AYDIN
ADNAN MENDERES UNIVERSITY
FACULTY OF ENGINEERING
Department of Electrical and Electronics Engineering

EE214 – Computer Aided Engineering Applications

Prof. Dr. Yılmaz KALKAN

EE214 – Computer Aided Engineering Applications

Lecture 4

Working With M–files and Program Flow Control

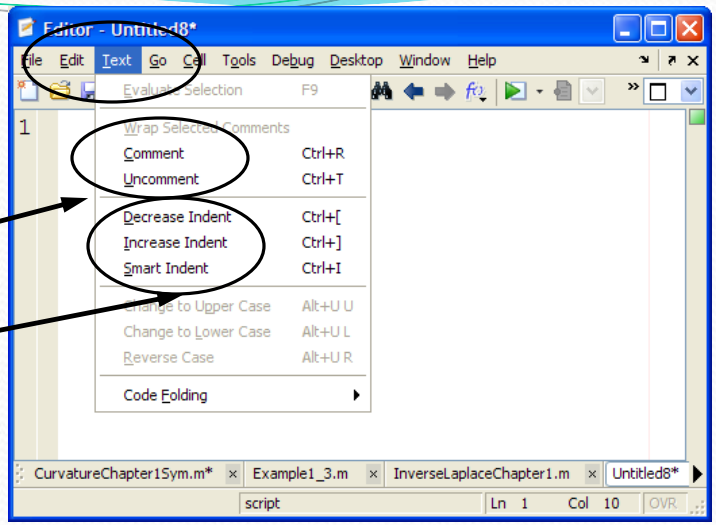
Creating Programs and Executing Them from the MATLAB Editor

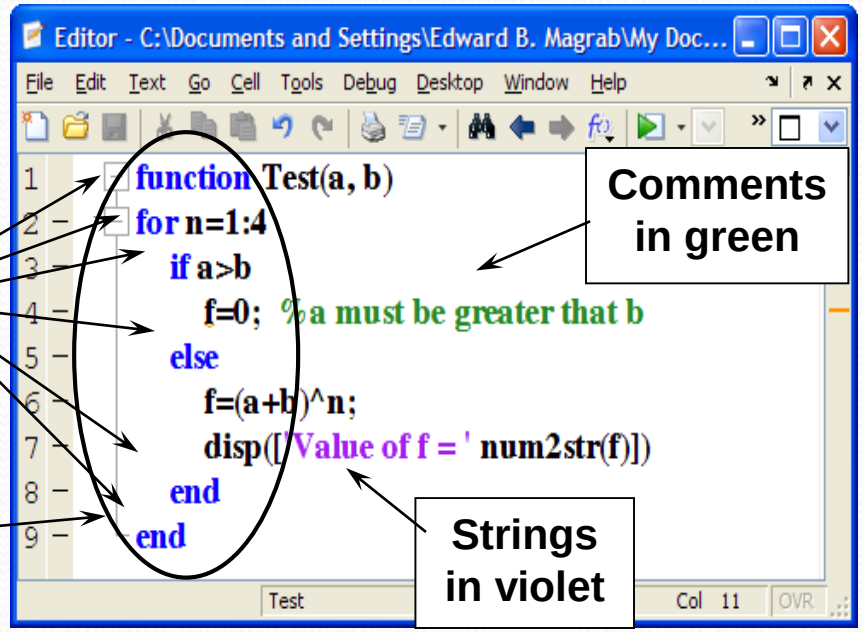
When to use the editor;

1. **The program will contain more than a few lines of code.**
2. **The program will be used again.**
3. **A permanent record is desired.**
4. **It is expected that occasional upgrading will be required.**
5. **Substantial debugging is required.**
6. **One wants to transfer the listing to another person or organization.**

Additional Reasons -

- Required when creating functions
- Editor has many features
  - Commenting/Un-commenting lines
  - Smart Indenting
  - Parentheses checking





Keywords in blue

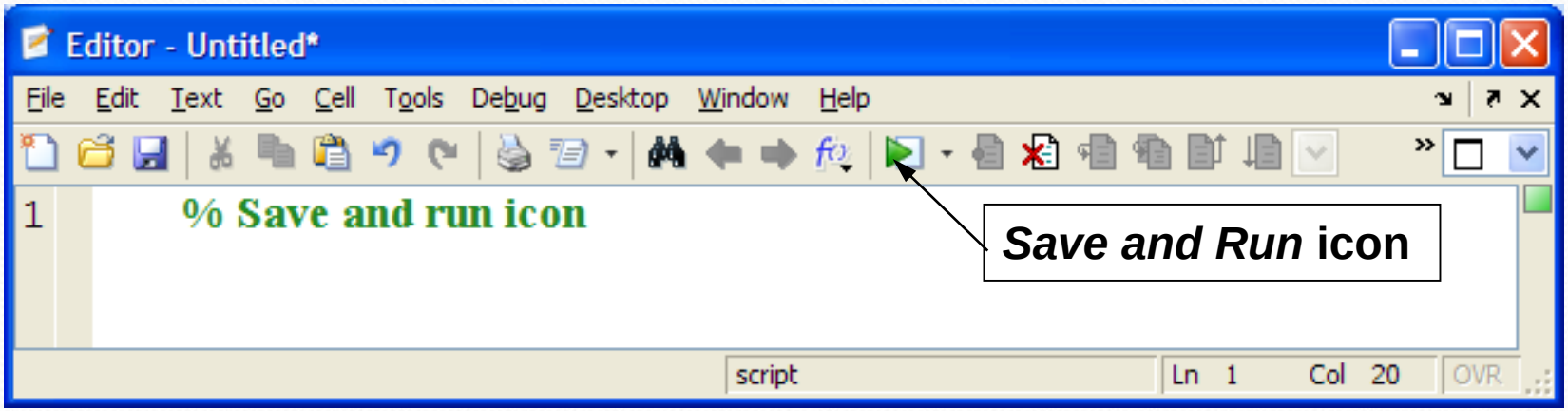
Smart indent

Comments in green

Strings in violet

Additional Reasons -

- **Click one icon to save and run a program (Must use *Save As* first time)**



Additional Reasons -

Enabling *M-Lint* from the *Preferences* menu

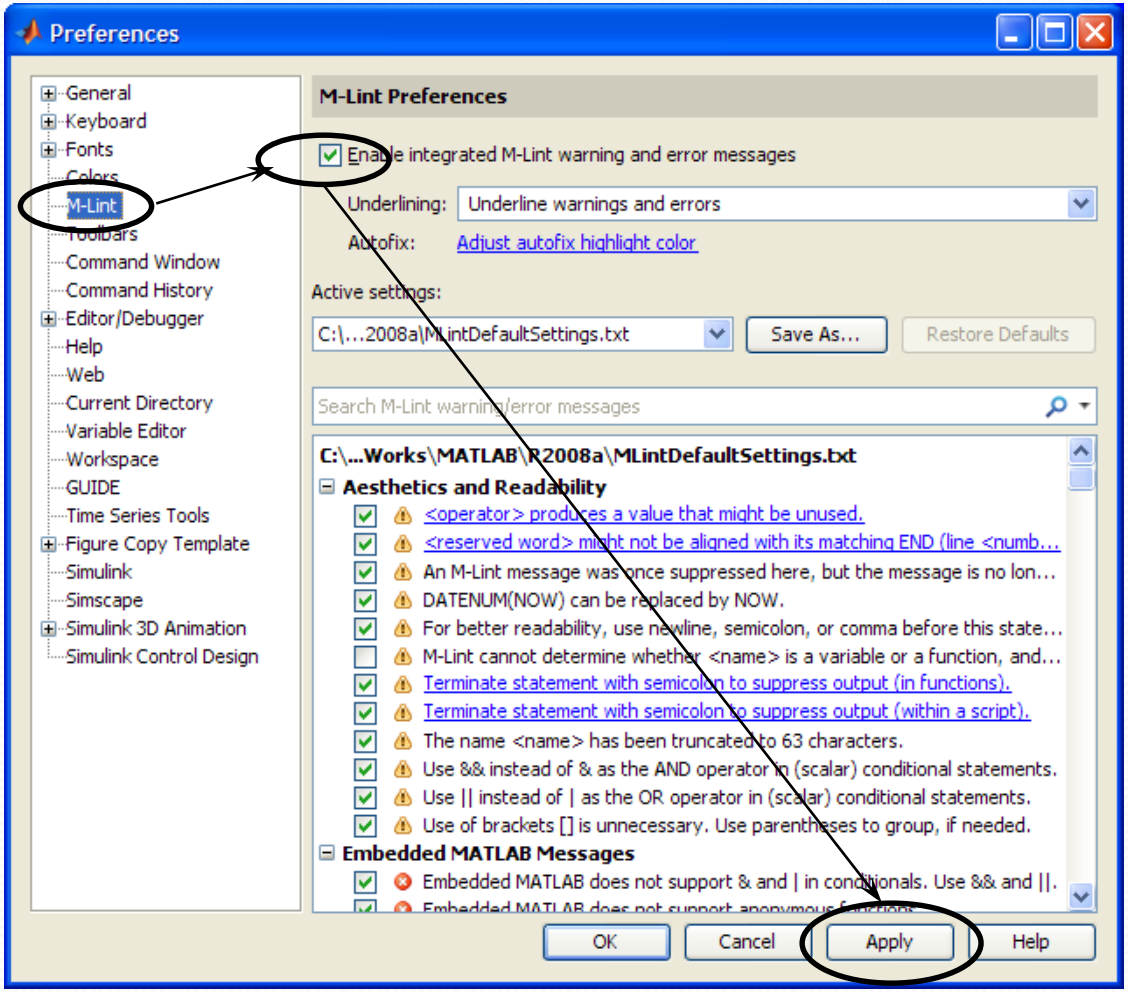
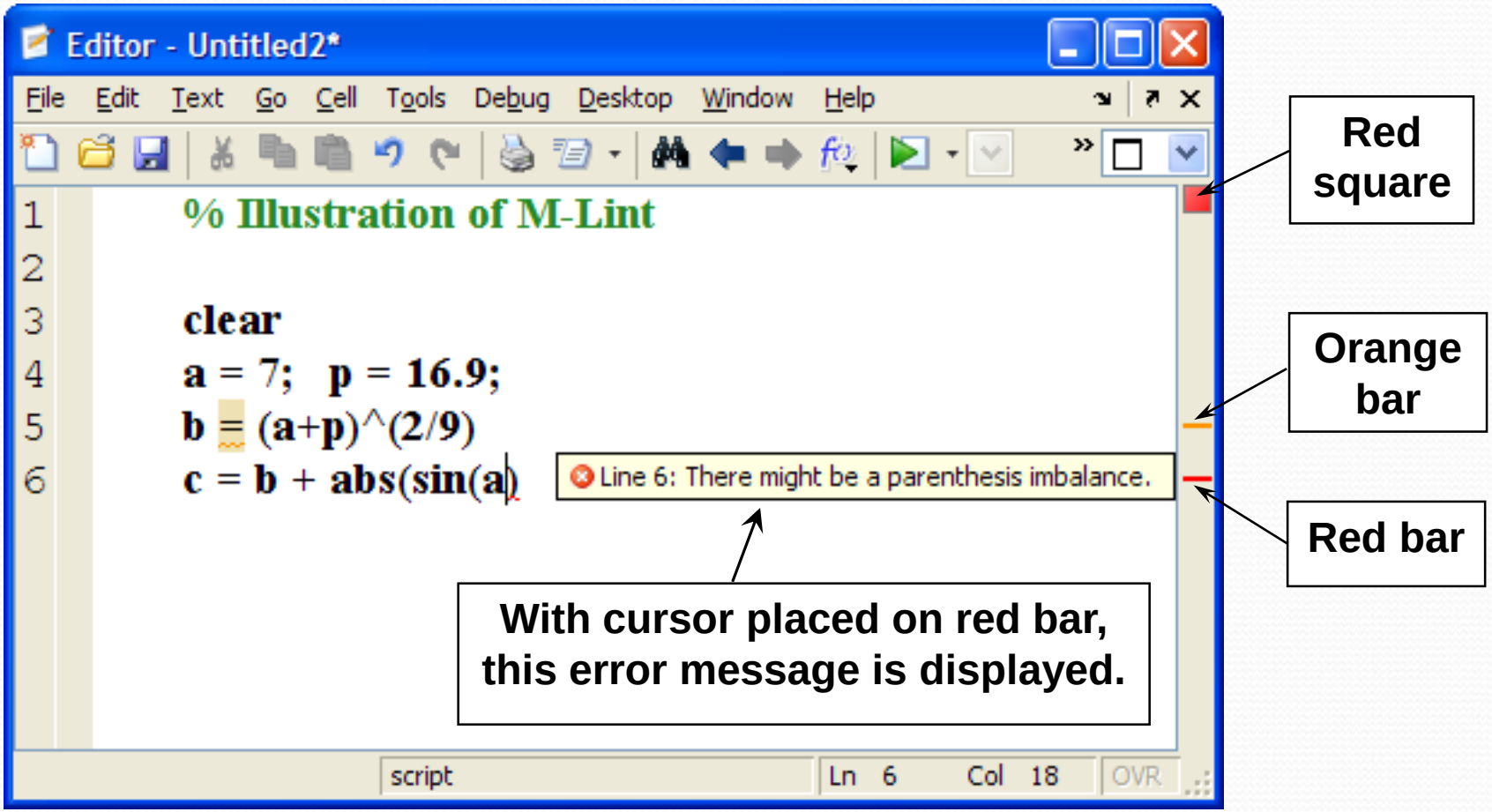


Illustration of *M-Lint*



Additional Reasons –

Enabling the cell feature of the Editor

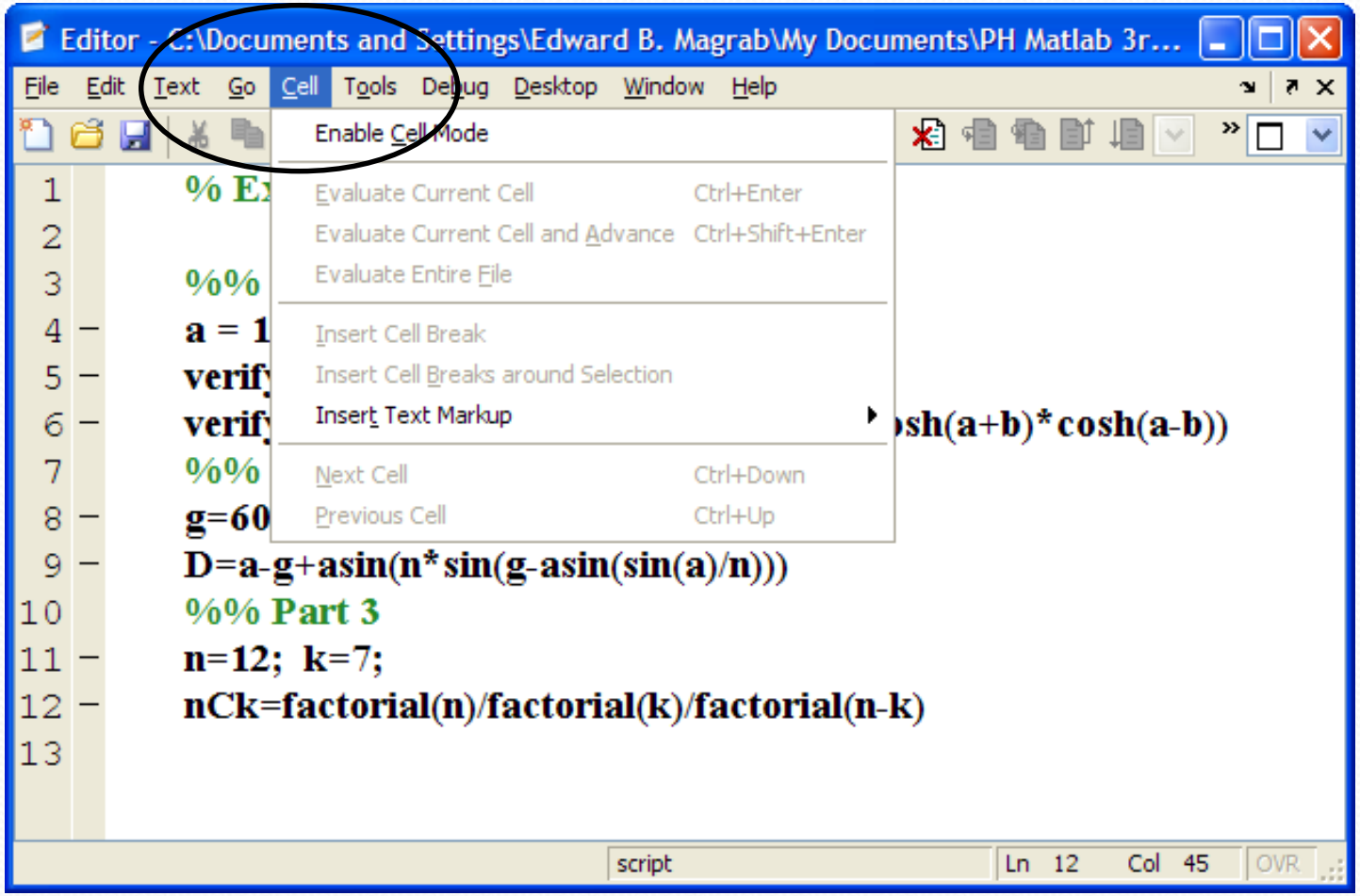
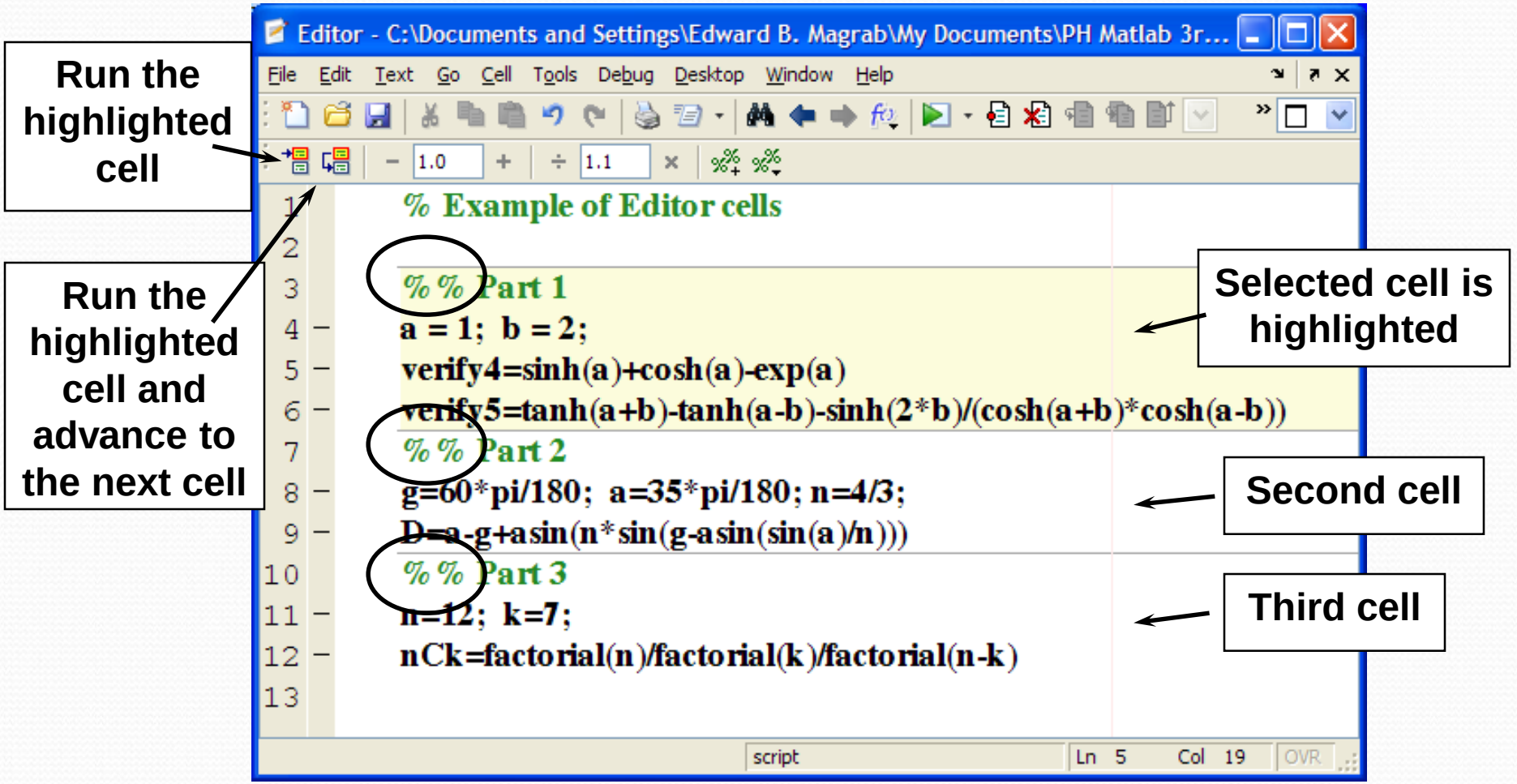


Illustration of Editing Cells



A Script or a Function Typically Has the Following Attributes -

1. ***Documentation***:

   Purpose and operations performed

   Programmer's Name

   Date originated

   Date(s) revised

   Description of the input(s): number, meaning, and type

   Description of the output(s): number, meaning, and type

2. ***Input*** -

which includes numerous checks to ensure that all input values have the qualities required for the script/function to work properly.

3. ***Initialization*** -

where the appropriate variables are assigned their numerical values.

4. ***Computation*** -

where the numerical evaluations are performed.

5. ***Output*** -

where the results are presented as graphical and/or annotated numerical quantities.

Naming of Programs and Functions

File name follows that for variable names.

Must start with an upper or lower case letter followed by up to 62 contiguous alphanumeric characters and the underscore character.

No blank spaces are allowed in file names.

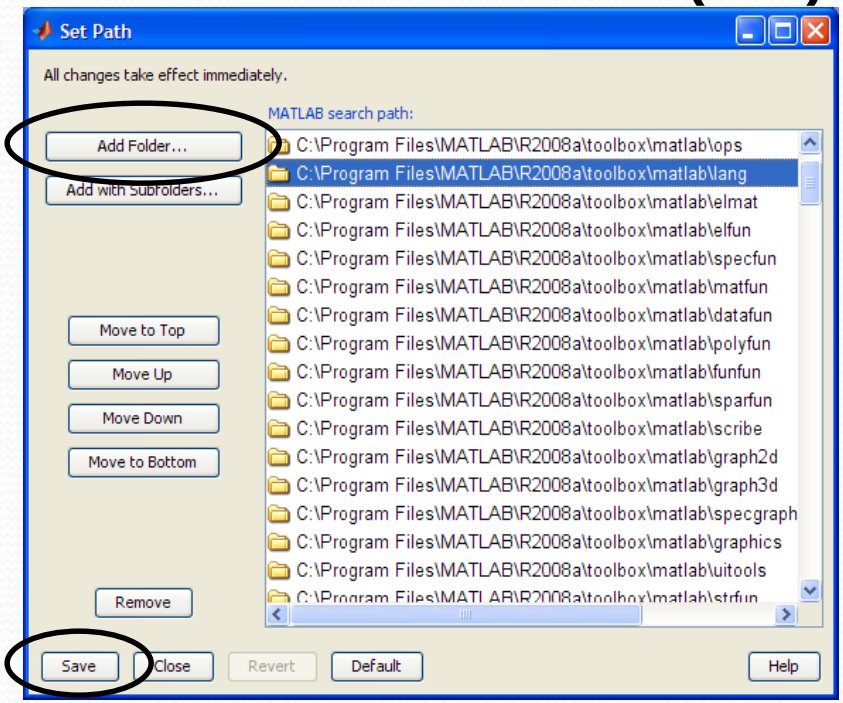
[This is different from what is allowed by the Windows operating system]

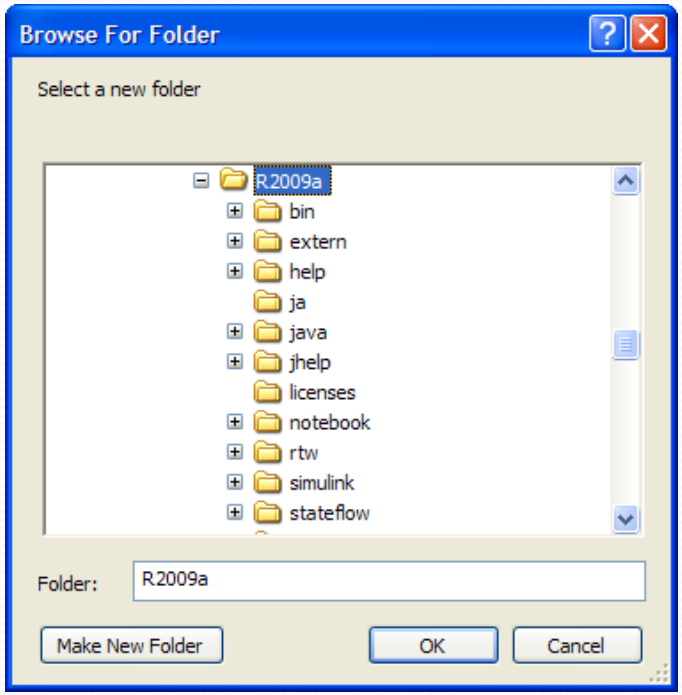
A '.m' suffix must be affixed to the file name –

Consequently, these files are called 'M' files

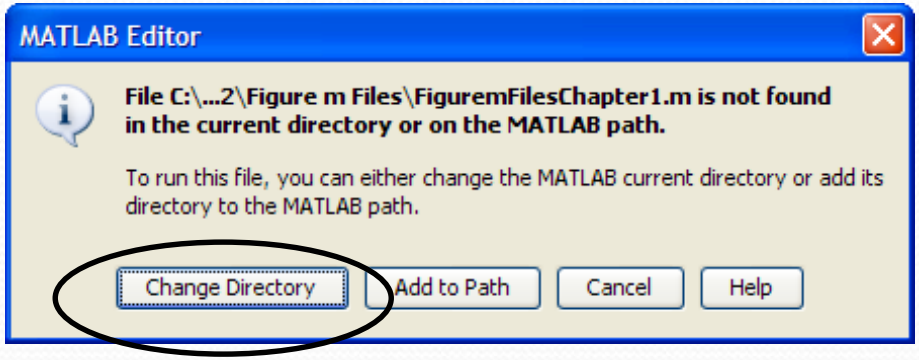
Saving/Executing (Running) M Files

Set Path window (*File*)

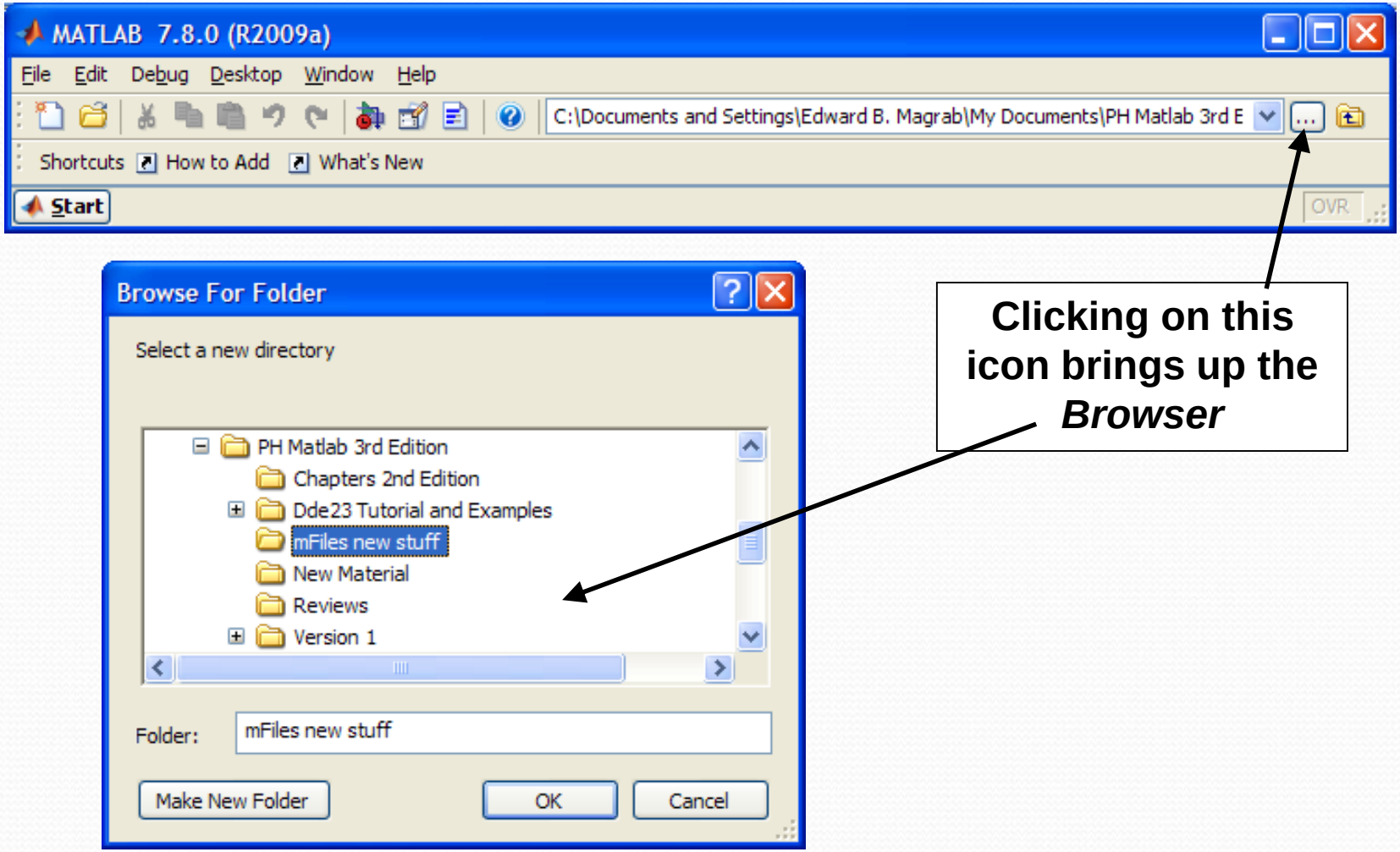




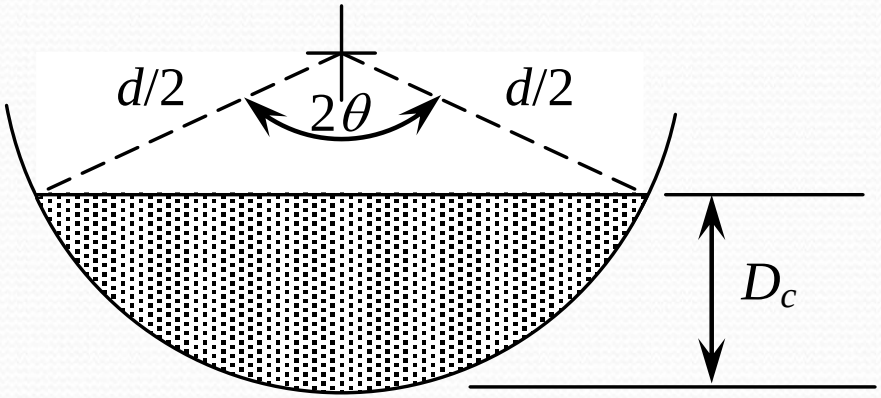
Change current path to file being executed.



Accessing the *Browser* Window to Change Current Path (Directory)



Example - Flow in a circular channel



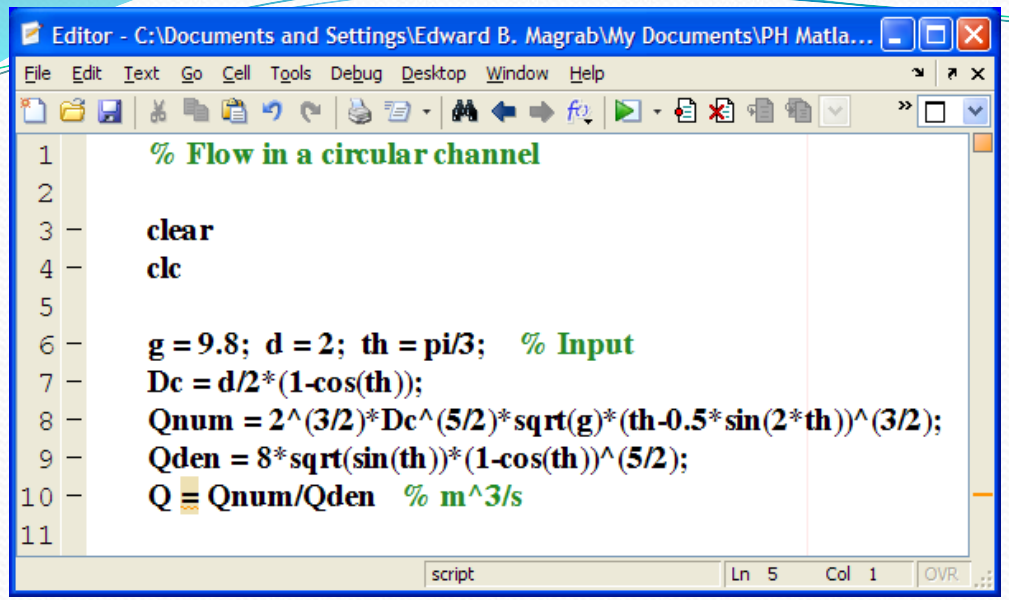
$$Q = \frac{2^{3/2} D_c^{5/2} \sqrt{g} \left(\theta - 0.5\sin(2\theta)\right)^{3/2}}{8\sqrt{\sin\theta}\left(1-\cos\theta\right)^{5/2}}$$

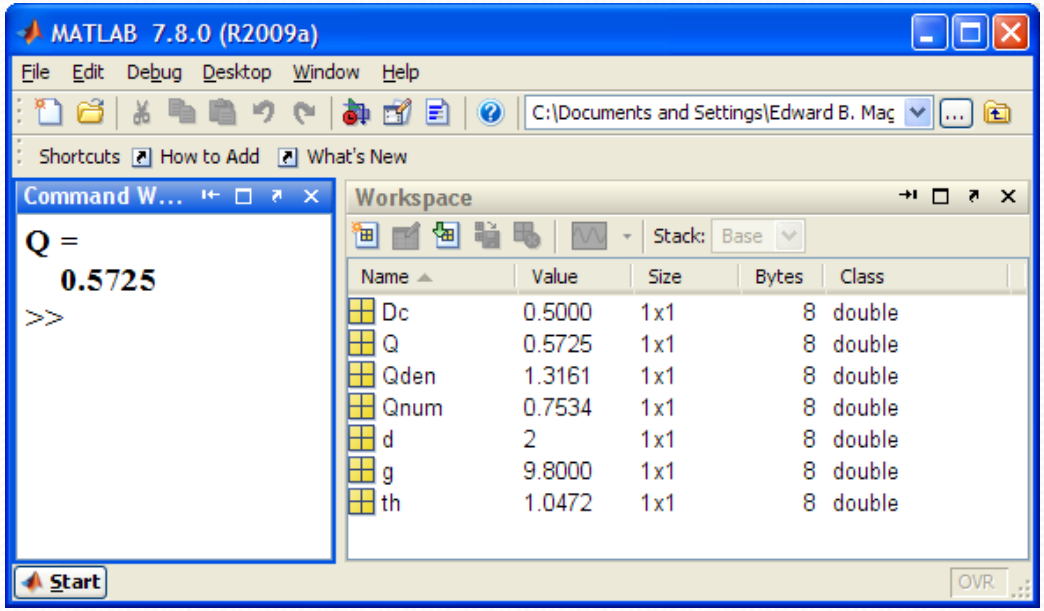
$$D_c = \frac{d}{2}(1-\cos\theta)$$

Let d = 2 m, g = 9.8 m/s², and θ = 60° = π/3. Then the script is

```
g = 9.8;  d = 2;  th = pi/3;    % Input
Dc = d/2*(1-cos(th));
Qnum = 2^(3/2)*Dc^(5/2)*sqrt(g)*(th-0.5*sin(2*th))^(3/2);
Qden = 8*sqrt(sin(th))*(1-cos(th))^(5/2);
Q = Qnum/Qden   % m^3/s
```

The various MATLAB windows after executing the program





EE214 – Computer Aided Engineering Applications

Lecture 4

Working With M–files and Program Flow Control

Objective

Introduce various means of controlling the order in which a program's expressions get evaluated and a set of relational and logical operators that are used to accomplish this control.

Topics

- Introduction – Logical Operators
- Control of Program Flow
  - Branching – `if` Statement
  - Branching – `switch` Statement
  - Specified repetition – `for` Loop
  - Unspecified repetition – `while` Loop
  - Early Termination of a `for` or `while` Loop

Program Control

- Achieved by four program flow control structures –

  `while`, `if`, `for`, and `switch`

- Each time one of these statements appears, it must be followed at a later place within the program by an `end` statement.
- All expressions that appear between the control structure statement and the `end` statement are executed until all requirements of the structure are satisfied.
- Each of these control structure statements can appear as often as necessary within themselves or within other control structures.

  When this occurs, they are called **nested structures**.

- **Control structures frequently rely on relational and logical operators to determine whether a condition has been met.**
- **When a condition has been met, the structure directs the program to a specific part of the program to execute one or more expressions.**
- **One can use the relational and logical operators to create a logical function whose output is 1 if the relational and logical operations are true and 0 if they are false.**

Relational and Logical Operators

Conditional	Mathematical symbol	MATLAB symbol
Relational operators		
equal	=	==
not equal	≠	~=
less than	<	<
greater than	>	>
less than or equal	≤	<=
greater than or equal	≥	>=
Logical operators		
and	AND	& or &&
or	OR	| or ||
not	NOT	~

Logical Operator –

Suppose that we want to create a function *g*(*x*) such that

$$g(x) = f(x) \qquad a \le x < b$$
$$= 0 \qquad x < a \text{ and } b \ge x$$

The logical operator is formed by

```
y = (a<=x & x<b)
```

where *a* and *b* have been assigned numerical values prior to this statement and

```
(a<=x & x<b)
```

is the logical operator that has a value of 1 (true) when $x \ge a$ and $x < b$. Its value is 0 (false) for all other values of *x*.

This logical operator can be used to create the unit step function $u(t)$, which is defined as

$$u(t) = 1 \quad t \geq 0$$
$$= 0 \quad t < 0$$

If t varies by increments of 0.25 in the range $-1 \leq t \leq 1$, then the following script creates the unit step function

```
t = -1:0.25:1;
UnitStep = (t>=0);
disp('    t      UnitStep')
disp([t' UnitStep'])
```

t	UnitStep
-1.0000	0
-0.7500	0
-0.5000	0
-0.2500	0
0	1.0000
0.2500	1.0000
0.5000	1.0000
0.7500	1.0000
1.0000	1.0000

Example –

Compare two vectors of equal length. Then

```
a = [4, 5, 6, 7, 8];
b = [4, 3, 2, 1, 8];
d = (a == b)
e = (a > b)
```

Its execution gives

```
d =
     1     0     0     0     1
e =
     0     1     1     1     0
```

Control of Program Flow

`if` Statement

The `if` statement is a conditional statement that branches to different parts of its structure depending on the satisfaction of certain conditional expressions.

The general form of the `if` statement is

```
if condition #1
   expressions #1
elseif condition #2      % (optional)
   expressions #2
else                     % (optional)
   expressions #3
end
```

Example –

```
if j == 1
   z = sin(x) ;
 if nnum <= 4
      nr = 1 ;
      nc = 1;
   else
      nr = 1 ;
      nc = 2;
 end
else
   nr = 2;
   nc = 1;
end
```

Executed only when $j = 1$.

This `if` statement encountered only when $j = 1$.

These statements executed only when $j = 1$ and $nnum \leq 4$.

These statements executed only when $j = 1$ and $nnum > 4$.

These statements executed only when $j \neq 1$.

Note: When comparing a vector to a scalar, the condition is satisfied only when each element in the vector satisfies the condition.

Switch Statement -

The switch structure is, essentially, an alternative to using a series of if-elseif-else-end structures.

The general form of the switch statement is

```
switch switch_expression
    case case_expression #1
        statements #1
    case case_expression #2
        statements #2
    case case_expression #n
        statements #n
    otherwise
       statements #n+1
end
```

Example –

```
k = 3;
switch k
   case 1
      disp('Case 1')          ← Executed only when k = 1
   case {2, 3}                ← Notice that cell is used
      disp('Case 2 or 3' )    ← Executed only when k = 2, 3
   case 9
      disp('Case 9')          ← Executed only when k = 9
   otherwise
      disp('Otherwise')       ← Executed only when k ≠ 1, 2, 3, or 9
end
```

`for` Loop

A `for` loop repeats a series of statements a specific number of times. Its general form is

```
for variable = expression
   statements
end
```

where one or more of the *statements* can be a function of *variable*.

`for` Loop Example

```
for  i = 1:5
     disp(i)
end
```

Same example can be written as

```
for i = 1:5, disp(i), end
```

Array Pre-allocation –

Consider a single `for` loop of the form

```
A = zeros(Nrow, 1);   % Array pre-allocation
for r = 1:Nrow
   Statements
   A(r) = ...
end
```

where *Nrow* is a positive integer that previously has been assigned a numerical value.

The addition of the array assignment statement

```
A = zeros(Nrow, 1);
```

ensures that the loop executes at maximum speed.

For nested `for` loops, we have that

```
B = zeros(Nrow, Ncol);   % Array pre-allocation
for c = 1:Ncol           % Column must be outer loop index
   Statements
   for r = 1:Nrow        % Row must be inner loop index
      Statements
      B(r, c) = ...      % Index order must be as shown
   end
end
```

where *Nrow* and *Ncol* are positive integers that previously have been assigned numerical values.

Example – Creation of a Sequentially-numbered Square Matrix

We shall generate an ($N \times N$) matrix in which the elements of each row are such that

$a_{11} = 1$ and $a_{1n} = N$
$a_{21} = N+1$ and $a_{2n} = 2N$
…
$a_{N1} = (N-1)N + 1$ and $a_{NN} = N^2$

The script is

```
N = input('Enter a positive integer < 15: ');
Matr = zeros(N, N);
for r = 1:N
   Matr(r,1:N) = ((r-1)*N+1):r*N;
end
disp(Matr)
```

Upon execution, we obtain

```
Enter a positive integer < 15: 9
    1    2    3    4    5    6    7    8    9
   10   11   12   13   14   15   16   17   18
   19   20   21   22   23   24   25   26   27
   28   29   30   31   32   33   34   35   36
   37   38   39   40   41   42   43   44   45
   46   47   48   49   50   51   52   53   54
   55   56   57   58   59   60   61   62   63
   64   65   66   67   68   69   70   71   72
   73   74   75   76   77   78   79   80   81
```

Example – Dot Multiplication of Matrices

We shall perform the dot multiplication of two matrices *A* and *B* of the same order.

The script is equivalent to A.*B.

In our case, we will illustrate the procedure using *A* = `magic`(3) and *B* = *A*'.

However, in general, before the multiplication can be performed, one must ensure that the order of the matrices is equal.

The script is

```
A = magic(3);
B = A';
[rA, cA] = size(A);
[rB, cB] = size(B);
if (rA~=rB)||(cA~=cB)
   error('Matrices must be the same size')
end
M = zeros(rA, cA);
for c = 1:cA
   for r = 1:rA
      M(r, c) = A(r, c)*B(r, c);
   end
end
disp(M)
```

Upon execution, we obtain

```
64   3   24
 3  25   63
24  63    4
```

Example - Specification of the Elements of an Array

We shall create an (*n*×*n*) matrix whose elements are either +1 or –1 such that

$$M = \begin{bmatrix} 1 & -1 & 1 & \cdots \\ -1 & 1 & -1 & \cdots \\ 1 & -1 & 1 & \cdots \\ \vdots & & & \end{bmatrix} \rightarrow (n \times n)$$

The script is

```
n = input('Enter the order of the square matrix: ');
k = 1:n;
M = zeros(n, n);
OddRow = (-1).^(k-1);
EvenRow = (-1).^k;
for m = 1:2:n
    M(m,:) = OddRow;
    if m+1 <= n
        M(m+1,:) = EvenRow;
    end
end
disp(M)
```

$$M = \begin{bmatrix} 1 & -1 & 1 & \cdots \\ -1 & 1 & -1 & \cdots \\ 1 & -1 & 1 & \cdots \\ \vdots & & & \end{bmatrix}$$

The execution of this script for *n* = 3 displays to the command window

Enter the order of the square matrix: 3

```
     1    -1     1
    -1     1    -1
     1    -1     1
```

where the value 3 was entered by the user.

`while` Loop

The `while` loop repeats one or more statements an indefinite number of times, leaving the loop only when a specified condition has been satisfied.

Its general form is

```
while condition
   statements
end
```

where the expression defining *condition* is usually composed of one or more of the variables evaluated by *statements*.

`while` Loop Example

```
i=10
while i>0
    disp (i)
    i=i-1;
end
```

Early Termination of Either a `for` or `while` Loop

The `break` function is used to terminate either a `for` or `while` loop.

If the break function is within nested `for` or `while` loops, then it returns to the next higher level `for` or `while` loop.

Example –

```
for j = 1:14
   ...
   b = 1
   while b < 25
      ...
      if n < 0
        break
      end
      ...
   end
   ...
end
```

When *n* < 0 the `while` loop is exited and the script continues from the next statement ***after*** this `end` statement

Example –

```
n = 0;
while abs(x) > 1
        x = x/2;
        n = n+1;
        if n > 50, break, end
end
```

Input/Output Data Files – `load/save`

`load`

- **Reads data on a row-by-row basis.**
  - **Each row separated by an *Enter*, where *Enter* is used instead of the semicolon.**
  - **Each data value separated by either one or more blanks or by a comma.**
- **The number of columns of data in each row must be the same.**
- **For a row vector, data entered without using *Enter*.**
- **For a column vector, each data value followed by an *Enter*.**

Example –

Assume that data reside in an ASCII text file *DataSection33.txt* in the form

11 12 13
21 22 23
31 32 33
41 42 43

The `load` function is

```
load('DataSection33.txt')
```

Let us square each element of the matrix in *DataSection33.txt*. The script is

```
load('DataSection33.txt')
y = DataSection33.^2
```

which, upon execution, results in

```
y =
         121         144         169
         441         484         529
         961        1024        1089
        1681        1764        1849
```

11	12	13
21	22	23
31	32	33
41	42	43

save

If one wants to save numerical values resulting from the execution of a script or function to a file, then we use

`save('File name', 'Variable 1', 'Variable 2', …, '-ascii')`

where

'*File name*' is a string containing the name of the file to be saved and its directory, if other that the current directory.

'*Variable n*' are strings containing the names of the *n* variables that are to be saved in the order that they appear.

'*-ascii*' is a string indicating that the data will be saved in ascii format.

Example –

Save the square of each value in a file named *DataSection33.txt* as ASCII text. The script is

```
load('DataSection33.txt')
y = DataSection33.^2;
save('SavedDataSection33.txt', 'y', '-ascii')
```

Upon execution, the script creates a text file whose contents are

```
1.2100000e+002  1.4400000e+002  1.6900000e+002
4.4100000e+002  4.8400000e+002  5.2900000e+002
9.6100000e+002  1.0240000e+003  1.0890000e+003
1.6810000e+003  1.7640000e+003  1.8490000e+003
```

A Note About Path Name –

When just the file name is given, MATLAB places the file in the current directory.

In order to place the file in a specific directory, the entire path name must be given.

For example,

```
load('DataSection33.txt')   % File in current directory
y = DataSection33.^2;
z = sqrt(Datasection33);
save('c:\Matlab mfiles\Matlab
        results\SavedDataSection33.txt', 'y', 'z', '-ascii')
```

Execution of this script creates the file *SavedDataSection331.tx*t with the contents

y

1.2100000e+002 1.4400000e+002 1.6900000e+002
4.4100000e+002 4.8400000e+002 5.2900000e+002
9.6100000e+002 1.0240000e+003 1.0890000e+003
1.6810000e+003 1.7640000e+003 1.8490000e+003

z

3.3166248e+000 3.4641016e+000 3.6055513e+000
4.5825757e+000 4.6904158e+000 4.7958315e+000
5.5677644e+000 5.6568542e+000 5.7445626e+000
6.4031242e+000 6.4807407e+000 6.5574385e+000