
MATH2089

Numerical Methods

Lecture 2

Applied **Matlab** Programming

Dr Victoria Timchenko

School of Mechanical and Manufacturing
Engineering

Variables

- To define a variable **x** we might type
`>>x=3`
- Should be read as: **x** is assigned a value of 3
- We are telling the machine to store the value on the right hand side of the equation in a memory location, and to name that location **x**
`>>x=x+2 ?`

Naming Variables

- All names must start with a letter
 - They may contain letters, numbers and the underscore (_)
 - Names are case sensitive
 - There are certain keywords you can't use
 - Use the keyword function for a list of keywords
-

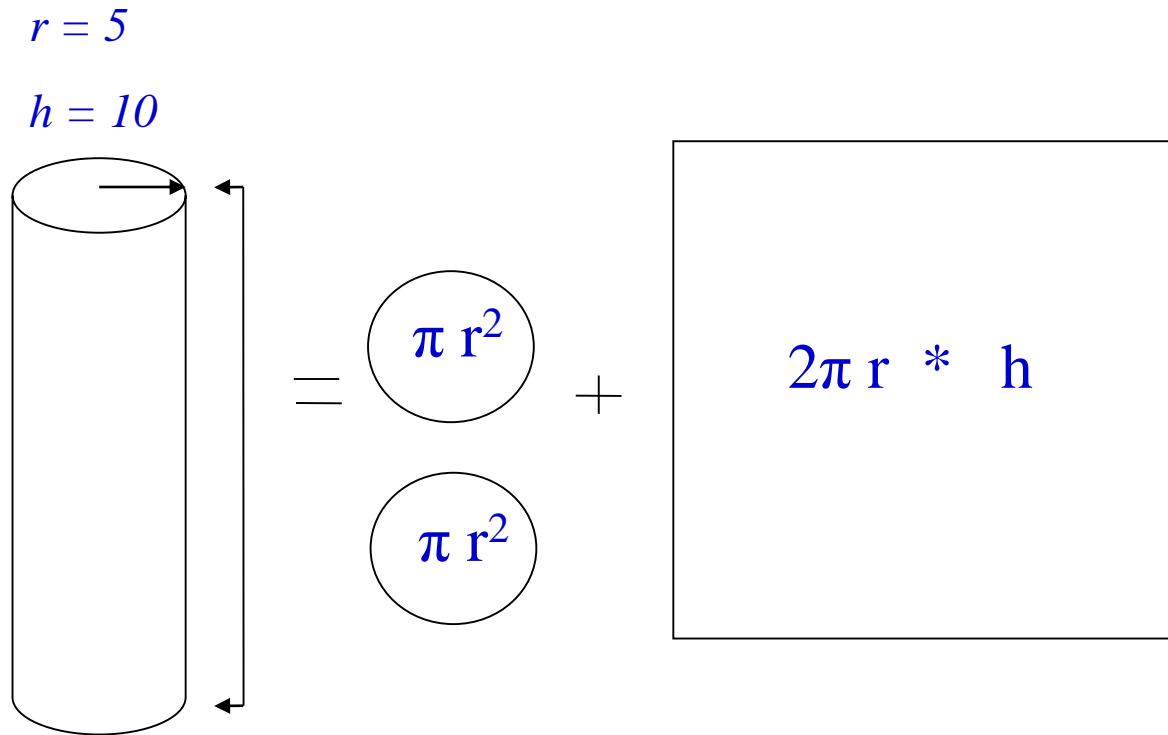
Which of these names are allowed in **Matlab**?

- test
- Test
- ~~if~~
- my~~x~~book
- my_book
- Thisisoneverylongnamebutisitstillallowed?
- ~~1~~stgroup
- group_one
- zzaAbc
- z34wAwy~~x~~12~~#~~
- sin } **bad**
- log } **idea**

Order of Operation

- Same as you've learned in math class
- Same as your calculator
 - ❑ Parentheses first
 - ❑ Exponentiation
 - ❑ Multiplication / division
 - ❑ Addition / subtraction

Calculate the Surface Area of a Cylinder



$$SA = 2\pi r^2 + 2\pi rh = 2\pi r(r + h)$$

Matrices in **Matlab**

All **Matlab** variables are matrices:

- A vector is a matrix with one row or one column
- A scalar is a matrix with one row and one column
- A character string is a row of column vector of characters

Consequences:

- Rules of linear algebra apply to addition, subtraction and multiplication
- Elements in the vectors and matrices are addressed as $x(2)$, $A(4,5)$. Usually this notation is clear from context, but it can be confused with a function call

Creating Arrays

>>x=[m:q:n] first element m, step q, last element n

>>x=[1:2:13]

x =

1 3 5 7 9 11 13

>>x=linspace(xi, xf, n)

first element xi, last element xf, number of elements n

>>x=linspace(0,8,6)

x =

0 1.6000 3.2000 4.8000 6.4000 8.0000

Creating matrices

- A matrix is created by assigning the elements of the matrix to a variable
- $A = [1\text{st row elements}; 2\text{nd row elements}; \dots \text{last row elements}]$
- For example:
- $A = [5, 35, 43; 4, 76, 81; 21, 32, 40]$
- Commands `ones(m, n)` and `zeros(m, n)` – creates a matrix with m rows and n columns, in which all elements are the numbers 0 or 1 respectively
- Command `eye (n)` creates a square matrix with n rows and n columns in which diagonal elements are equal to 1
- The transpose operator when applied to a vector, switches a row (column) vector to a column (row) vector

Use colon to refer to an entire column or row

➤ `>> A = [1 2 3; 4 5 6; 7 8 9];`

➤ `>> A(:,1)`

➤ `ans =`

➤ `1`

➤ `4`

➤ `7`

➤ `>> A(2,:)`

➤ `ans =`

➤ `4 5 6`

Strings

Strings are matrices with character elements

- String constants are enclosed in single quotes
- Colon notation and subscript operations apply

For example:

```
>>name = 'John'
```

```
>>whos name
```

Name	Size	Bytes	Class
name	1x4	8	char array

Grand total is 4 elements using 8 bytes

String Manipulation

Function	Operation
<code>char</code>	Converts an integer to the character using ASCII codes, or combines characters into a character matrix
<code>findstr</code>	Finds one string in another string
<code>length</code>	Returns the number of characters in a string
<code>num2str</code>	Converts a number to string
<code>str2num</code>	Converts a string to a number
<code>strcmp</code>	Compares two strings
<code>strmatch</code>	Identifies rows of a character array that begin with a string
<code>strncmp</code>	Compares the first n elements of two strings
<code>sprintf</code>	Converts strings and numeric values to a string

Mathematical Operations with Arrays

- Addition and subtraction are element-by-element operations
- Multiplication is executed according to the rules of linear algebra
- **Note!** $A*B$ can be performed only if number of the columns in Matrix A is equal to the number of rows in matrix B

```
>> A=[1 2 3; 4 5 6];
```

```
>> B=A*A'
```

```
B =
```

```
14 32
```

```
32 77
```

Mathematical Operations with Arrays

Division and Matrix Power are also associated with the rules of linear algebra

- To calculate the power of a matrix, one can use the matrix power operator \wedge .
- `>> A=[1 2; 3 4];`
- `>> A^3`
- `ans =`
- `37 54`
- `81 118`
- The above example calculates the 3rd power of the matrix

“.” operator

For element-by-element multiplication, division and exponentiation of two vectors or matrices use a period in front of arithmetic operator.

- .* element-by-element multiplication
- ./ element-by-element “right” division
- .\ element-by-element “left” division
- .^ element-by-element exponentiation
- >>x= [5 10 15];
- >> y= [10 20 30];
- x.*y
- ans = 50 200 450

In each case the size of the arrays must match

Command Window

 New to MATLAB? Watch this [Video](#), see [Demos](#), or read [Getting Started](#).

```
>> a=[1,2 ,3]
```

a =

```
     1     2     3
```

```
>> b=a+5
```

b =

```
     6     7     8
```

```
>> c=a.*b
```

c =

```
     6    14    24
```

```
>> c=a*b
```

```
??? Error using ==> mtimes
Inner matrix dimensions must agree.
```


Vectorization

- Vectorization is the use of single, compact expressions that operate on all elements of a vector without explicitly writing the code for a loop
- Most built-in functions support vectorized operations

For example:

- `>> x = 0:pi/4:pi` (define a row vector)
- `x =`
- `0 0.7854 1.5708 2.3562 3.1416`
- `>> y = cos(x)` (evaluate cosine of each `x(i)`)
- `y =`
- `1.0000 0.7071 0 -0.7071 -1.0000`

Presenting Results

- In **Matlab**, all plots are displayed through figure window using **figure ()**
- A figure window will be created automatically when you generate plots
- **Matlab** will assign a figure number to each figure window. One can use the figure number to make active the desired figure
- For example, if one wants to active the 3rd figure window, one can enter:
- `>> figure(3)`

Plots

- Two dimensional plots are created with the plot function `plot ()`

Syntax:

```
plot(x,y)
```

```
plot(xdata,ydata,symbol)
```

```
plot(x1,y1,x2,y2,...)
```

```
plot(x1,y1,symbol1,x2,y2,symbol2,...)
```

Note! `x` and `y` must have the same type, `x1` and `y1` must have the same type, `x2` and `y2` must have the same type, etc.

Line and Symbol Types in Plotting

Color		Symbols				Line	
y	yellow	.	point	^	triangle (up)	-	solid
m	magenta	o	circle	<	triangle (left)	:	dotted
c	cyan	x	x-mark	>	triangle (right)	-.	dashdot
r	red	+	plus	p	pentagram	--	dashed
g	green	*	star	h	hexagram		
b	blue	s	square				
w	white	d	diamond				
k	black	v	triangle (down)				

Example: `plot(x,y,'yo')`

Multiple plots per figure window

- The **subplot** function is used to create a matrix of plots in a single figure window.

Syntax:

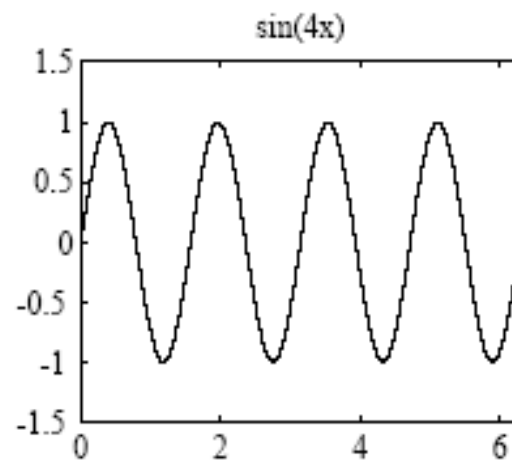
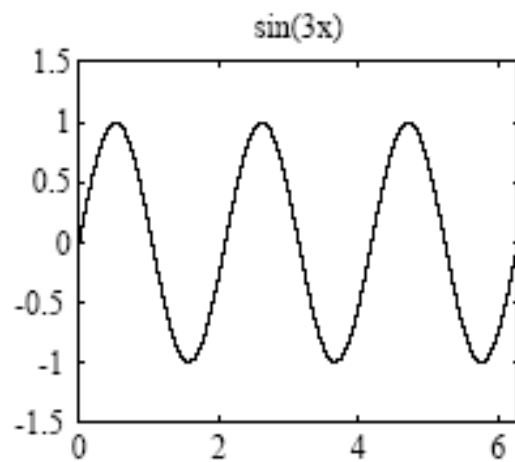
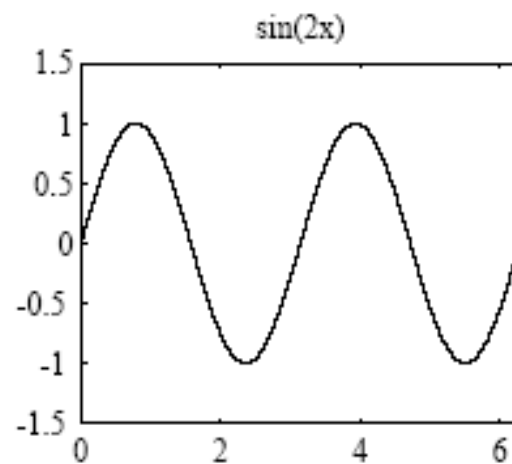
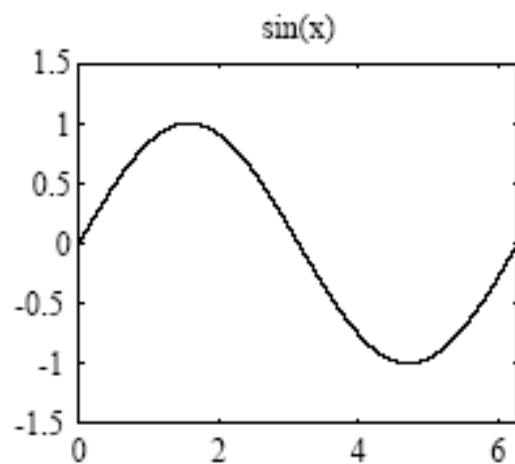
```
subplot(nrows, ncols, thisPlot)
```

Multiple plots per figure window (con't)

Example:

```
>> x = linspace(0,2*pi);  
>> subplot(2,2,1);  
>> plot(x,sin(x)); axis([0 2*pi -1.5 1.5]); title('sin(x)');  
>> subplot(2,2,2);  
>> plot(x,sin(2*x)); axis([0 2*pi -1.5 1.5]); title('sin(2x)');  
>> subplot(2,2,3);  
>> plot(x,sin(3*x)); axis([0 2*pi -1.5 1.5]); title('sin(3x)');  
>> subplot(2,2,4);  
>> plot(x,sin(4*x)); axis([0 2*pi -1.5 1.5]); title('sin(4x)');
```

Multiple plots per figure window (con't)

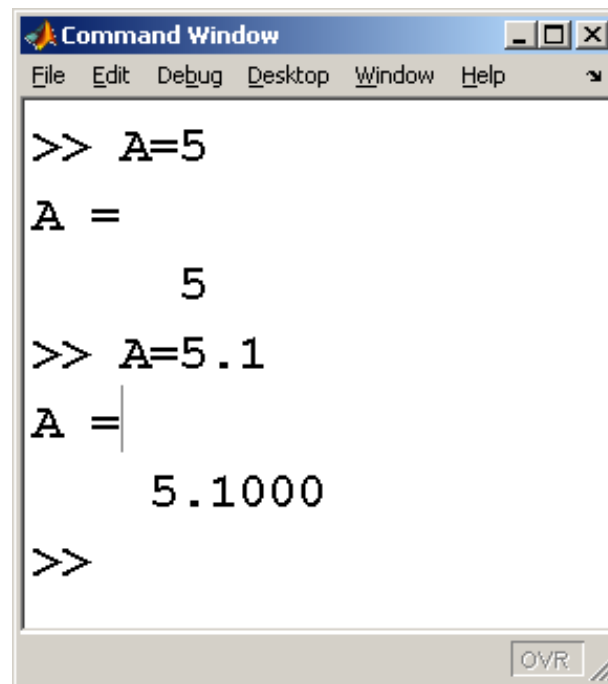


Display Format

- Multiple display formats are available
 - No matter what display format you choose, **Matlab** uses double precision floating point numbers in its calculations
 - **Matlab** handles both integers and decimal numbers as floating point numbers
-

Default

- The default format is called short
- If an integer is entered it is displayed without trailing zeros
- If a floating point number is entered four decimal digits are displayed

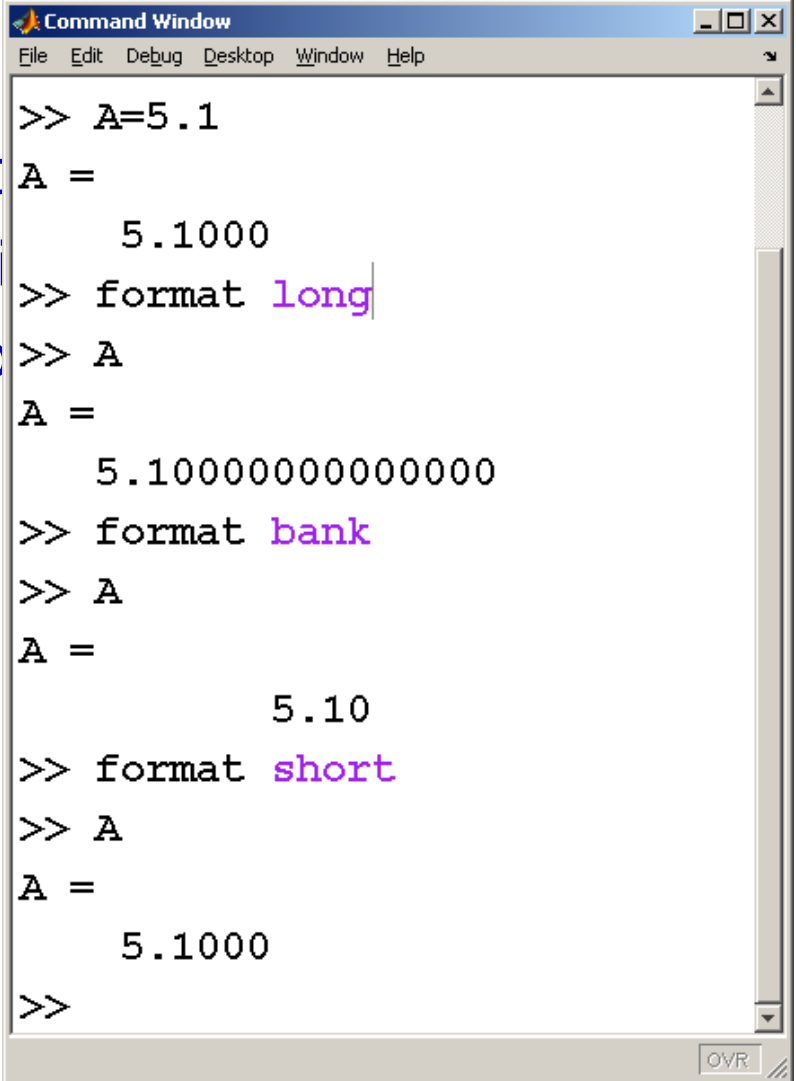


```
Command Window
File Edit Debug Desktop Window Help
>> A=5
A =
    5
>> A=5.1
A =
    5.1000
>>
```

The image shows a screenshot of the MATLAB Command Window. The window has a title bar that says "Command Window" and a menu bar with "File", "Edit", "Debug", "Desktop", "Window", and "Help". The command history shows two assignments: first, `A=5`, which results in `A = 5`; second, `A=5.1`, which results in `A = 5.1000`. The prompt `>>` is shown at the end of the second command. At the bottom right of the window, there is a button labeled "OVR".

Other Formats

- Changing the format affects all
 - format long results in 14 decimal digits
 - format bank results in 2 decimal digits
 - format short returns the display format of the workspace

A screenshot of the MATLAB Command Window. The window has a title bar 'Command Window' and a menu bar with 'File', 'Edit', 'Debug', 'Desktop', 'Window', and 'Help'. The command history shows: 1. '>> A=5.1' followed by 'A =' and '5.1000'. 2. '>> format long' followed by '>> A' and 'A =' and '5.1000000000000000'. 3. '>> format bank' followed by '>> A' and 'A =' and '5.10'. 4. '>> format short' followed by '>> A' and 'A =' and '5.1000'. The window ends with '>>'. An 'OVR' button is visible in the bottom right corner.

```
>> A=5.1
A =
    5.1000
>> format long
>> A
A =
    5.1000000000000000
>> format bank
>> A
A =
    5.10
>> format short
>> A
A =
    5.1000
>>
```

Script M-files

- This command window is good for testing ideas and running sequences of operations contained in functions
 - If you want to save your work, you need to create an M-file
 - File->New->M-file
 - Type your commands in the edit window that opens
-

Functions

- *Function: is a piece of computer code that accepts an input, performs calculation and provides an output.*

For example

```
function  c=aplusb(a,b)  
    c=a+b;
```

Input parameters allow the same calculation procedure (same algorithm) to be applied to different data. Thus, function m-files are reusable.

Specific tasks can be encapsulated into functions. This modular approach enables development of structured solutions to complex problems

-
- The syntax of using the command function is:
where `function [OUT1,OUT2,...] = FUN(IN1,IN2,...)`
 - IN1, IN2,... are the list of required input parameters
 - OUT1,OUT2,... are the list of required output parameters
 - FUN is the name of the function (should be equal to the M-file name, but it is not necessary!)
 - The command function must be at the first line of your code
 - Example: `function c=aplusb(a,b)`
-

The required input parameters must be given when one calls the function. Otherwise, **Matlab** will return error message. For example:

```
>> aplusb(1)
??? Input argument 'b' is undefined.
Error in ==> E:\matlab\apusb.m
On line 2 ==> c=a+b;
```

One can run a function without specifying output parameter(s); no error message will be displayed. However, if the function consists of two or more output parameters, only the first will be returned

Example: motion(t)

Local and Global Variables

- Functions use local variables that exist only while the function is executing. Local variables are distinct from variables of the same name in the workspace or in other functions.
- **If one wants to share variables between functions, one can use the command `global`. Once a variable is defined as a global variable, its value will be placed in the global memory area. Then, all functions can get access to this variable**

Functions can call other Functions

➤ function disp_my_plot

Text output from Functions

- Output to the command window is achieved with either the ***disp*** function or the ***fprintf*** function. Output to a file requires the ***fprintf*** function
- ***disp*** Simple to use. Provides limited control over appearance of output

Syntax:

`disp(outval)`

where outval is either a string matrix or a numeric matrix

For example: Numeric output

```
>> disp(5)
```

5

```
>> x = 1:3; disp(x)
```

1 2 3

The num2str function is often used to with the ***disp*** function to create a labeled output of a numeric value

Syntax:

`stringValue = num2str(numericValue)`

Converts `numericValue` to a string representation of that numeric value

The `disp(['x = ', num2str(x)])` construct works when `x` is a row vector, but not when `x` is a column vector or matrix

Example: `outstring.m`

The *fprintf* command

Provides total control over appearance of output

Syntax: `fprintf(outFormat,outVariables)`

`fprintf(fileHandle,outFormat,outVariables)`

- Uses the outFormat string to convert outVariables to strings that are printed. In the first line the output is displayed in the command window. In the second one, the output is written to a file referred to by the fileHandle
 - The outFormat string specifies how the outVariables are converted and displayed. The outFormat string can contain any text characters. It also must contain a conversion code for each of the outVariables. The following table shows the basic codes
-

Code	Conversion instruction
%s	format as a string
%d	format with no fractional part (integer format)
%f	format as a floating-point value
%e	format as a floating-point value in scientific notation
%g	format in the most compact form of either %f or %e
\n	insert newline in output string
\t	insert tab in output string

For example

voltage=3.5

```
Fprintf('The voltage is %8.2f millivolts \n', voltage);
```

Returns: The voltage is 3.50 millivolts

The *feval* command

- The *feval* (function evaluate) command evaluates the value of a function for a given value (or values) of the function's argument (or arguments)

Syntax:

Variable=feval('function name', argument value)

```
>> feval('sqrt', 64)
```

```
ans =
```

```
8
```

```
>> x = feval('sin', pi/6)
```

```
x=0.5
```

Programming in Matlab

Similar to other programming languages, Matlab has flow control structures for:

- Repetition: looping or iteration
 - Conditional execution : branching
 - Comparison
-

Relational operators

Relational operators are used in comparing two values.

Operator	Meaning
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
~=	not equal to

The result of applying a relational operator is a logical value, i.e. the result is either true or false.

For example:

$$y = (6 < 10) + (7 > 8) + (5 * 3 == 60 / 4)$$
$$y = 2$$

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

The arithmetic operations (+, -, *, /, \) have precedence over relational operations.
The relational operators have equal precedence, evaluated from left to right.

$$3 + (4 < 16) / 2$$

Ans=3.5

Logical functions

The primary logical function is *find*.

It can be used in place of both traditional selection structures.

It searches a matrix and identifies which elements in the matrix meet a given criterion

```
Height=[63, 67, 65, 72, 69, 78, 75]
accept=find(height>=66)
accept=
     2     4     5     6     7
```

For Loop

The syntax of using for loop is:

for VARIABLE = EXPR, STATEMENT, ..., STATEMENT, end

The function disp() is used to display a value.

In an M-file, one can write the *for loop* in different lines as:

```
for ii = 0:0.1:5
    disp(ii);
    a(ii+1)=ii;
end
```

OR

```
for ii = 5:-0.1:0
    disp(ii);
end
```

While Loop

The syntax of *while loop* is:

```
while EXPRESSION, STATEMENT, ... STATEMENT, end
```

```
    ii = 0;  
    while ii <= 5  
        disp(ii);  
        ii = ii + 1;  
    end
```

One advantage of the *while loop* when compared with the *for loop* is that the loop counter is not necessarily going in one direction. That is, it is possible to have some statements within the loop to increase, decrease, or even reset the counter.

If condition

The syntax of the if statement is:

```
if EXPRESSION  
  STATEMENT, ...  
elseif EXPRESSION  
  STATEMENT, ...  
  ...  
else  
  STATEMENT, ...  
end
```

The elseif, and else parts are optional..

Switch and Case

The switch/case structure is often used when a series of programming path options exists for a given variable, depending on its value...

```
city = input('Enter the name of a city in a single quotes:');  
    switch city  
    case 'Melbourne'  
        disp('QF345')  
    case 'Perth'  
        disp('QF1500')  
    otherwise  
        disp('information is not available')  
    end
```

Break command

The break command is used to terminate the execution of a *for* or a *while* loop.

It will not stop the program but just terminate the loop.

```
for ii = 1:5
    disp(ii);
    if ii == 3
        break
    end
end
disp(['ii stopped at ' num2str(ii)])
```