Statement

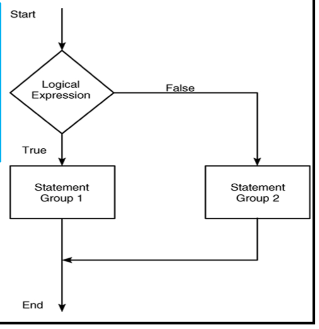
- Note the indentions



THE `else` STATEMENT:

If two mutually exclusive actions can occur as a result of a decision, use the **`else`** statement.

```
if logical expression
  statement group1
else
  statement group2
end
```

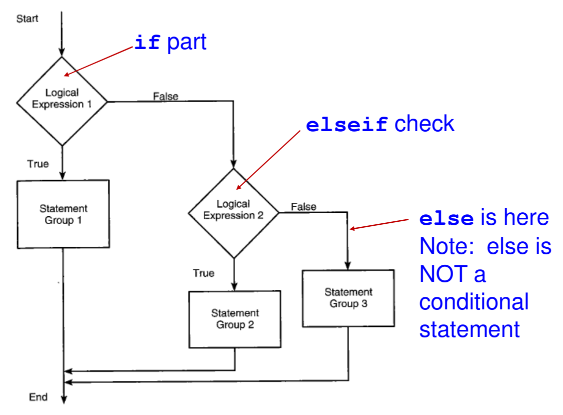


The `elseif` statement:

When three actions can occur as a result of a decision, the **`else`** and **`elseif`** statements are used along with the **`if`** statement.

Remember: ONLY ONE ACTION WILL ACTUALY OCCUR!!!

```
if logical expression 1
  statement group1
elseif logical expression 2
  statement group 2
else
  statement group 3
end
```



IF / ELSEIF / ELSE Structures

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

Does the order that I check things matter? YES!

The if, elseif and else statements

```
I=9;
J=8;
if I == J
   disp('equal');
   else if I<J
        disp('I is less than J');
      else
        disp('I is greater than J');
      end %else if
end %if
Output: I is greater than J
```

```
I=9;
J=8;
if I == J
   disp('equal');
elseif I<J
   disp('I is less than J');
else
   disp('I is greater than J');
end %if

Output: I is greater than J
```

- Works on Conditional statements
- Logic condition is ‘true’ if its different then 0.
- ELSEIF does not need a matching END, while ELSE IF does.

More on if-elseif-else-end

Syntax for the if-elseif-else-end construct if there are more than three alternatives:

```
if logical expression1
   Statements group1
elseif logical expression2
   Statements group2
elseif logical expression3
   Statements group3
elseif logical expression4
   Statements group4
…
else
   Statement if all other cases are false
end
```

Switch Case

- switch…end. Most general form:

```
switch variable/expression
  case value1
    statement group 1
  case value2
    statement group 2
  otherwise
    default case
end
```

Name of a variable /expression containing the value of the switch parameter

Possible values that the switch parameter might have

optional

Switch expression can be numeric or a string constant

Switch, Case, and Otherwise

- **More efficient** than elseif statements
- Only the first matching case is executed

```
switch input_num
case -1
  input_str = 'minus one';
case 0
  input_str = 'zero';
case 1
  input_str = 'plus one';
case {-10,10}
  input_str = '+/- ten';
otherwise
  input_str = 'other value';
end
```

Problem

- Build a program which receives a variable x and its units

  (mm, cm, inch, meter) and calculates Y- it's value in centimeters units.
- Use switch case.
- 1 Inch = 2.54 cm
- Write a comment for error case.
- Save the file under units.m

Solution

```
x = 3.0;
units = 'mm';
switch units
case {'in','inch'}
   y = 2.54*x        % converts to centimeters
case {'m','meter'}
   y = x*100          % converts to centimeters
case { 'millimeter','mm'}
        y = x/10;
      disp    ([num2str(x)  '   in   ' units ' converted
   to cm is :' num2str(y)])
case {'cm','centimeter'}
     y = x
otherwise
     disp    (['unknown units:' units])
       y = nan;
end
```

If many values of **switch variable**
Should cause the same code to execute, all of those values may be included in a single block by enclosing them in brackets, as in this example

The *for* Loop in MATLAB

- Used when you want the calculations to be performed a defined number of times

The *for* Loop in MATLAB

- In MATLAB, a *for* loop begins with the statement indicating how many times the statements in the loop will be executed
- A counter is defined within this statement
- Examples:

  `for k = 1:100`

  (counter = *k*, the loop will be executed 100 times)

  `for i = 1:2:7`

  (counter = *i*, the counter will be incremented by a value of 2 each time until its value reaches 7. Therefore, the loop will be executed 4 times (*i* = 1,3,5, and 7)

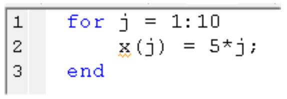
The *for* Loop in MATLAB

Note the following rules when using for loops with the <u>loop variable expression k = m:s:n:</u>

- 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.

The *for* Loop in MATLAB

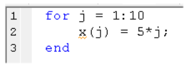
- The loop ends with an *end* statement
- In M-files, the MATLAB editor will automatically indent text between the *for* and *end* statements:



```
for j = 1:10
    x(j)  = 5*j;
end
```

- Determine what the variable *x* will be after running this M-file?

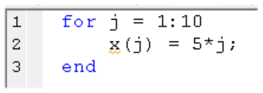
for Loop Example



```
for j = 1:10
    x(j) = 5*j;
end
```

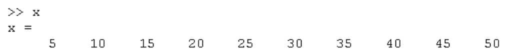
- The first time through the loop, *j* = 1
- Because of the single value in parentheses, *x* will be a one-dimensional array
- *x*(1) will be set equal to 5*1 = 5
- The second time through the loop, j = 2
- *x*(2) will be set equal to 5*2 = 10
- This will be repeated until *j* = 10 and *x*(10) = 50

for Loop Example



```
for j = 1:10
    x(j) = 5*j;
end
```

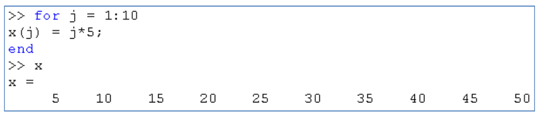
- *x* will be a one-dimensional array (a row matrix) with 10 elements:



```
>> x
x =
     5    10    15    20    25    30    35    40    45    50
```

for Loop in Interactive Mode

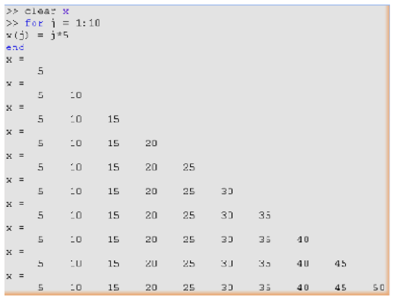
- Loop commands can be entered directly from the command prompt
- The calculations are not performed until the end statement is entered



```
>> for j = 1:10
x(j) = j*5;
end
>> x
x =
     5    10    15    20    25    30    35    40    45    50
```

for Loop in Interactive Mode

- Remember that if you leave off the semi-colon, the results of the calculations will be written to the screen in every loop:



```
>> clear x
>> for j = 1:10
x(j) = j*5
end
x =
     5
x =
     5    10
x =
     5    10    15
x =
     5    10    15    20
x =
     5    10    15    20    25
x =
     5    10    15    20    25    30
x =
     5    10    15    20    25    30    35
x =
     5    10    15    20    25    30    35    40
x =
     5    10    15    20    25    30    35    40    45
x =
     5    10    15    20    25    30    35    40    45    50
```

for Loop Examples

- What result will be output to the screen in each of the following examples?

```
y = 0;
for k = 1:5
   y = y + k;
end
y
```

```
y =
    15
```

for Loop Examples

```
y = 0;
for k = 2:2:8
    y = y + k;
end
y
```

```
y =
    20
```

for Loop Examples

```
for k = 1:5
    y(k)=k^2;
end
y
```

```
y =
     1     4     9    16    25
```

for Loop Examples

```
for j = 1:3
    for k = 1:3
        T(j,k) = j*k;
    end
end
T
```

```
T =
     1     2     3
     2     4     6
     3     6     9
```

for Loop Example

- Consider this equation:

$$y = 2^{0.4x} + 5$$

- Plot this equation for values of *x* from -10 to 10
- Use a *for* loop to calculate and store *x* and *y* values in one-dimensional arrays

for Loop Example

$$y = 2^{0.4x} + 5$$

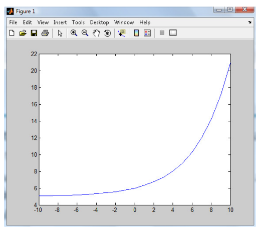
```
for i = 1:21
    x(i) = -10 +(i-1);
    y(i) = 2^(0.4*x(i)) + 5;
end
```

- After running these lines of code, two one-dimensional arrays, *x* and *y*, have been created, each with 21 elements

Plot Command

- The stored arrays can be plotted with the command:

  `plot(x,y)`
- Any two one-dimensional arrays can be plotted, as long as they are exactly the same size
- The plot will be created in a new window



Accessing an Array

- Loops can also sample data from the cells of a pre-existing array. Suppose:

```
hours=[40, 65, 48, 30, 20];
totalOvertime = 0;
for k = 1:length(hours) % length(X) returns the size of the longest dimension of X.
  if  hours(k) > 40
      totalOvertime=totalOvertime + hours(k) - 40;
  end
end
totalOvertime
```

(result is 33)

Accessing an Array

- Loops can also sample data from the cells of a pre-existing array. Suppose:

```
hours=[40, 65, 48, 30, 20];
totalOvertime = 0;
for k = hours
   if  k > 40
         totalOvertime=totalOvertime + k - 40;
   end
end
totalOvertime
```

(result is 33)

Accessing an Array

- Loops can also sample data from the cells of a pre-existing array. Suppose:

```
totalOvertime = 0;
     for k = [40, 65, 48, 30, 20]
        if  k > 40
              totalOvertime=totalOvertime + k - 40;
        end
     end
     totalOvertime
```

(result is 33)

Accessing an Array

```
for k = [40, 65, 48, 30, 20]
        disp('Hello');
End
```

Output:

```
Hello
Hello
Hello
Hello
Hello
```

Accessing an Array

```
for k = [40, 65, 48, 30, 20]
        disp(k);
End
```

Output:

```
40
65
48
30
20
```

Accessing an Array

```
for k = [40, 65, 48, 30, 20; 35, 55, 65, 75, 85]
        disp('Hello');
End
```

Output:

```
Hello
Hello
Hello
Hello
Hello
```

Accessing an Array

```
for k = [40, 65, 48, 30, 20; 35, 55, 65, 75, 85]
        disp(k);
End
```

Output:

```
40
35

65
55

48
65

30
75

20
85
```

Accessing an Array

```
s=0;
for k = [40, 65, 48, 30, 20; 35, 55, 65, 75, 85]
    s=s+k;
End
```

Output:

```
s =

203
315
```

while Loops in MATLAB

- A *for* loop will be executed a fixed number of times
- A *while* loop will continue to be repeated until some condition is satisfied

Syntax:

```
while expression
      statements
end
```

expression is a MATLAB expression that evaluates to a result of logical 1 (true) or logical 0 (false).
If *expression is true then statements will be executed*

Example 1:

```
k=20;
while k==20
   disp(k);
   k=k-1;
End
```

Output: 20

while loop: Example 2

- What are the values of *A* and *m* after execution of these MATLAB commands:

```
m = 0;
A = 20;
while A <= 50
    A = A + 5;
    m = m + 1;
end
A
m
```

```
A =
       55
m =
        7
```

while loop: Example 3

- What are the values of *A* and *m* after execution of these MATLAB commands:

```
m = 0;
A = 100;
while A > 15
    A = A/2;
    m = m + 1;
end
A
m
```

```
A =
   12.5000
m =
     3
```

while loop: Example 4

```
A = [40, 65, 48, 30, 20; 35, 55, 65, 75, 85];
i=0;
while A
   i=i+1;
   disp('Expression Evaluates Logical 1 (true)');
   if i==3
      A(2,3)=0;
   end
end
```

Output:

```
Expression Evaluates Logical 1 (true)
Expression Evaluates Logical 1 (true)
Expression Evaluates Logical 1 (true)
```

while loop: Example 5

```
A = [40, 65, 48, 30, 20; 35, 55, 65, 75, 85];
i=0;
while all(A)
   i=i+1;
   disp('Expression Evaluates Logical 1 (true)');
   if i==3
      A(2,3)=0;
   end
end
```

all: Determines whether all array elements are nonzero

Output:

```
Expression Evaluates Logical 1 (true)
Expression Evaluates Logical 1 (true)
Expression Evaluates Logical 1 (true)
```

Infinite Loops

- When using a while loop, there is a danger of encountering an infinite loop
- Since termination of the loop is dependent upon achieving some condition (in this case, a balance of less than or equal to zero), it is possible that the condition will never be reached, and therefore the looping will continue endlessly

Infinite Loops: Example 1

```
A = [40, 65, 48, 30, 20; 35, 55, 65, 75, 85];
while all(A)
    disp('Expression Evaluates Logical 1 (true)');
end
```

Output:

```
Expression Evaluates Logical 1 (true)
Expression Evaluates Logical 1 (true)
Expression Evaluates Logical 1 (true)
.
.
.
```

Infinite Loops: Example 2

- What are the values of *A* and *m* after execution of these MATLAB commands:

```
m = 0;
A = 10;
while A > 0
    A = sqrt(A);
    m = m + 1;
end
A
m
```

Infinite Loop: the square root will never reach zero

break statement in Matlab

- **Syntax**

  break
- **break** exits a loop. Used to halt a certain process, like a file read of unknown length.
- **break** terminates execution of for or while loop
- break terminates the execution of a for or while loop. Statements in the loop that appear after the break statement are not executed.
- In nested loops, break exits only from the loop in which it occurs. Control passes to the statement that follows the end of that loop.

Example of break

```
for i = 1:5
  if i== 3
    break;
  end
  fprintf('i = %d\n', i);
  end
disp('End of loop!');
```

Output:

```
i = 1
i = 2
End of loop!
```

continue statement in Matlab

- **Syntax**

  continue
- **continue** jumps to the end of a loop, but keeps on iterating. Used to "skip" bad values (like divide by zero).
- **Continue** passes control to next iteration of for or while loop
- continue passes control to the next iteration of the for or while loop in which it appears, skipping any remaining statements in the body of the loop. The same holds true for continue statements in nested loops. That is, execution continues at the beginning of the loop in which the continue statement was encountered.

Example of continue

```
for i = 1:5
  if i== 3
     continue;
  end
  fprintf('i = %d\n', i);
  end
disp('End of loop!');
```

Output:

```
i = 1
i = 2
i = 4
i = 5
End of loop!
```

Thank You All

3-D plots in next lecture

- Dr. G. L. Prajapati