

VARIABLES, VECTORS and MATRICES in MATLAB*

* Some of the contents are adopted from
Essential MATLAB for Engineers and Scientists, B. H. Hahn, D. T. Valentine, 5th Ed., Academic Press, MA, 2013.

Variables

- A variable *name* must comply with the following two rules,
 - It may consist only of the letters *a–z*, the digits *0–9*, and the underscore (*_*),
 - It **must** start with a letter.

Examples:

1) `a_variable`, `myvar2` are valid.

2) `a-variable`, `2myvar`, `var$`, `_var` are **invalid**.

- A variable is created simply by assigning a value to it at the command line or in a program.

For example,

`a = 98,`

assigns 98 to the variable `a`.

Variables ctd.

- MATLAB is *case-sensitive*, which means it distinguishes between upper and lowercase letters.
- All the variables you create during a session remain in the workspace until you clear them.
- Values of the variables in the workspace can be used or changed at any stage during the session.
- Remember that, the command `who` lists the names of all the variables in your workspace. And the command `whos` lists the size of each variable as well.

Variables ctd.

- In MATLAB, all variables are referred to as *arrays*, *whether* they are single-valued (scalars) or multi-valued (vectors or matrices). In other words, a scalar is a 1-by-1 array.
- Each scalar occupies eight *bytes of storage* where a byte is the amount of computer memory required for one character (one byte is the same as eight *bits*).

Examples:

```
value = 10;  
var1 = value;  
a = 5i-5;  
b = a/5;  
x = 3, y = 5;
```

Vectors

- A *vector* is a special type of matrix, having only one row or one column. Vectors are called *lists* or *arrays* in some programming languages.
- MATLAB handles vectors and matrices in the same way.
- Elements in the list must be enclosed in square brackets, not parentheses. And Elements in the list must be separated *either by spaces or by commas*.
- To begin, we try the accompanying short exercises on the command line.

Exercises on Vectors

Enter a statement like:

```
v1 = [2 4 -5 0 10]
```

or

```
v1 = [2,4,-5,0,10]
```

- Can you see that you have created a vector (list) with five elements?
- Enter the command `disp(v1)` to see how MATLAB displays a vector.
- Enter the command `whos` (or look in the Workspace browser).

The Colon Operator

- A vector can also be generated with the *colon(:) operator*

For example, enter the following statements:

- 1) `x = 1:10` (elements are the integers 1, 2, ..., 10),
- 2) `x = 1:0.5:4` (elements are the values 1, 1.5, ..., 4 in increments of 0.5.),
- 3) `x = 10:-1:1` (elements are the integers 10, 9, ..., 1),
- 4) `x = 1:2:6` (elements are 1, 3, 5).

The Colon Operator ctd.

The General Structure:

$j:k$ is the same as $[j, j+1, \dots, k]$, or empty when $j > k$.

$j:i:k$ is the same as $[j, j+i, j+2i, \dots, j+m*i]$, where m is the rounded value of $((k-j)/i)$ towards zero, for integer values.

linspace and logspace Functions

- The linspace function generates linearly spaced vectors. It is similar to the colon operator ":", but gives direct control over the number of points. For example,

```
linspace(0, pi/2, 10)
```

creates a vector of 10 equally spaced points from 0 to $\pi/2$.

- The logspace function generates logarithmically spaced vectors. Especially useful for creating frequency vectors, it is a logarithmic equivalent of linspace and the ":" or colon operator. For example,

```
y = logspace(a,b,n)
```

generates n points between decades 10^a and 10^b .

Transposing Vectors

- All of the vectors we examined so far are *row vectors*. Each has one row and several columns.

To generate the *column vectors*,

- 1) ‘;’ can be used, i.e. $x1 = [1;2;3]$ generates a 3 row 1 column vector,
- 2) A row vector can be transposed, i.e. $x2 = [1 \ 2 \ 3]'$ also generates a 3 row 1 column vector.

$$x1 = x2 = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Subscripts

- We can refer to particular elements of a vector by means of *subscripts*.

For example,

- Let $v = [0,5,10,15,20,25,30]$;
- $v(3)$ gives the third element of v which is 10, here the numeral 3 is the subscript.
- $v(2:4)$ gives the second, third and fourth elements of v .
- $v(1:2:6)$ gives the first, third and fifth elements of v .
- $v([1,7,3])$ gives the first, seventh and third elements of v .

Matrices

- A *matrix* may be thought of as a table consisting of rows and columns. We can create a matrix just as we do a vector, except that a semicolon is used to indicate the end of a row.

For example, $a = [1\ 2\ 3; 4\ 5\ 6]$ results in

```
a =  
    1    2    3  
    4    5    6
```

- A matrix can also be transposed, i.e. a' results in

```
ans =  
    1    4  
    2    5  
    3    6
```

Matrices ctd.

- A matrix can be constructed from column vectors of the same length. For example,

```
x = 0:30:180;  
table = [x' sin(x*pi/180)']
```

result in,

```
table =  
      0      0  
30.0000  0.5000  
60.0000  0.8660  
90.0000  1.0000  
120.0000  0.8660  
150.0000  0.5000  
180.0000  0.0000
```

Matrices ctd.

- Similar to the vectors, particular elements of matrices can also be referred by means of subscripts. For example, let

$m =$

1	2	3
4	5	6

then, $m(2,3)$ results in 6.

Some Built-in Functions

Function	Description
<code>zeros(n)</code>	returns $n \times n$ matrix of zeros where n is an integer
<code>zeros(n,m)</code>	returns $n \times m$ matrix of zeros where n and m are integers
<code>ones(n)</code>	returns $n \times n$ matrix of ones where n is an integer
<code>ones(n,m)</code>	returns $n \times m$ matrix of ones where n and m are integers
<code>size(y)</code>	gives the sizes of each dimension of array y in a vector
<code>eye(n)</code>	returns $n \times n$ identity matrix
<code>eye(n,m)</code>	returns an $n \times m$ matrix with 1's on the diagonal and 0's elsewhere.
<code>length(y)</code>	returns the length of the vector y or largest array dimension of y

Examples

```
>>x = [1,2;3,4];
```

```
>>y = ones(size(x))
```

```
y =
```

```
    1    1
```

```
    1    1
```

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

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

```
>>c = ones(size(a))-eye(size(b))
```

```
c =
```

```
    0    1    1
```

```
    1    0    1
```

```
>>d = ones(size(a))-length(b)
```

```
d =
```

```
   -2   -2   -2
```

```
   -2   -2   -2
```


Arithmetic Operations for Arrays

Operation	Operator	Description
Addition	$a+b$	Element by element matrix addition
Substraction	$a-b$	Element by element matrix subtraction
Multiplication	$a.*b$	Element by element matrix multiplication (a or b can be a scalar)
Matrix multiplication	$a*b$	Matris multiplication
Right division	$a./b$	Element by element right division (a or b can be a scalar)
Left division	$a.\backslash b$	Element by element left division (a or b can be a scalar)
Right matrix division	a/b	The inverse of the matrix b is multiplied by the matrix a from left
Left matrix division	$a\backslash b$	The matrix b is multiplied by the inverse of the matrix a from left
Power	$a.^b$	Element by element power (a or b can be a scalar)

Examples

```
>> x = [2 4;6 8];y = [2 8;5 9];
```

```
>> z = x-y
```

```
z =
```

```
0   -4
```

```
1   -1
```

```
>> x = rand(2,2);y = rand(3,2)
```

```
y =
```

```
0.6324  0.5469
```

```
0.0975  0.9575
```

```
0.2785  0.9649
```

```
>> z = x+y
```

??? Error using ==> plus

Matrix dimensions must agree.

```
>> x = [2 4;6 8];y = [2 8;5 9];
```

```
>> b = x+1
```

```
b =
```

```
3   5
```

```
7   9
```

```
>> x*y
```

```
ans =
```

```
24   52
```

```
52  120
```

```
>> x.*y
```

```
ans =
```

```
4   32
```

```
30   72
```

Submatrices, Examples

```
>> x = [1 2;3 4];
```

```
>> x(:,1)
```

```
ans =
```

```
1
```

```
3
```

```
>> x(2,:)
```

```
ans =
```

```
3 4
```

```
>> x(:)
```

```
ans =
```

```
1
```

```
3
```

```
2
```

```
4
```

```
>> y = [1 2 3; 4 5 6; 7 8 9; 10 11 12];
```

```
>> y(2,:)
```

```
ans =
```

```
4 5 6
```

```
>> y(2:4,1:2)
```

```
ans =
```

```
4 5
```

```
7 8
```

```
10 11
```

Some Constants and Permanent Variables in MATLAB

Function	Description
ans	Most recent answer
pi	The π number in double precision
i,j	Imaginary unit
eps	Floating-point relative accuracy = 2^{-52}
Inf	Infinity (i.e. 1/0)
NaN	Not-a-Number (i.e. 0/0)
clock	Current time as date vector (<i>year, month, day, h, min, sec</i>)
date	Current date <i>string</i>

Some Constants and Permanent Variables in MATLAB

Function	Description
tic, toc	<p>Measure performance using stopwatch timer</p> <div> <p>Measure the time required to create two random matrices.</p> <pre>tic A = rand(12000,4400); B = rand(12000,4400); toc</pre> <p><i>Elapsed time is 0.605918 seconds.</i></p> </div>
realmax, realmin	<p>Largest and smallest normalized floating-point number</p> <p><i>Realmax</i> : $(2-2^{(-52)}) \cdot 2^{1023}$</p> <p><i>Realmin</i>: $2^{(-1022)}$</p>
flintmax	<p>Largest consecutive integer in floating-point format</p> <p>2^{53}</p>

BUILT-IN MATHEMATICAL FUNCTIONS in MATLAB

Elementary Math Functions

The matrices A and B are defined as **A = [1,2,3]**, **B = [1,2,3;4,5,6;7,8,9]** for the following examples. If the input is a matrix, *func()* treats the columns of the matrix as vectors, returning a row vector of the sums of each column.

Function	Description	Example
sum()	Sum of array elements	sum(A) = 6 sum(B) = [12,15,18]
prod()	Product of array elements	prod(A) = 6 prod(B) = [28,80,162]
max()	Largest elements in array	max(A) = 3 max(B) = [7,8,9]
min()	Smallest elements in array	min(A) = 1 min(B) = [1,2,3]
mean()	Average or mean value of array	mean(A) = 2 mean(B) = [4,5,6]

Elementary Math Functions ctd.

Function	Description	Example
rand(m,n)	returns an m-by-n matrix, containing pseudorandom values drawn from the standard uniform distribution on the open interval (0,1)	
round()	Round to nearest integer	round([1.5,-1.4]) = [2,-1]
floor()	Round toward negative infinity	floor([1.5,-1.4]) = [1,-2]
ceil()	Round toward positive infinity	ceil([1.5,-1.4]) = [2,-1]
fix()	Round toward zero	fix([1.5,-1.4]) = [1,-1]
mod(x,y)	Modulus after division	mod(3,2) = 1, mod(3,-2) = -1
rem(x,y)	Remainder after division	rem(3,2) = 1, rem(3,-2) = 1

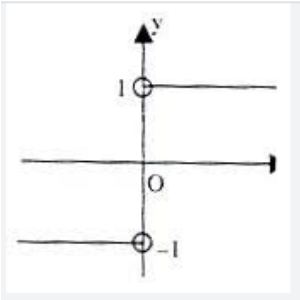
Elementary Math Functions ctd.

Function	Description	Example
factorial()	Factorial function	factorial(5) = 120
perms()	All possible permutations	perms([3,5]) = [5,3;3,5] perms([2,6,8]) = [8,6,2;8,2,6;6,8,2;6,2,8;2,8,6;2,6,8]
factor()	Prime factors	factor(15) = [3,5] factor(120) = [2,2,2,3,5]
primes()	Generate list of prime numbers	primes(7) = [2,3,5,7]
isprime()	Array elements that are prime numbers	isprime([6,7]) = [0,1]
lcm()	Least common multiple	lcm(30,20) = 60 lcm(30,15) = 30
gcd()	Greatest common divisor	gcd(30,20) = 10

Complex Number Functions

Function	Description	Example
$a+bi$ $a+bj$	Complex number input in the command line	$x = 3+5i$ $x = 3+5j$
<code>complex()</code>	Construct complex data from real and imaginary components	<code>complex(a,b) = a+bi</code>
<code>abs()</code>	Absolute value and complex magnitude	<code>abs(3+4i) = 5</code>
<code>angle()</code>	Phase angle of the complex number	<code>angle(3+4i) = 0.9273</code>
<code>conj()</code>	Complex conjugate	<code>conj(3+5i) = 3-5i</code>

Complex Number Functions ctd

Function	Description	Example
real()	Real part of complex number	real(3+5i) = 3
imag()	Imaginary part of complex number	imag(3+5i) = 5
isreal()	Check if input is real array	x = 3+5i isreal(x) = 0 isreal(x+conj(x))=1
sign()	<p>Signum function</p> $f(x) = \begin{cases} \frac{x}{ x }, & x \neq 0 \\ 0, & x = 0 \end{cases}$ $f(x) = \begin{cases} 1, & x > 0 \\ 0, & x = 0 \\ -1, & x < 0 \end{cases}$	<p>sign(3+4i)=0.6+0.8i</p> 

Trigonometric Functions

- sine, cosine, tangent, cotangent, secant and cosecant functions are defined in MATLAB as `sin()`, `cos()`, `tan()`, `cot()`, `sec()`, `csc()`, respectively for inputs in radians.
- Each trig function can be used for the inputs in degrees by adding “d” at the end of the function.

Example: `sind(90)=1`.

- “h” is added to the end of a function for hyperbolic functions.

Example : `sinh(1)=1.1752`.

- “a” is added to the beginning of a function for inverse trigonometric functions.

Example: `acos(1)=0`.

Exponential Functions

Function	Description	Example
<code>sqrt()</code>	Square root	<code>sqrt(4) = 2</code> <code>sqrt([1,4]) = [1,2]</code>
<code>exp()</code>	Exponential	<code>exp(1) = 2.718281828459046</code>
<code>expm1(x)</code>	Compute $\exp(x)-1$ accurately for small values of x because it compensates for the round-off error in $\exp(x)$	<code>expm1(1) = 1.7183</code>
<code>nextpow2()</code>	<code>p=nextpow2(A)</code> gives the value of p where $2^p \geq A $	<ul style="list-style-type: none"><code>nextpow2(8) = 3</code><code>nextpow2(9) = 4</code><code>nextpow2(-16) = 4</code>

Exponential Functions ctd.

Function	Description	Example
<code>log()</code>	Natural logarithm (<i>$\ln(x)$</i>)	$\log(\exp(1)) = 1$ $\log(\exp(2)) = 2$
<code>log10()</code>	Common (base 10) logarithm	$\log_{10}(100) = 2$
<code>log2()</code>	Base 2 logarithm	$\log_2(4) = 2$ $\log_2(\text{eps}) = -52$
<code>log1p(x)</code>	Compute $\log(1+x)$ accurately for small values of x	
<code>reallog()</code>	Natural logarithm for nonnegative real arrays	$a = \exp(2);$ $\text{reallog}([a, a^2]) = [2, 4]$

Polynomial Functions

- An n-degree polynomial with constant coefficients (a_i) is defined as,

$$a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$$

- Polynomials are defined as row vectors in MATLAB, i.e.

$$p_1(x) = x^5 + 2x^4 + 2x^3 + 3x^2 + x + 4$$

$$p_1 = [1, 2, 2, 3, 1, 4]$$

p1=[1,2,3]; p2 = [2,3,4]; are considered to be defined for the examples given in this subsection.

Function	Description	Example
roots()	Polynomial roots	roots(p1) = [-1+1.4142i,-1-1.4142i]
polyval(p,x)	Polynomial evaluation	polyval(p2,[3,5]) = [31,69]

Polynomial Functions ctd.

Function	Description	Example
conv()	<p>Convolution and polynomial multiplication</p> <div style="border: 2px solid purple; padding: 10px; margin: 10px 0;"> <pre> m = length(u) and n = length(v). $w(k) = \sum_j u(j)v(k - j + 1).$ w(1) = u(1)*v(1) w(2) = u(1)*v(2)+u(2)*v(1) w(3) = u(1)*v(3)+u(2)*v(2)+u(3)*v(1) ... w(n) = u(1)*v(n)+u(2)*v(n-1)+ ... +u(n)*v(1) ... w(2*n-1) = u(n)*v(n) </pre> </div>	<p>conv(p1,p2) = [2,7,16,17,12]</p>
deconv()	Deconvolution and polynomial division	<p>[a,b]=deconv(p1,p2) a = 0.5 b = 0 0.5 1</p>

Polynomial Functions ctd.

Function	Description	Example
<code>poly(A)</code>	If A is an n-by-n matrix , returns an n+1 element row vector whose elements are the coefficients of the characteristic polynomial ($\det(sI-A)$). If A is a vector , returns a row vector whose elements are the coefficients of the polynomial whose roots are the elements of A.	
<code>polyder()</code>	Polynomial derivative	<code>polyder(p2) = [4,3]</code>
<code>polyint(p,k)</code>	Returns a polynomial representing the integral of polynomial p, using a scalar constant of integration k.	<code>polyint(p2,0) = [0.666,1.5,4,0]</code>

Matrix Operation Functions

$v = [1,2,3]$, $A = [1,2,3;4,5,6;7,8,10]$ are considered to be defined for the examples given in this subsection.

Function	Description	Example
<code>diag(x)</code>	When x is a vector of n components, returns a square matrix with the elements of x on the diagonal. When x is a matrix , returns a column vector formed from the elements of the diagonal of x .	$\text{diag}(v) =$ $[1,0,0;0,2,0;0,0,3]$ $\text{diag}(A) = [1,5,10]$
<code>det()</code>	Matrix determinant	$\text{det}(A) = -3$

Matrix Operation Functions ctd.

Function	Description	Example
inv()	Matrix inverse	
norm()	Vector and matrix norms	norm(v,1) = 6, norm(v,2) = 3.7417
null()	Null space	null(A) = Empty matrix: 3-by-0
rank()	Rank of matrix	rank(A) = 3
rref()	Reduced row echelon form	
trace()	Sum of diagonal elements	trace(A)=16
sqrtm()	Matrix square root	$X = \text{sqrtm}(A) \rightarrow$ $X * X = A$

Matrix Operation Functions ctd.

Function	Description	Example
eig()	[V,D]=eig(X) produces matrices of eigenvalues (D) and eigenvectors (V) of matrix X	<p>[V,D]=eig(A)</p> <p>V =</p> <pre>-0.2235 -0.8658 0.2783 -0.5039 0.0857 -0.8318 -0.8343 0.4929 0.4802</pre> <p>D =</p> <pre>16.7075 0 0 0 -0.9057 0 0 0 0.1982</pre>
expm(A)	Matrix exponential	
linsolve(A,B)	Solve the linear system $A*x=B$	

Linear System Example

Let

$$3x_1 + 5x_2 + x_3 = 16$$

$$x_1 + x_2 = 3$$

$$3x_2 + 5x_3 = 21$$

The solution to that linear system can be obtained in MATLAB as given below.

```
>> A = [3,5,1;1,1,0;0,3,5];
```

```
>> B = [16;3;21];
```

```
>> linsolve(A,B)
```

```
ans =
```

```
1.0000
```

```
2.0000
```

```
3.0000
```

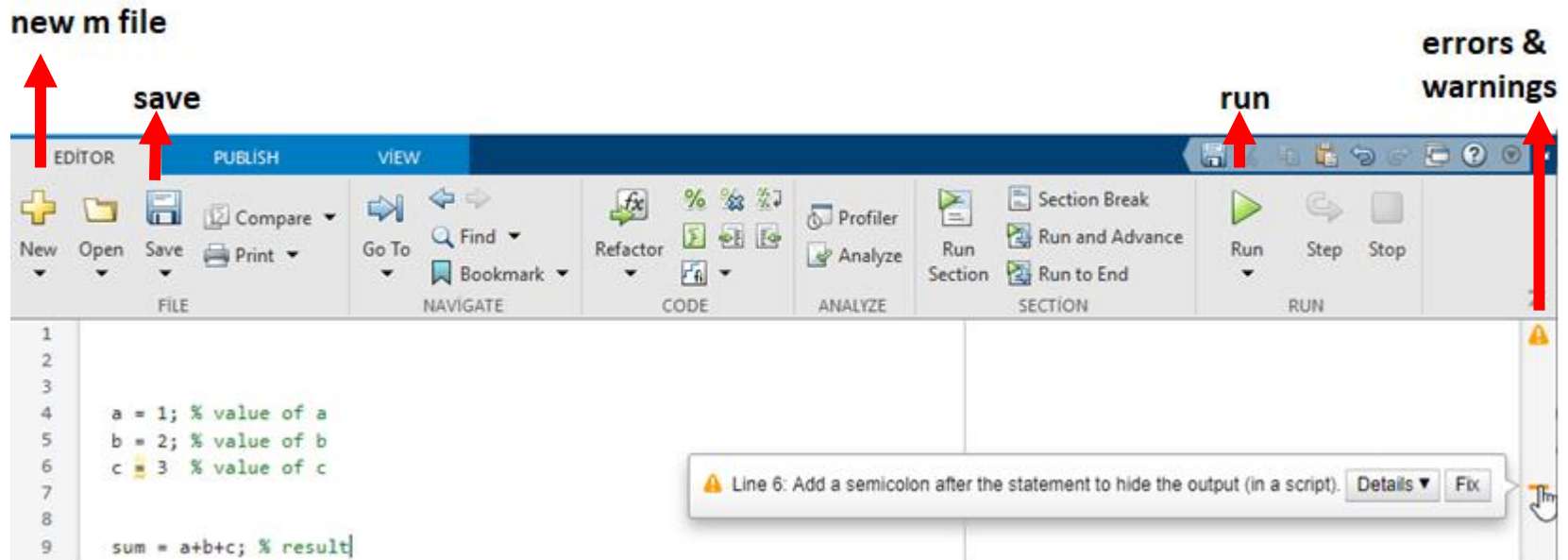
M-FILES, USER-DEFINED INPUT and OUTPUT

M-files

- Useful utilities translated into MATLAB (either sequences of command lines or *functions*) and saved as **m-files** in the working directory are our primary goals.
- In the working directory, we will begin to accumulate a set of m-files that we have created as you use MATLAB.
- The goals in designing a software tool are that it works, it can easily be read and understood, and, hence, it can be systematically **modified** when required.
- For programs to work well they must satisfy the requirements associated with the problem or class of problems they are intended to solve.
- The program must be readable and hence clearly understandable. Thus, it is useful to decompose major tasks (or the main program) into subtasks (or subprograms) that do specific parts of it. It is much easier to read subprograms, which have fewer lines, than one large main program that doesn't segregate the subtasks effectively, particularly if the problem to be solved is relatively complicated.

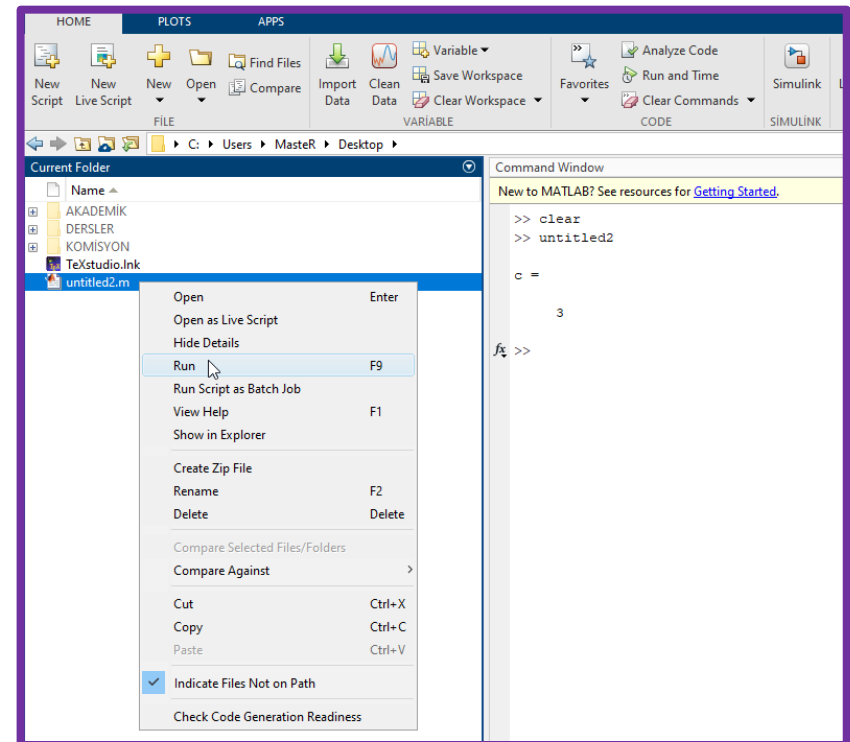
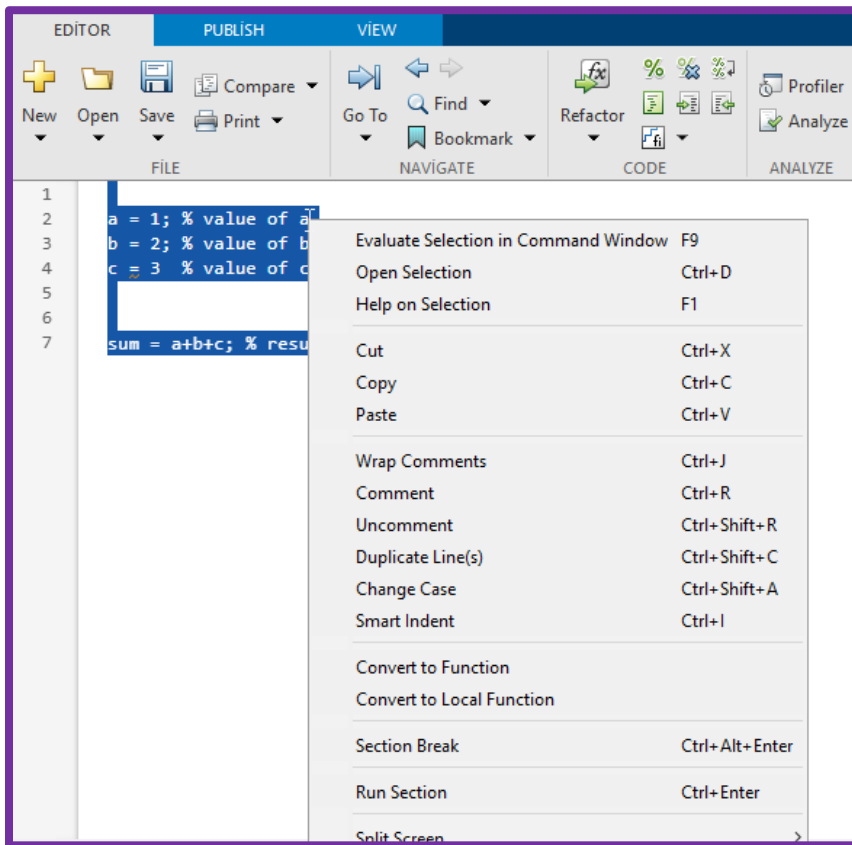
MATLAB Editor

- Example: `a = 1;b = 2;c = 3;sum = a+b+c;`



Running m-files

- In addition to save&run button in MATLAB editor, m-files can also be run as depicted in the figures below.



User Defined Input

- “input” function displays a text string in the command window and then waits for the user to provide the requested input. The input value can then be used in subsequent calculations

```
>> x = input('enter a value: ')
enter a value: 14
x =
    14
```

- The same approach can be used to enter a one- or two-dimensional matrix. The user must provide the appropriate brackets and delimiters.

```
>> x = input('enter the input vector: ')
enter the input vector: [1,2,3]
x =
     1     2     3
```

Output Options

- There are several ways to display the contents of an output.
- The general rule of displaying the result is scaled fixed-point format, with 4 digits after the decimal point.
- In MATLAB, format function controls the output format of numeric values displayed in the Command Window.

Function	Description	Example (π)
format short (default)	Scaled fixed-point format, with 4 digits after the decimal point	3.1416
format long	Scaled fixed-point format, with 15 digits after the decimal point for double	3.14159265358979
format shortE (longE)	Floating-point format, with 4 (15) digits after the decimal point for double	3.1416e+000 3.141592653589793e+000
format shortG (longG)	Fixed- or floating-point, whichever is more readable, with 4 (15) digits after the decimal point for double	3.1416 3.14159265358979
format rat	Ratio of small integers.	355/113

Output Options ctd.

- “`disp()`” displays an array, without printing the array name. If the input contains a text string, the string is displayed.
- The `fprintf` function (formatted print function) gives you even more control over the output than you have with the `disp()` function. In addition to displaying both text and matrix values, you can specify the format to be used in displaying the values, and you can specify when to skip to a new line.
- The general form of the `fprintf` command contains two arguments, one a string and the other a list of variables:

`fprintf(format-string, var,...)`

Output Options ctd.

- Consider the following example:

```
>> x = 5;  
>> fprintf('The value of x is %f \n', x);  
The value of x is 5.000000  
>>
```

- The string, which is the first argument inside the `fprintf` function, contains a placeholder (%) where the value of the variable (in this case, `x`) will be inserted. The placeholder also contains formatting information (see next slide).
- “\n” causes MATLAB to start a new line, we need to use \n, called a *linefeed*, at the end of the string.

Output Options ctd.

Other special format commands used for “fprintf” are listed in the table below

Format	Description
%d	Base 10 values
%e	Exponential notation (3.141593e+00)
%f	Fixed-point notation
%g	The more compact of %e or %f, with no trailing zeros
%E	Same as %e, but uppercase (3.141593E+00)
%G	The more compact of %E or %f, with no trailing zeros
%c	Single character
%s	String of characters
%o	Base 8 (octal)
%u	Base 10
%x	Base 16 (hexadecimal), lowercase letters
%X	Same as %x, uppercase letters

Examples

```
>> B = [8.8 7.7 ; ...  
        8800 7700];  
fprintf('X is %f meters or %f mm\n', 9.9, 9900, B)
```

```
X is 9.900000 meters or 9900.000000 mm  
X is 8.800000 meters or 8800.000000 mm  
X is 7.700000 meters or 7700.000000 mm
```

```
>> a = [1.02 3.04 5.06];  
fprintf('%d\n', a);  
1.020000e+000  
3.040000e+000  
5.060000e+000
```

```
>> a = [1 9 23];  
>> fprintf('%u\n', a);  
1  
9  
23  
>> fprintf('%o\n', a);  
1  
11  
27  
>> fprintf('%x\n', a);  
1  
9  
17
```

```
>> x = [2 987654321];  
>> fprintf('x' in degeri: %f \n',x)  
x' in degeri: 2.000000  
x' in degeri: 987654321.000000  
>> fprintf('The value of x: %e \n',x)
```

The value of x: 2.000000e+000

The value of x: 9.876543e+008

>> fprintf('The value of x: %g \n',x)

The value of x: 2

The value of x: 9.87654e+008