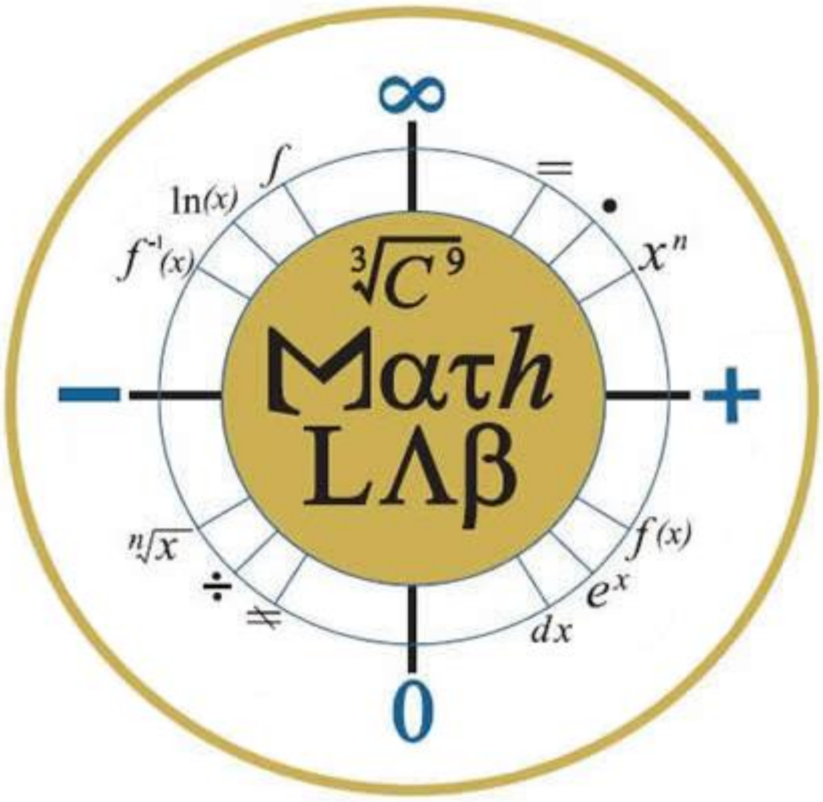
Software for Engineer Design



Lecture 01

Introduction

Subjects:

- ✓ **Theoretical Foundation**
- ✓ **The Matlab Interface**
- ✓ **Basic functionality of Matlab**

Keyword

Field, skill, package, simulation, image processing, differential equation.

Abstract

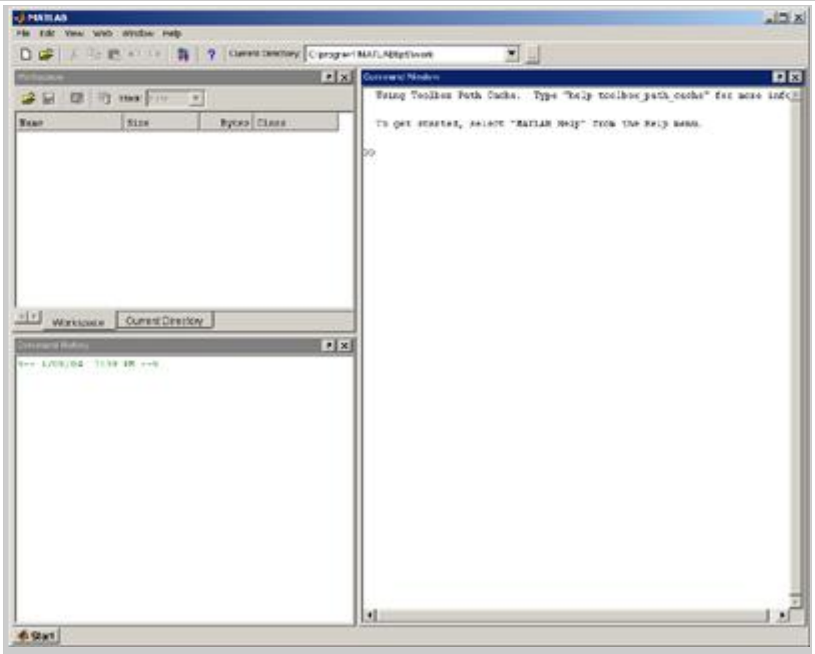
Matlab is a widely used tool in all Engineering fields. There is no doubt that knowing at least the basics of Matlab is a valuable, if not essential skill to have. Matlab is a very versatile software package that can be expanded by any interest group to match their special computational needs. It is thus not only used for simple or complex arithmetic, but also in image processing, statistical analysis, simulation of many kinds, differential equations, etc.

We begin by looking at some of the very basic functionality of Matlab, like variable declarations, arithmetic, and vector and matrix data structures and manipulation. We then apply this foundation to more complex problems to solve those problems and learn about methods of displaying the solutions.

This lecture serves to build an understanding of the Matlab interface, basic mathematical operations, and the fundamental data structures.

1.1 - Theoretical Foundation

The Matlab Interface (Figure 1.1) is divided into 3 windows: The *Workspace* (upper left corner) lists all defined variables and m-files (discussed later); the *Command History* (lower left corner) lists the ordered sequence of commands issued in this session; and the *Command Window* (right side) is the main source of input and output of commands, variable definitions, and errors.

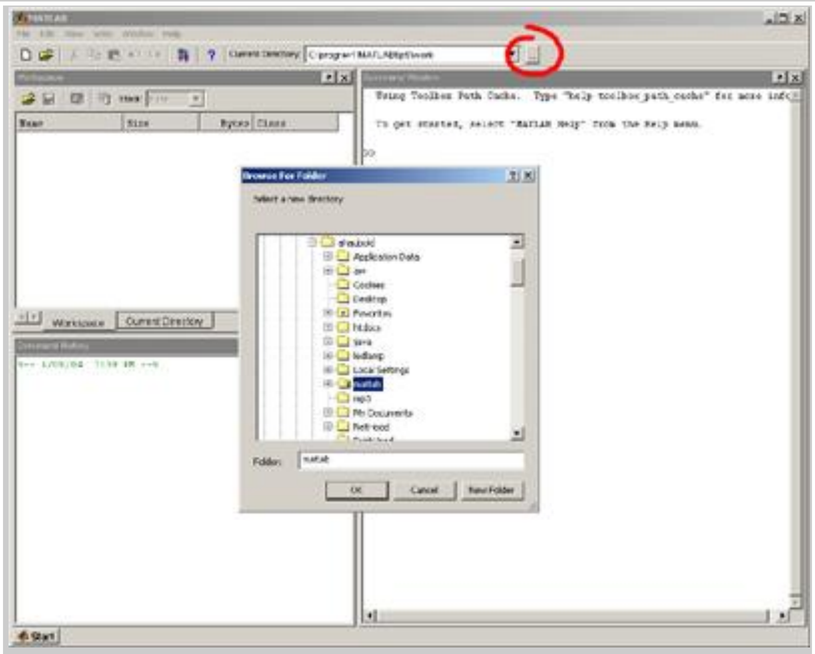


```
Using Toolbox Path Cache.  Type "help toolbox_path_cache" for more info.

To get started, select "MATLAB Help" from the Help menu.

>>
```

Figure 1.1
Click image to enlarge, or click here to open

In the toolbar of the interface, we can designate a directory from which to read and to which to write m-files. It is recommended to create a new directory (e.g. *matlab*) in your account (the *H:* drive in case of the Gateway Lab), which will host any m-files, exercises, and assignments. For simplicity, keep all Matlab related files in this directory.

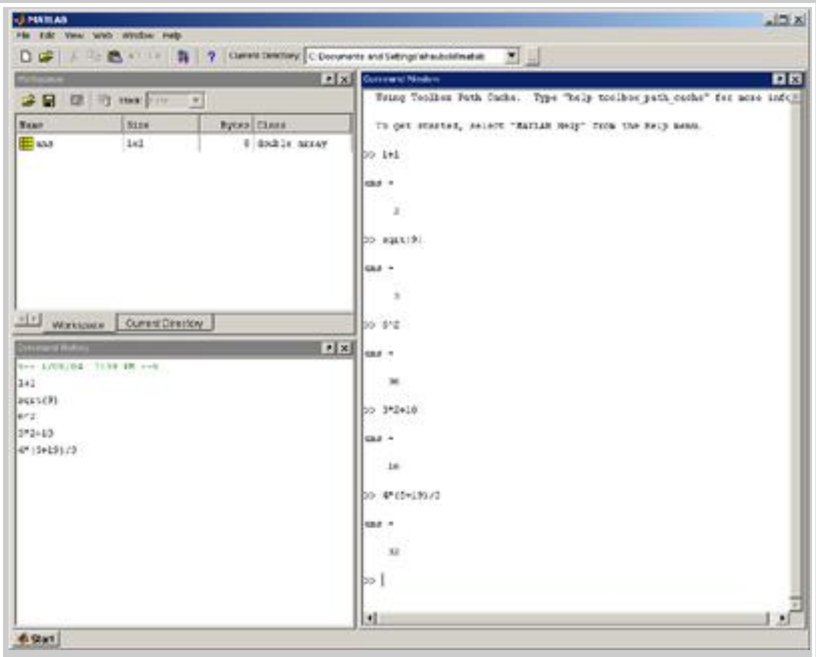
Whenever starting Matlab, change the *Current Directory* to this directory to access relevant files.



```
Using Toolbox Path Cache.  Type "help toolbox_path_cache" for more info.

To get started, select "MATLAB Help" from the Help menu.

>>
```

Figure 1.2
Click image to enlarge, or click here to open

Figure 1.3 shows a few simple arithmetic expressions that are evaluated in the *Command Window*: Exponentials are typed in as `x^y` , square roots as `sqrt(x)` . The *Workspace* and *Command History* Windows update as expressions are evaluated in the *Command Window*. Any variable that appears in the *Workspace* can be used. `ans` denotes the variable for the last evaluated answer.



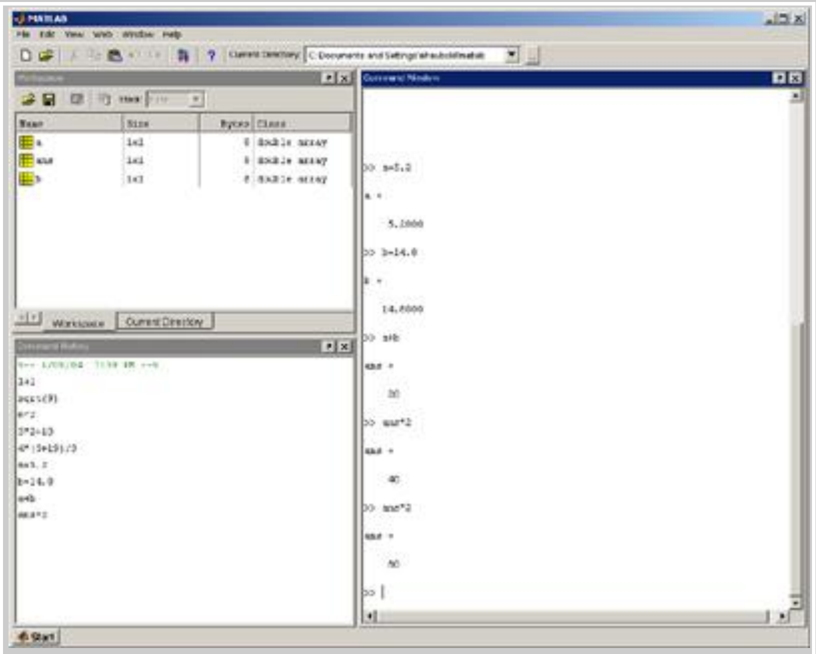
Name	Size	Bytes	Class
ans	1x1	8	double array

```
Using Toolbox Path Cache.  Type "help toolbox_path_cache" for more info

To get started, select "MATLAB Help" from the Help menu.

>> 1+1

ans =

     2

>> sqrt(9)

ans =

     3

>> 6^2

ans =

    36

>> 3*2+10

ans =

    16

>> 4*(5+19)/3

ans =

    32

>> |
```

```
%-- 1/09/04  7:38 PM --%
1+1
sqrt(9)
6^2
3*2+10
4*(5+19)/3
```

Figure 1.3
Click image to enlarge, or click here to open

Variable are formed by assigning letters or words to numbers or expressions, as shown in Figure 1.4. All defined variables in the *Workspace* can be used throughout the session.



Name	Size	Bytes	Class
a	1x1	8	double array
ans	1x1	8	double array
b	1x1	8	double array

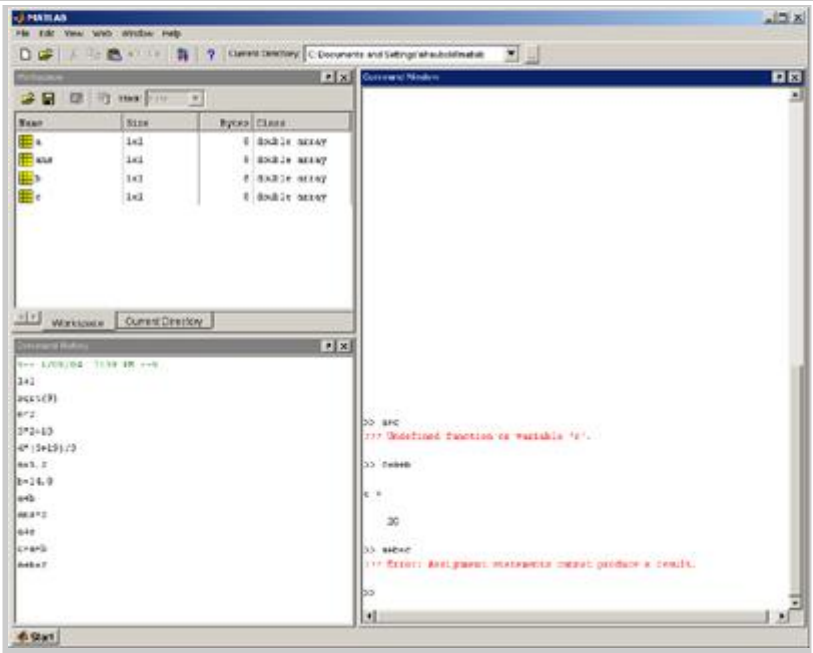
```
>> a=5.2

a =

    5.2000

>> b=14.8

b =

   14.8000

>> a+b

ans =

    20

>> ans*2

ans =

    40

>> ans*2

ans =

    80

>> |
```

```
%-- 1/09/04  7:38 PM --%
1+1
sqrt(9)
6^2
3*2+10
4*(5+19)/3
a=5.2
b=14.8
a+b
ans*2
```

Figure 1.4
Click image to enlarge, or click here to open

Using variables that haven't been defined returns an appropriate error.

Declarations must follow the convention of assignment where the left side of an equal sign is always a variable and the right side an arbitrary expression.



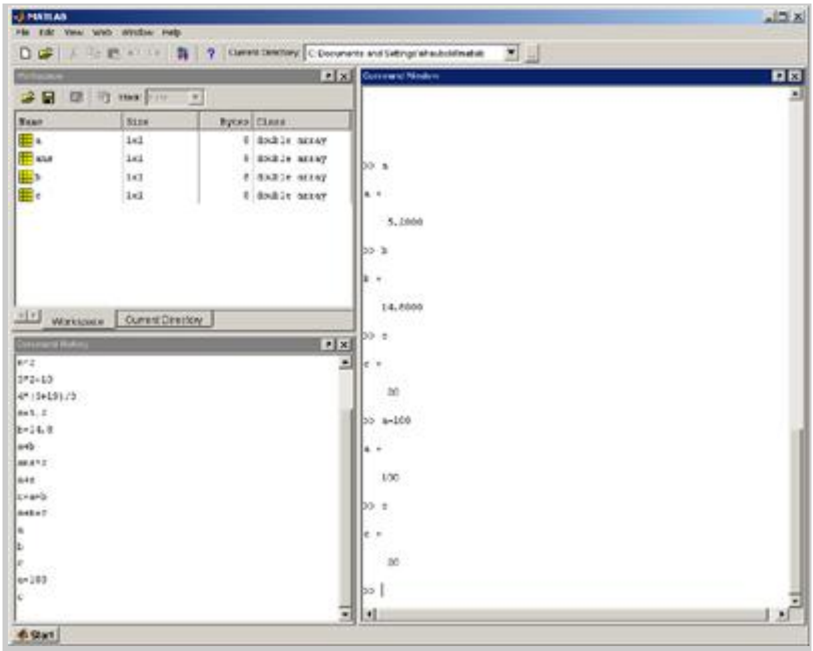
Name	Size	Bytes	Class
a	1x1	8	double array
ans	1x1	8	double array
b	1x1	8	double array
c	1x1	8	double array

```
1+1
sqrt(9)
e^2
3*2-13
4*(5+13)/3
a=5.2
b=14.8
a+b
ans*2
a+c
c=a+b
a+b=c
```

```
>> a+c
??? Undefined function or variable 'c'.

>> c=a+b

c =

    20

>> a+b=c
??? Error: Assignment statements cannot produce a result.

>>
```

Figure 1.5
Click image to enlarge, or click here to open

Matlab handles variables by their value as opposed to symbolically. When a variable has been assigned to a value and is used in another expression, the variable acts as the value rather than a reference to the variable name. We can observe this behavior in the series of expressions in Figure 1.6. Even though `c=a+b` has been declared in terms of variables **a** and **b**, when changing the value of **a**, **c** remains to be the sum of the initial **a** and **b**.



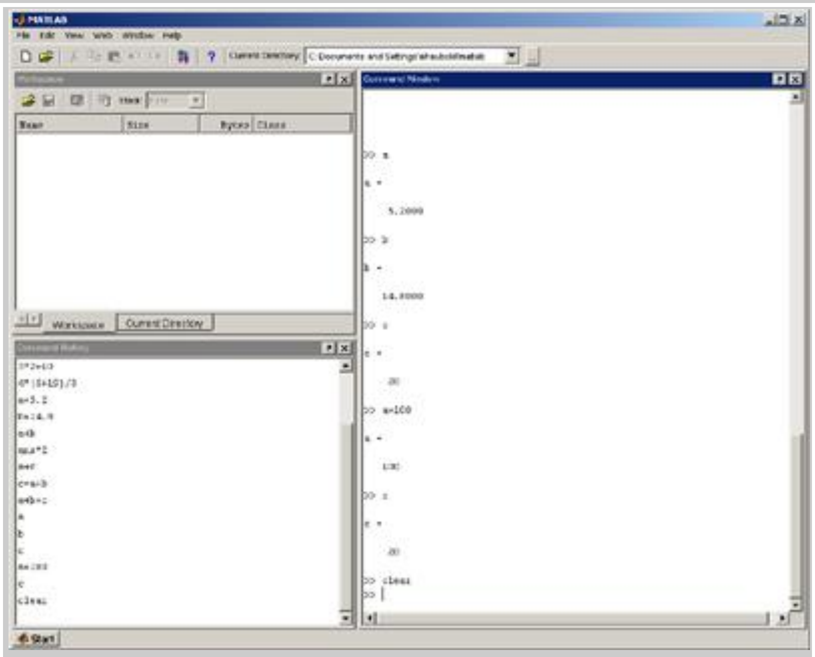
Name	Size	Bytes	Class
a	1x1	8	double array
ans	1x1	8	double array
b	1x1	8	double array
c	1x1	8	double array

```
e^2
3*2-13
4*(5+13)/3
a=5.2
b=14.8
a+b
ans*2
a+c
c=a+b
a+b=c
a
b
c
a=100
c
```

```
>> a

a =

    5.2000

>> b

b =

   14.8000

>> c

c =

    20

>> a=100

a =

   100

>> c

c =

    20

>> |
```

Figure 1.6
Click image to enlarge, or click here to open

The contents of the *Workspace* can be cleared by issuing the command `clear` . This will remove all previously defined variables.

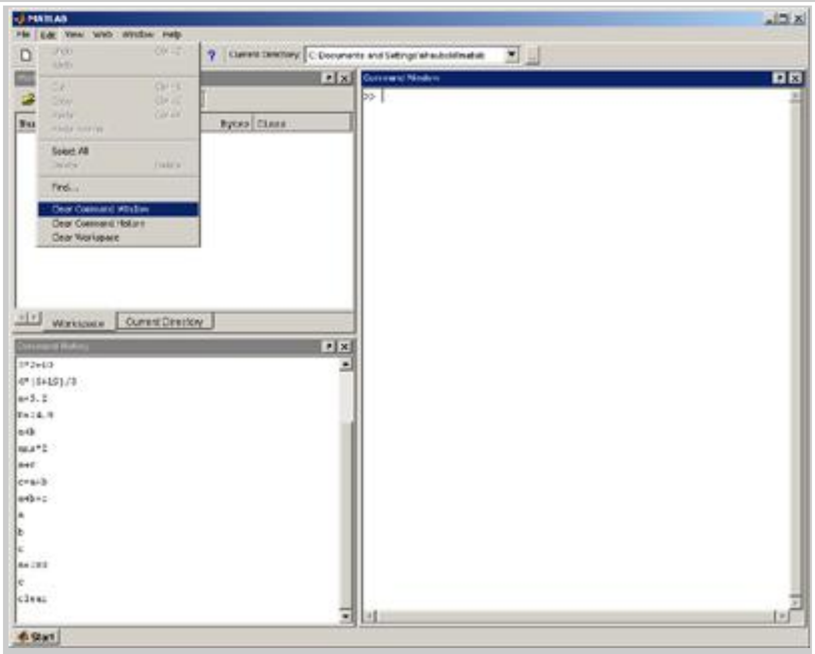

MATLAB
File Edit View Web Window Help
Current Directory: C:\Documents and Settings\...\matlab
Workspace
Name Size Bytes Class
Workspace Current Directory
Command History
Command Window
Start

```
3^2+10
4*(5+15)/3
a=5.2
b=14.8
a+b
m=a*2
a+c
c=a+b
a+b+c
a
b
c
a=100
c
clear
```

```
>> a

a =

    5.2000

>> b

b =

   14.8000

>> c

c =

    20

>> a=100

a =

   100

>> c

c =

    20

>> clear
>> 
```

Figure 1.7
Click image to enlarge, or click here to open

The contents of all windows can be cleared from the *Edit* menu.



```
3^2+10
4*(5+15)/3
a=5.2
b=14.8
a+b
m=a*2
a+c
c=a+b
a+b+c
a
b
c
a=100
c
clear
```

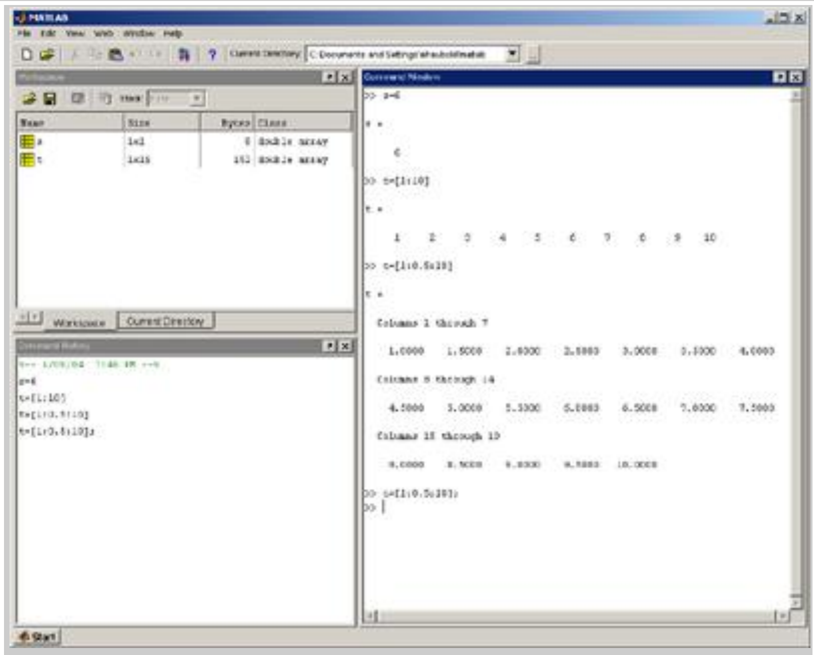
```
>> 
```

Figure 1.8
Click image to enlarge, or click here to open

[x:z] defines a vector of numbers starting at **x** and ending at **z**, where each element of the array is incremented by **1**. E.g. `[3:9]` defines the vector *[3,4,5,6,7,8,9]*.

[x:y:z] defines a vector of numbers starting at **x** and ending at **z**, where each element of the array is incremented by **y**. E.g. `[10:2:20]` defines the vector *[10,12,14,16,18,20]*.

Every definition explicitely returns the defined quantity, and each expression returns a result. The output can be suppressed by ending the expression with a **;** (semicolon). E.g. `a=[1:1000000]` returns a vector of 1000000 elements, a quantity that we do not necessarily want to see see element by element. The expression `a=[1:1000000];` will merely define the variable without returning unwanted output.



```
>> s=6

s =

     6

>> t=[1:10]

t =

     1     2     3     4     5     6     7     8     9    10

>> t=[1:0.5:10]

t =

  Columns 1 through 7

    1.0000    1.5000    2.0000    2.5000    3.0000    3.5000    4.0000

  Columns 8 through 14

    4.5000    5.0000    5.5000    6.0000    6.5000    7.0000    7.5000

  Columns 15 through 19

    8.0000    8.5000    9.0000    9.5000   10.0000

>> s=[1:0.5:10];
>> 
```

MATLAB
File Edit View Web Window Help
Current Directory: C:\Documents and Settings\...\matlab
Workspace
Current Directory
Command History
Command Window
Start

Name	Size	Bytes	Class
s	1x1	8	double array
t	1x19	152	double array

```
%-- 1/09/04  7:46 PM --%
s=6
t=[1:10]
t=[1:0.5:10]
t=[1:0.5:10];
```

Figure 1.9
Click image to enlarge, or click here to open

Figure 1.10 pictorially shows the difference between scalars and vectors: a scalar is a single element, while a vector is a series (or a list) of elements (of scalars).

In Figure 1.9 we have declared a scalar **s** and a vector **t**.

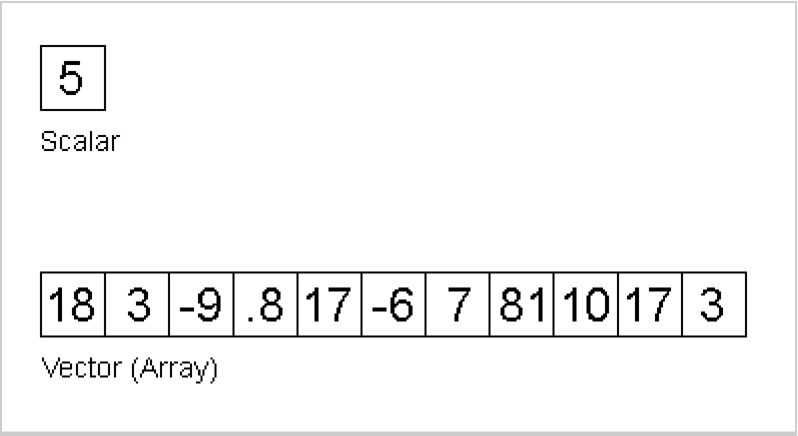
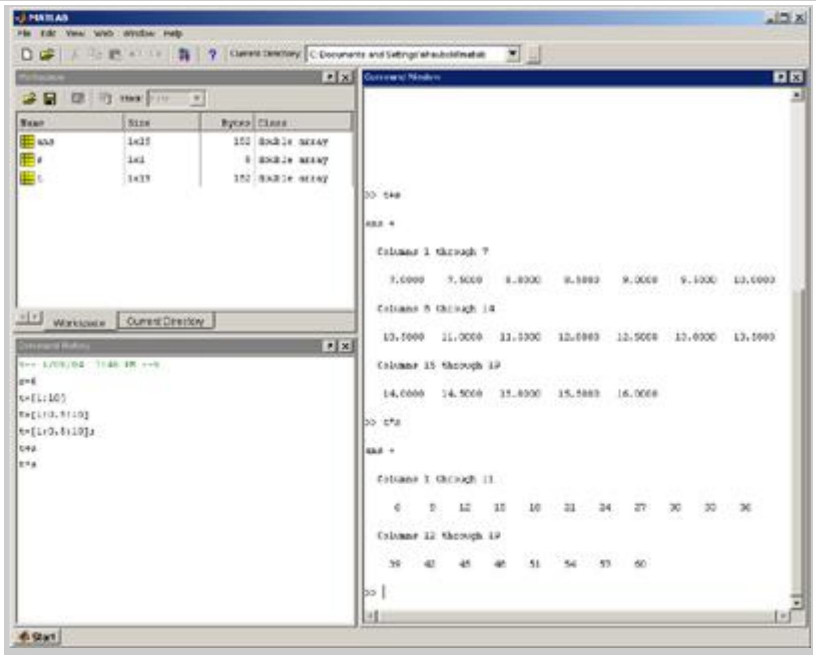


Figure 1.10

Having defined a vector and a scalar, we can perform arithmetic operations. Adding (multiplying, etc.) a vector by a scalar adds (multiplies, etc.) each element of the vector by the scalar. The result of `s * t` can be assigned to an output variable **u** as in `u = s * t` . Variable **u** then references the resulting vector.



Name	Size	Bytes	Class
ans	1x19	152	double array
s	1x1	8	double array
t	1x19	152	double array

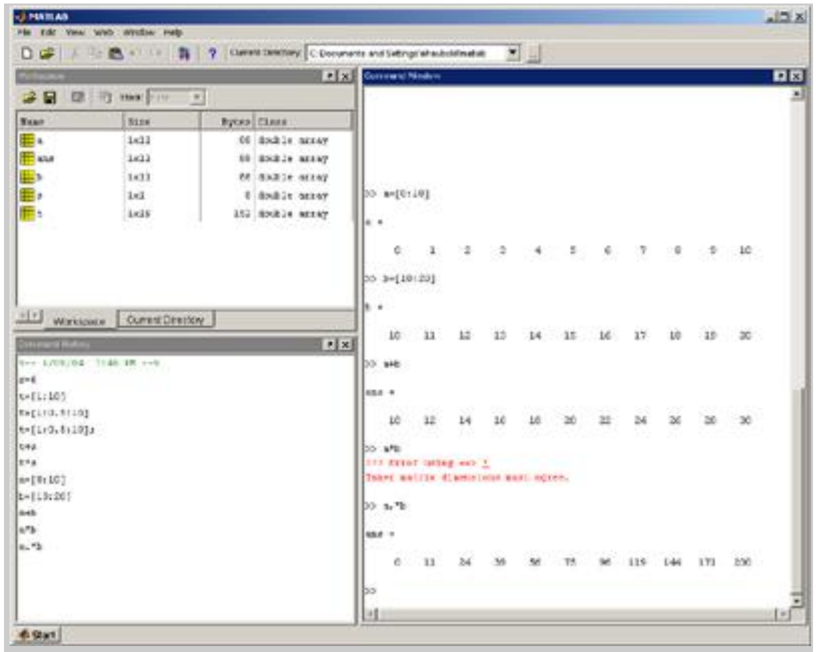
```
s=6
t=[1:10]
t=[1:0.5:10]
t=[1:0.5:10];
t+s
t*s
```

```
>> t+s

ans =

  Columns 1 through 7

    7.0000    7.5000    8.0000    8.5000    9.0000    9.5000   10.0000

  Columns 8 through 14

   10.5000   11.0000   11.5000   12.0000   12.5000   13.0000   13.5000

  Columns 15 through 19

   14.0000   14.5000   15.0000   15.5000   16.0000

>> t*s

ans =

  Columns 1 through 11

     6     9    12    15    18    21    24    27    30    33    36

  Columns 12 through 19

    39    42    45    48    51    54    57    60

>>
```

Figure 1.11
Click image to enlarge, or click here to open

Two same-sized vectors **a** and **b** are added/subtracted element-by-element using the **+** and **-** operators, as is done in matrix addition and subtraction. (Vectors are merely matrices with only one row).

In order to multiply two vectors element-by-element, the operator ***** (or **/**) must be preceded by a period **.**, as shown in Figure 1.12.



Name	Size	Bytes	Class
a	1x11	88	double array
ans	1x11	88	double array
b	1x11	88	double array
s	1x1	8	double array
t	1x19	152	double array

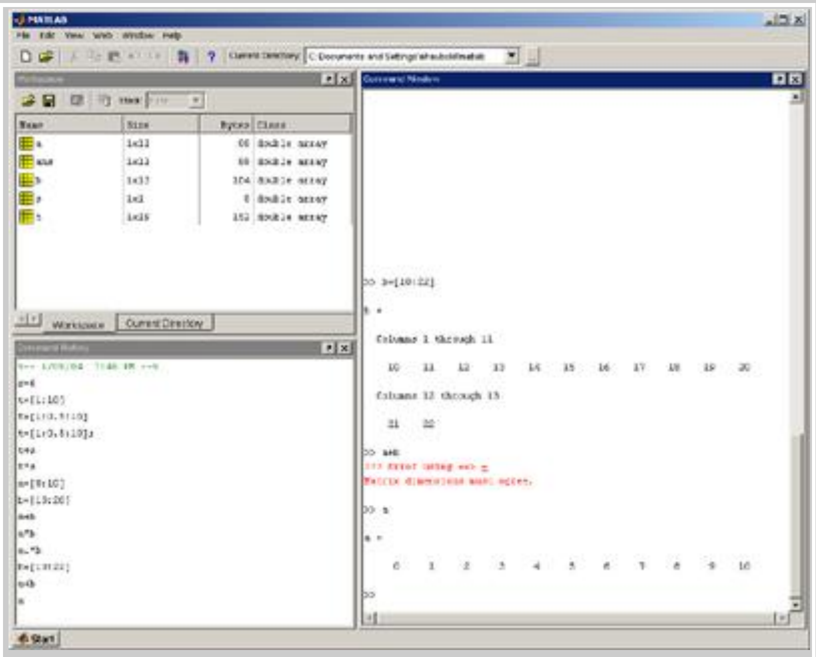
```
s=6
t=[1:10]
t=[1:0.5:10]
t=[1:0.5:10];
t+s
t*s
a=[0:10]
b=[10:20]
a+b
a*b
a.*b
```

```
>> a=[0:10]

a =

     0     1     2     3     4     5     6     7     8     9    10

>> b=[10:20]

b =

    10    11    12    13    14    15    16    17    18    19    20

>> a+b

ans =

    10    12    14    16    18    20    22    24    26    28    30

>> a*b
??? Error using ==> *
Inner matrix dimensions must agree.

>> a.*b

ans =

     0    11    24    39    56    75    96   119   144   171   200

>>
```

Figure 1.12
Click image to enlarge, or click here to open

In order to perform any element-by-element operation on two vectors, their lengths must match, otherwise Matlab returns an error.

Figure 1.14 shows how two differently sized vectors cannot be combined element-by-element, because there exist ambiguities as to how an operation is defined between a scalar and a "nothing".



Name	Size	Bytes	Class
a	1x11	88	double array
ans	1x11	88	double array
b	1x13	104	double array
g	1x1	8	double array
t	1x19	152	double array

```
g=6
t=[1:10]
t=[1:3,5:10]
t=[1:3,5:10];
t+g
t*g
a=[0:10]
b=[10:20]
a+b
a*b
a.*b
b=[10:22]
a+b
a
```

```
>> b=[10:22]

b =

  Columns 1 through 11

    10    11    12    13    14    15    16    17    18    19    20

  Columns 12 through 13

    21    22

>> a+b
??? Error using ==> +
Matrix dimensions must agree.

>> a

a =

     0     1     2     3     4     5     6     7     8     9    10

>>
```

Figure 1.13
Click image to enlarge, or click <u>here</u> to open

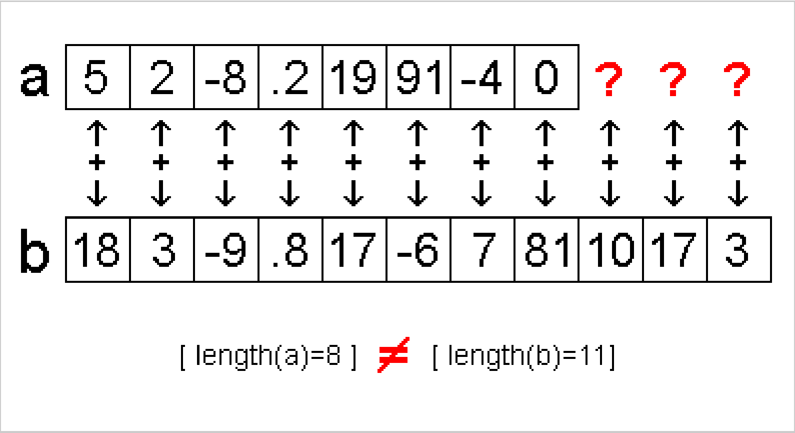
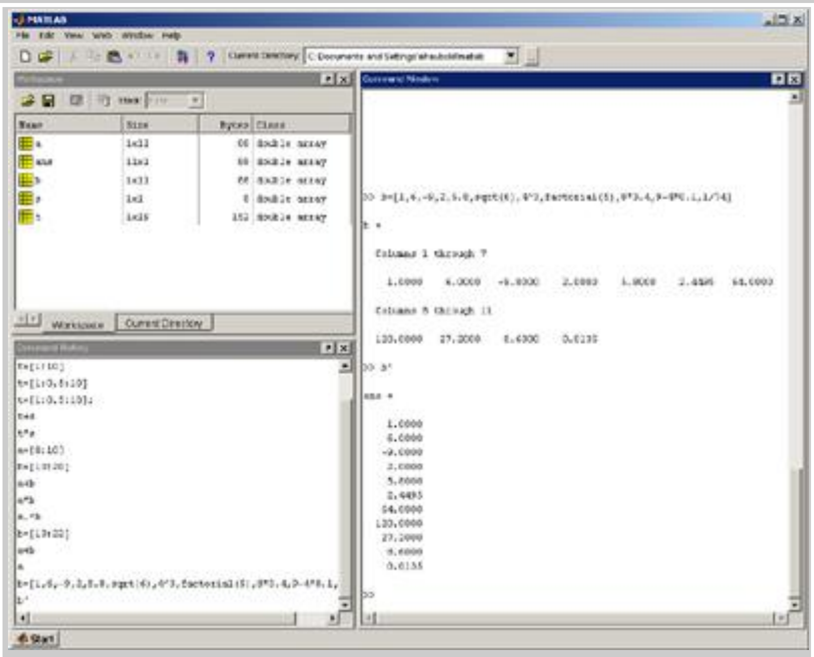


Figure 1.14

Besides using the range notation, vectors can also be defined by enumerating the elements one by one (Figure 1.15). Transposing can be accomplished by using a single quote (') after the variable name, e.g. `b'` . Two same size vector can then be multiplied in matrix-terms by issuing the command `a * b'` .

Figure 1.16 pictorially shows a transpose and a double transpose operation.



Name	Size	Bytes	Class
a	1x11	88	double array
ans	11x1	88	double array
b	1x11	88	double array
s	1x1	8	double array
t	1x19	152	double array

```
t=[1:10]
t=[1:0.5:10]
t=[1:0.5:10];
t+4
t*s
a=[0:10]
b=[10:20];
a+b
a*b
a.*b
b=[12:22]
a+b
a
b=[1,6,-9,2,5.8,sqrt(6),4^3,factorial(5),8*3.4,9-4*8.1,
b'
```

```
>> b=[1,6,-9,2,5.8,sqrt(6),4^3,factorial(5),8*3.4,9-4*8.1,1/74]

b =

  Columns 1 through 7

    1.0000    6.0000   -9.0000    2.0000    5.8000    2.4495   64.0000

  Columns 8 through 11

  120.0000   27.2000    9.6000    0.0135

>> b'

ans =

    1.0000
    6.0000
   -9.0000
    2.0000
    5.8000
    2.4495
   64.0000
  120.0000
   27.2000
    9.6000
    0.0135

>>
```

Figure 1.15
Click image to enlarge, or click *<u>here</u>* *to open*

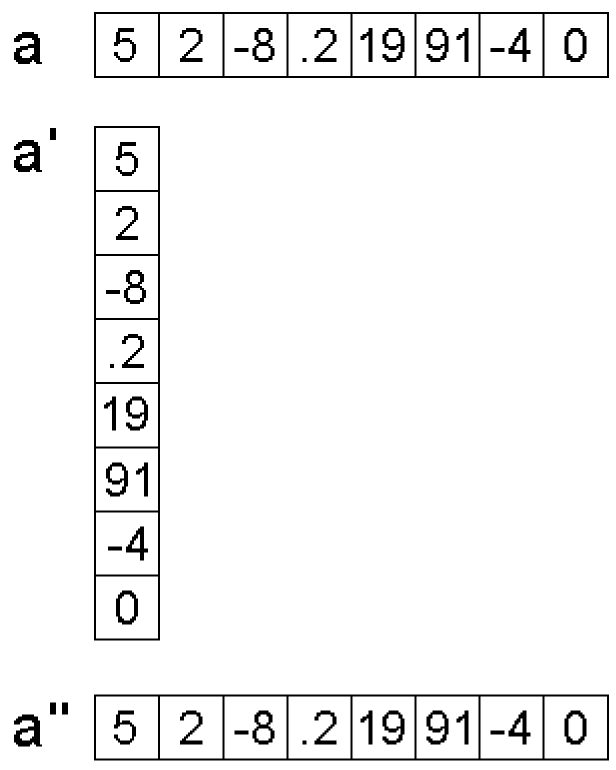
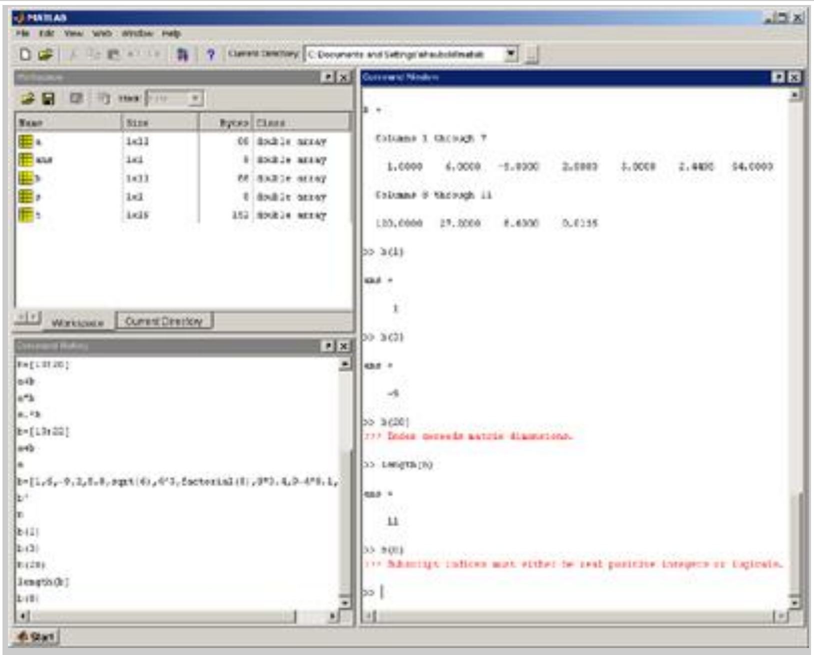


Figure 1.16

Elements in a vector can be addressed individually by enclosing the elements index in parentheses after the vector variable name, e.g. `b(3)` . The vector's elements' indices start at 1 and stop at the length of the vector. Index 0 (zero) is not used, even though it is commonly done so in programming languages.

The command `length(b)` for any matrix **b** evaluates to the number of elements in vector **b**.

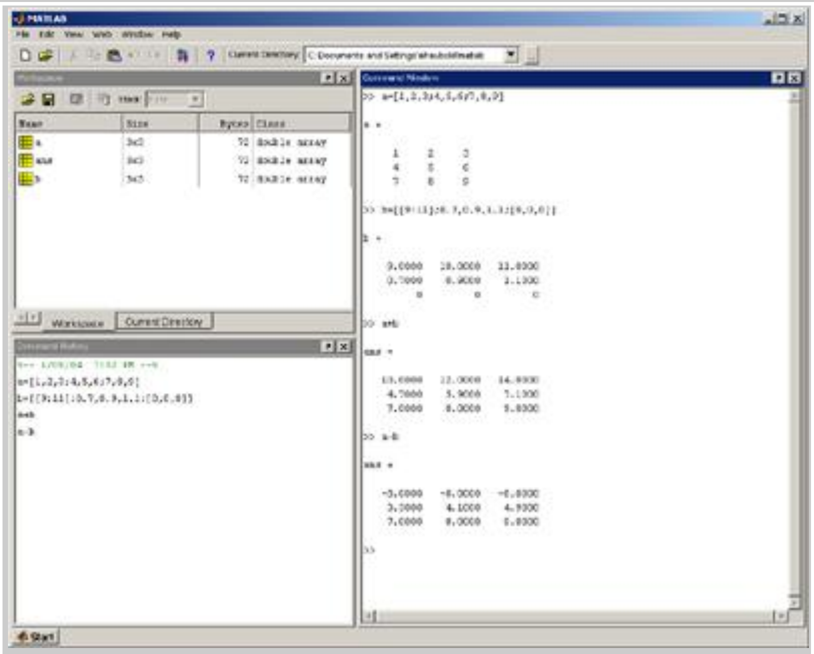


```
b =

  Columns 1 through 7

    1.0000    6.0000   -9.0000    2.0000    5.8000    2.0000   64.0000

  Columns 8 through 11

  120.0000   27.[illegible]    [illegible]    [illegible]

>> b(1)

ans =

     1

>> b(3)

ans =

    -9

>> b(20)
??? Index exceeds matrix dimensions.

>> length(b)

ans =

    11

>> b(0)
??? Subscript indices must either be real positive integers or logicals.

>>
```

Figure 1.17
Click image to enlarge, or click <u>here</u> to open

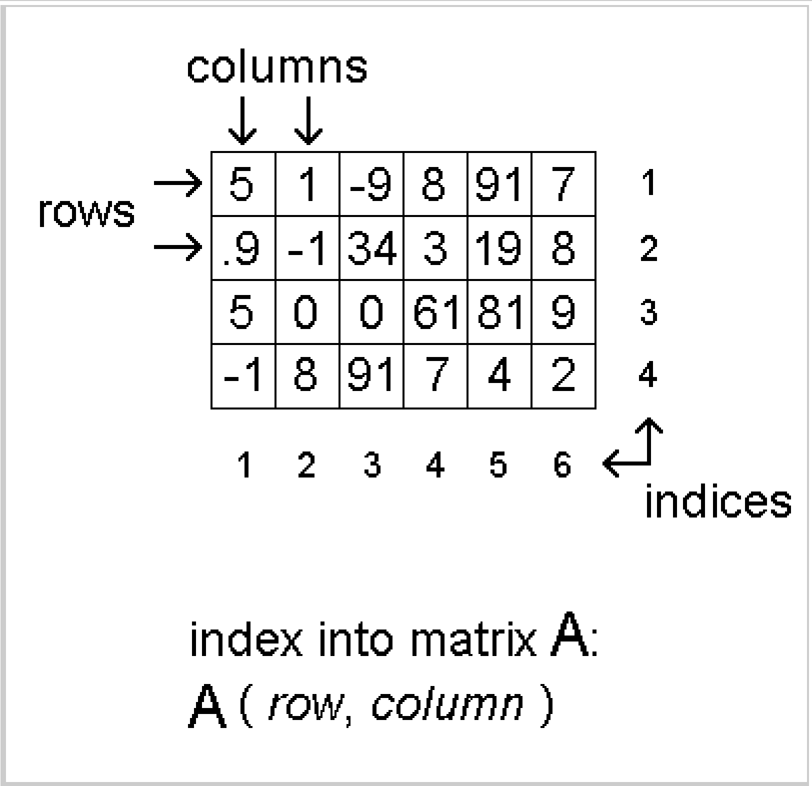
A matrix is defined by rows and columns. A row of elements is delimited by commas, e.g. `1,2,3` . Columns are delimited by semi-colons, e.g. `5;6;7` . A matrix can thus be defined using any of the vector defining notations used previously, e.g. `[1,2,3;4,5,6;7,8,9]` or `[[1:3];[4:6];[7:9]]` .

Figure 1.19 pictorially shows what are considered rows and columns of a matrix, indexing conventions, and the general form of indexing into a matrix.



```
>> a=[1,2,3;4,5,6;7,8,9]

a =

     1     2     3
     4     5     6
     7     8     9

>> b=[[9:11];0.7,0.9,1.1;[0,0,0]]

b =

    9.0000   10.0000   11.0000
    0.7000    0.9000    1.1000
         0         0         0

>> a+b

ans =

   10.0000   12.0000   14.0000
    4.7000    5.9000    7.1000
    7.0000    8.0000    9.0000

>> a-b

ans =

   -8.0000   -8.0000   -8.0000
    3.3000    4.1000    4.9000
    7.0000    8.0000    9.0000

>>
```

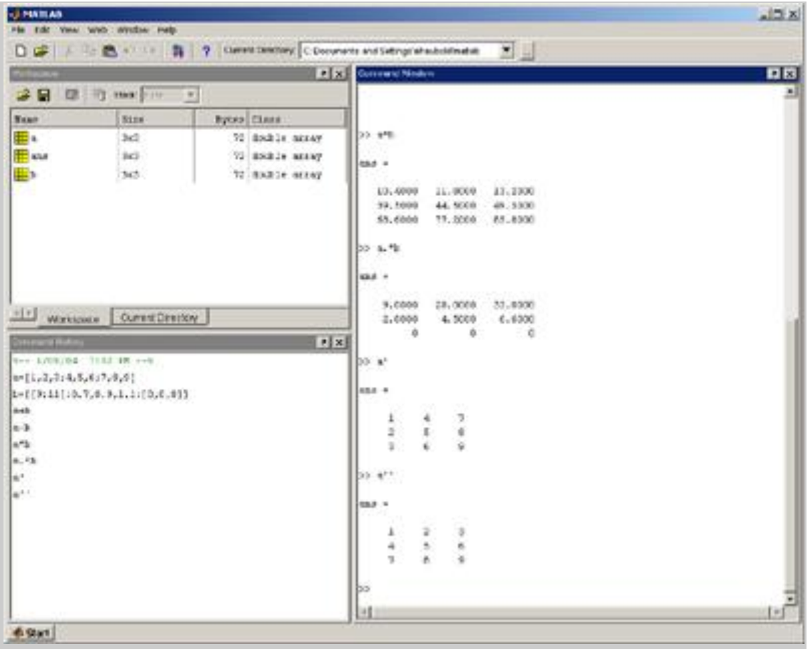
Figure 1.18
Click image to enlarge, or click <u>here</u> to open



$$\begin{array}{c|cccccc|c}
 & & & & & & & \\
\text{rows}\rightarrow & 5 & 1 & -9 & 8 & 91 & 7 & 1 \\
\rightarrow & .9 & -1 & 34 & 3 & 19 & 8 & 2 \\
 & 5 & 0 & 0 & 61 & 81 & 9 & 3 \\
 & -1 & 8 & 91 & 7 & 4 & 2 & 4 \\
\hline
 & 1 & 2 & 3 & 4 & 5 & 6 &
\end{array}$$

Figure 1.19

Matrices of same dimensions can be added and subtracted as is defined in matrix arithmetic. In multiplication, however, we achieve different results depending on whether we multiply two matrices by using `a*b` or `a.*b` , where the latter is element-by-element multiplication.



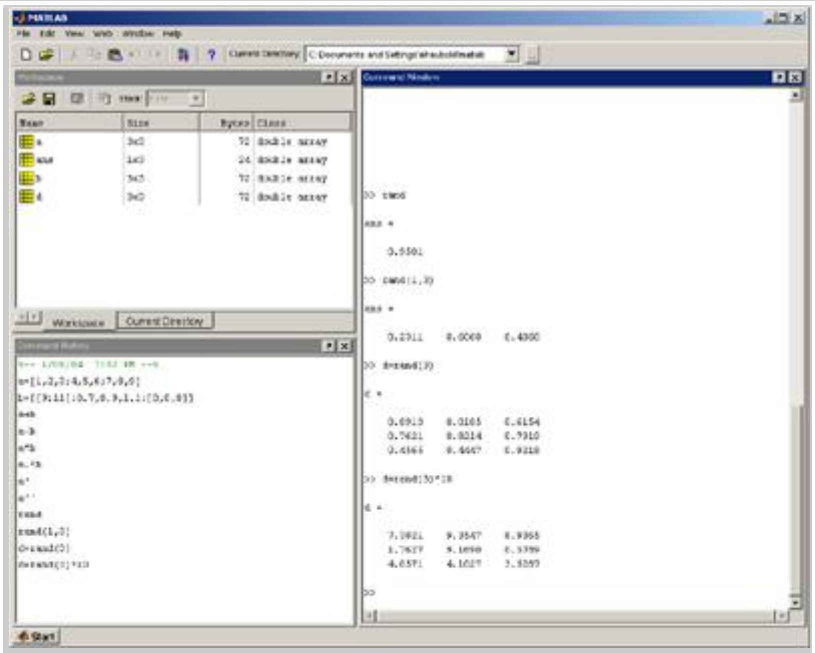
Name	Size	Bytes	Class
a	3x3	72	double array
ans	3x3	72	double array
b	3x3	72	double array

```
a=[1,2,3;4,5,6;7,8,9]
b=[illegible]
a+b
a-b
a*b
a.*b
a'
a''
```

```
>> a*b

ans =

   13.4000   11.0000   13.2300
   39.5000   44.5000   49.5300
   65.6000   77.2000   85.8300

>> a.*b

ans =

    9.0000   28.0000   33.0000
    2.0000    4.5000    6.6000
         0         0         0

>> a'

ans =

     1     4     7
     2     5     8
     3     6     9

>> a''

ans =

     1     2     3
     4     5     6
     7     8     9

>>
```

Figure 1.20
Click image to enlarge, or click here to open

Random numbers, vectors, and matrices can be formed using the function **rand**. **rand** takes as arguments the number of rows, and number of columns: **rand(rows,cols)** and returns a matrix of that size with random numbers ranging from 0 to 1. The function can easily be multiplied and/or added to scalars in order to produce random numbers in any range, e.g. a random number between 0 and 10: `rand(1,1) * 10` , or a random number between 10 and 100: `(rand(1,1) * 90) + 10` .


MATLAB
File Edit View Web Window Help
Current Directory: C:\Documents and Settings\ahaubold\matlab
Workspace
Command History
Command Window
Workspace
Current Directory
Start

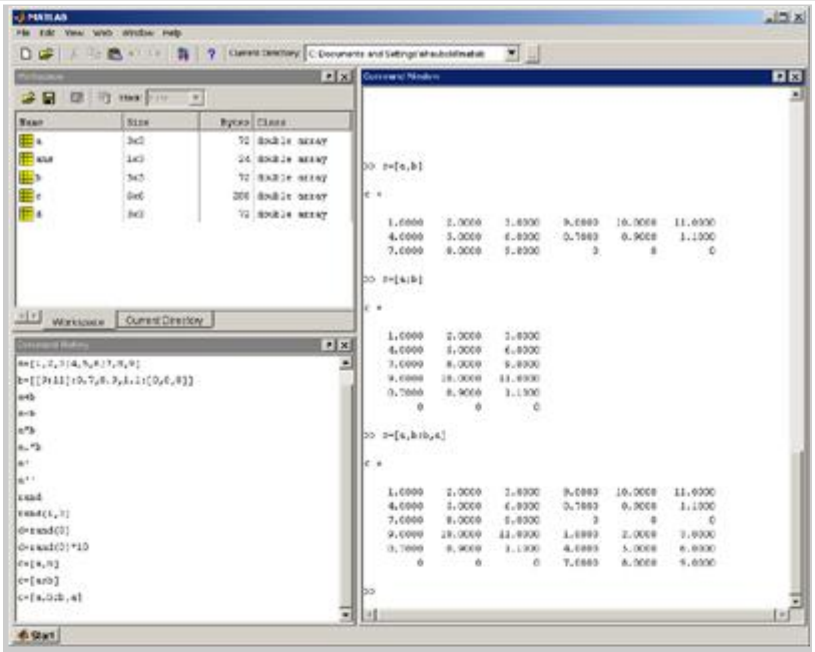
```
Name    Size    Bytes  Class
a       3x3        72  double array
ans     1x3        24  double array
b       3x3        72  double array
d       3x3        72  double array
```

```
a=[1,2,3;4,5,6;7,8,9]
b=[[9:11];[0.7,0.9,1.1];[0,0,0]]
a+b
a-b
a*b
a.*b
a'
a''
rand
rand(1,3)
c=rand(3)
d=rand(3)*10
```

```
>> rand

ans =

    0.9501

>> rand(1,3)

ans =

    0.2311    0.6068    0.4860

>> c=rand(3)

c =

    0.8913    0.0185    0.6154
    0.7621    0.8214    0.7919
    0.4565    0.4447    0.9218

>> d=rand(3)*10

d =

    7.3821    9.3547    8.9365
    1.7627    9.1690    0.5789
    4.0571    4.1027    3.5287

>>
```

Figure 1.21
Click image to enlarge, or click here to open

Matrices and vectors can be appended to form larger matrices and vectors. Given two similarly or differently sized vectors **a** and **b**, a vector **c** can be formed by `c=[a,b]` . **c** is now as long as **a** and **b** combined. The same holds for matrices. A 3x3 matrix **a** can be combined with a 3x5 matrix **b** by `[a,b]` but not by `[a;b]` , while a 3x3 matrix **a** can be combined with a 5x3 matrix **b** by `[a;b]` but not by `[a,b]` . Depending on which way matrices are appended, one of the dimensions (rows or columns) must match.


MATLAB
File Edit View Web Window Help
Current Directory: C:\Documents and Settings\ahaubold\matlab
Workspace
Command History
Command Window
Workspace
Current Directory
Start

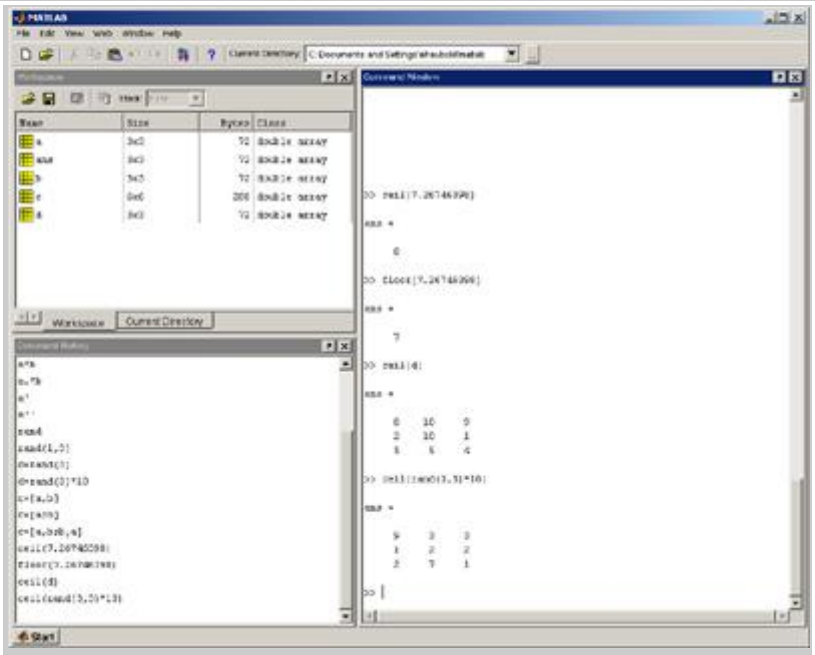
```
Name    Size    Bytes  Class
a       3x3        72  double array
ans     1x3        24  double array
b       3x3        72  double array
c       6x6       288  double array
d       3x3        72  double array
```

```
a=[1,2,3;4,5,6;7,8,9]
b=[[9:11];[0.7,0.9,1.1];[0,0,0]]
a+b
a-b
a*b
a.*b
a'
a''
rand
rand(1,3)
d=rand(3)
d=rand(3)*10
c=[a,b]
c=[a;b]
c=[a,b;b,a]
```

```
>> c=[a,b]

c =

    1.0000    2.0000    3.0000    9.0000   10.0000   11.0000
    4.0000    5.0000    6.0000    0.7000    0.9000    1.1000
    7.0000    8.0000    9.0000         0         0         0

>> c=[a;b]

c =

    1.0000    2.0000    3.0000
    4.0000    5.0000    6.0000
    7.0000    8.0000    9.0000
    9.0000   10.0000   11.0000
    0.7000    0.9000    1.1000
         0         0         0

>> c=[a,b;b,a]

c =

    1.0000    2.0000    3.0000    9.0000   10.0000   11.0000
    4.0000    5.0000    6.0000    0.7000    0.9000    1.1000
    7.0000    8.0000    9.0000         0         0         0
    9.0000   10.0000   11.0000    1.0000    2.0000    3.0000
    0.7000    0.9000    1.1000    4.0000    5.0000    6.0000
         0         0         0    7.0000    8.0000    9.0000

>>
```

Figure 1.22
Click image to enlarge, or click here to open

Functions **ceil(x)** and **floor(x)** round decimal numbers (doubles) up and down to the next integer, respectively. This operation can be performed on scalars, vectors, and matrices.


MATLAB
File Edit View Web Window Help
Current Directory: C:\Documents and Settings\...\matlab
Workspace
Workspace
Current Directory
Command History
Command Window
Start

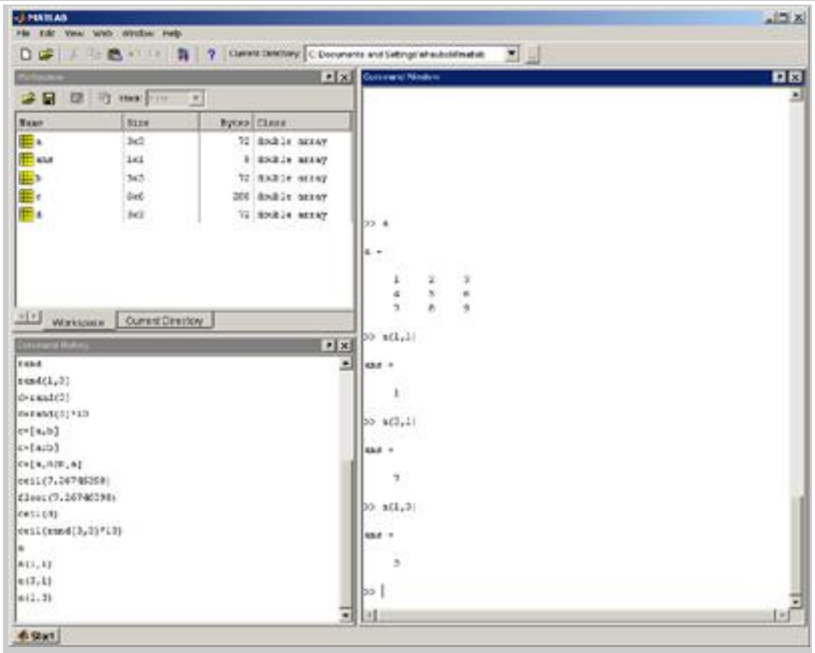
Name	Size	Bytes	Class
a	3x3	72	double array
ans	3x3	72	double array
b	3x3	72	double array
c	6x6	288	double array
d	3x3	72	double array

```
a*b
a.*b
a'
a''
rand
rand(1,3)
d=rand(3)
d=rand(3)*10
c=[a,b]
c=[a;b]
c=[a,b;b,a]
ceil(7.26746398)
floor(7.26746398)
ceil(d)
ceil(rand(3,3)*10)
```

```
>> ceil(7.26746398)

ans =

     8

>> floor(7.26746398)

ans =

     7

>> ceil(d)

ans =

     6    10     9
     2    10     1
     1     5     4

>> ceil(rand(3,3)*10)

ans =

     9     3     3
     1     2     2
     2     7     1

>> |
```

Figure 1.23
Click image to enlarge, or click <u>here</u> to open

Elements in a matrix can be accessed by rows and columns: **A(r,c)**, where **A** is a matrix, **r** is a row, and **c** is a column.


MATLAB
File Edit View Web Window Help
Current Directory: C:\Documents and Settings\...\matlab
Workspace
Workspace
Current Directory
Command History
Command Window
Start

Name	Size	Bytes	Class
a	3x3	72	double array
ans	1x1	8	double array
b	3x3	72	double array
c	6x6	288	double array
d	3x3	72	double array

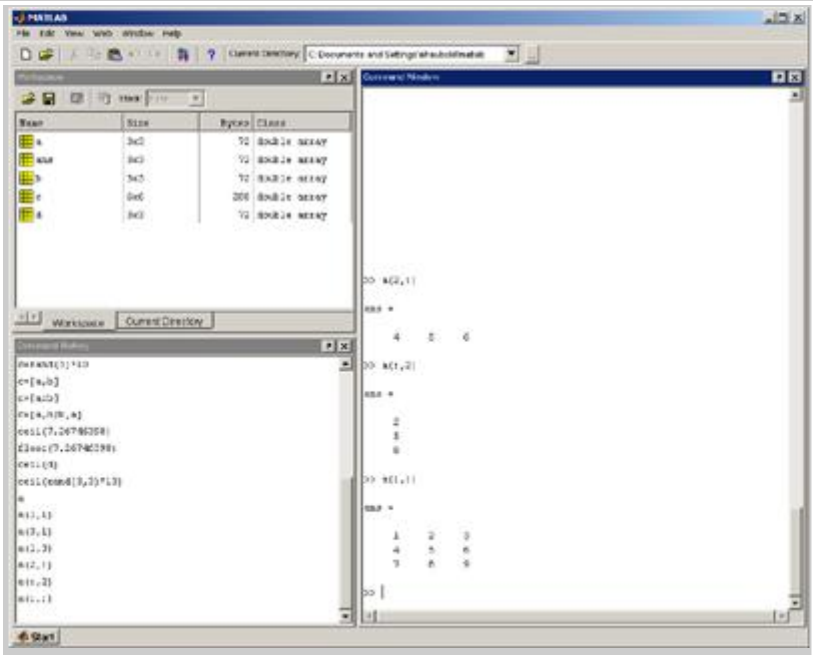
```
rand
rand(1,3)
d=rand(3)
d=rand(3)*10
c=[a,b]
c=[a;b]
c=[a,b;b,a]
ceil(7.26746398)
floor(7.26746398)
ceil(d)
ceil(rand(3,3)*10)
a
a(1,1)
a(3,1)
a(1,3)
```

```
>> a

a =

     1     2     3
     4     5     6
     7     8     9

>> a(1,1)

ans =

     1

>> a(3,1)

ans =

     7

>> a(1,3)

ans =

     5

>> |
```

Figure 1.24
Click image to enlarge, or click <u>here</u> to open

Rows and columns can also be accessed. To access a row in matrix **A**, a colon (**:**) is substituted for the column, and vice versa: `row_r = A(r,:)`, `column_c = A(:,c)`.

The concept of indexing into a matrix by row and column, only row, and only column is pictorially described in Figure 1.26.



MATLAB
File Edit View Web Window Help
Current Directory: C:\Documents and Settings\...\matlab
Workspace
Stack
Workspace
Current Directory
Command History
Command Window
Start

Name	Size	Bytes	Class
a	3x3	72	double array
ans	3x3	72	double array
b	3x3	72	double array
c	6x6	288	double array
d	3x3	72	double array

```
a=rand(3)*10
c=[a,b]
c=[a;b]
c=[a,b;b,a]
ceil(7.26786358)
floor(7.26786358)
ceil(4)
ceil(rand(3,3)*10)
a
a(1,1)
a(3,1)
a(1,3)
a(2,:)
a(:,2)
a(:,:)
```

```
>> a(2,:)

ans =

     4     5     6

>> a(:,2)

ans =

     2
     5
     8

>> a(:,:)

ans =

     1     2     3
     4     5     6
     7     8     9

>> 
```

Figure 1.25
Click image to enlarge, or click here to open

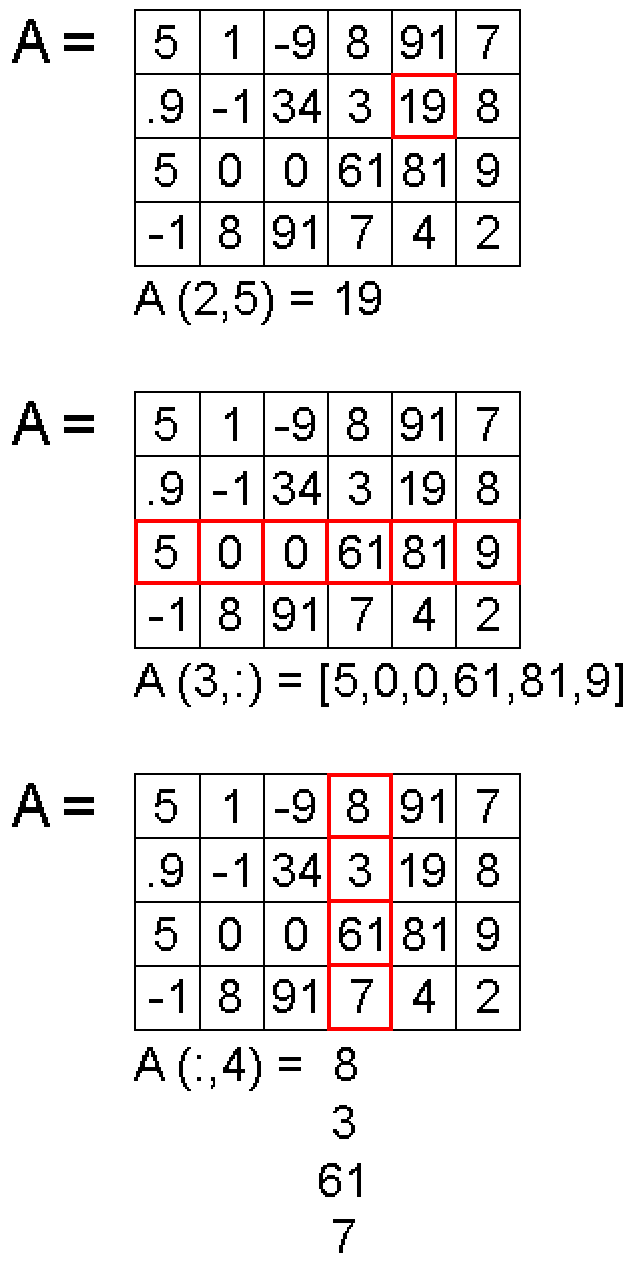
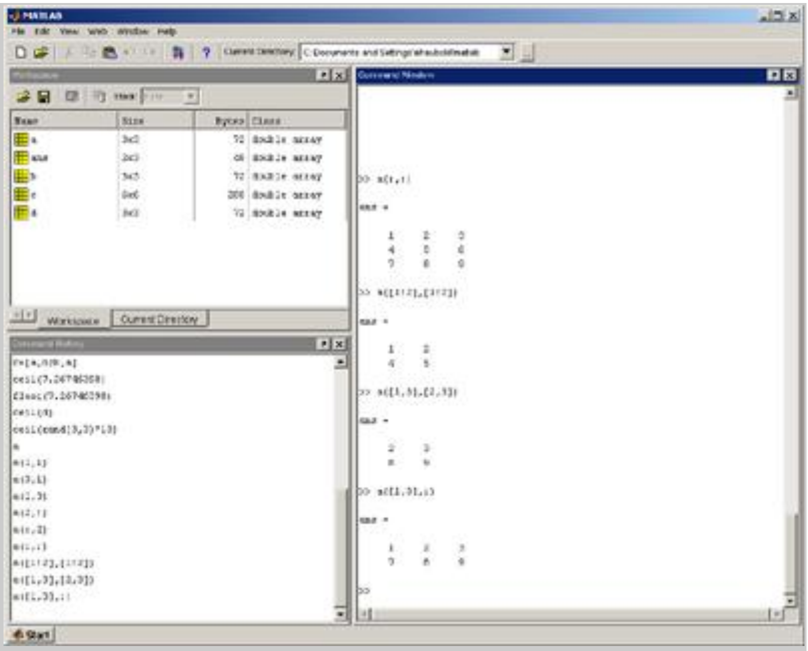

A =
5 1 -9 8 91 7
.9 -1 34 3 19 8
5 0 0 61 81 9
-1 8 91 7 4 2
A (2,5) = 19
A (3,:) = [5,0,0,61,81,9]
A (:,4) = 8 3 61 7

Figure 1.26

It is also possible to extract portions of a matrix by using ranges such as **[1,2]** or **[1:3]** as indices, as shown in Figure 1.27.

The concept of extracting sub-matrices from matrices is shown in Figure 1.28.

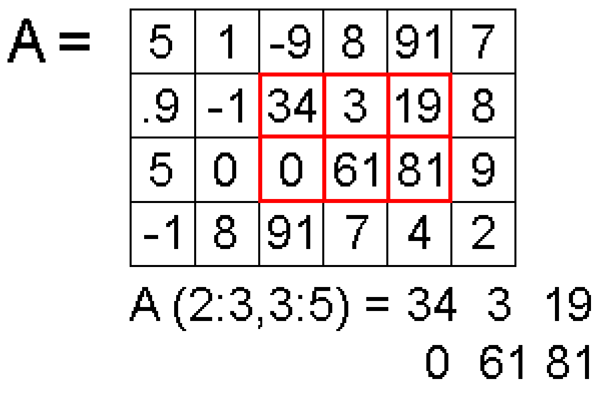

MATLAB
File Edit View Web Window Help
Current Directory: C:\Documents and Settings\[illegible]\matlab
Workspace
Current Directory
Command History
Command Window
Start

Name	Size	Bytes	Class
a	3x3	72	double array
ans	2x3	48	double array
b	3x3	72	double array
c	6x6	288	double array
d	3x3	72	double array

```
c=[a,b;b,a]
ceil(7.26786258)
floor(7.26786258)
ceil(4)
ceil(rand(3,3)*10)
a
a(1,1)
a(3,1)
a(1,3)
a(2,:)
a(:,2)
a(:,:)
a([1:2],[1:2])
a([1,3],[2,3])
a([1,3],:)
```

```
>> a(:,:)

ans =

     1     2     3
     4     5     6
     7     8     9

>> a([1:2],[1:2])

ans =

     1     2
     4     5

>> a([1,3],[2,3])

ans =

     2     3
     8     9

>> a([1,3],:)

ans =

     1     2     3
     7     8     9

>>
```

Figure 1.27
Click image to enlarge, or click here to open



A =

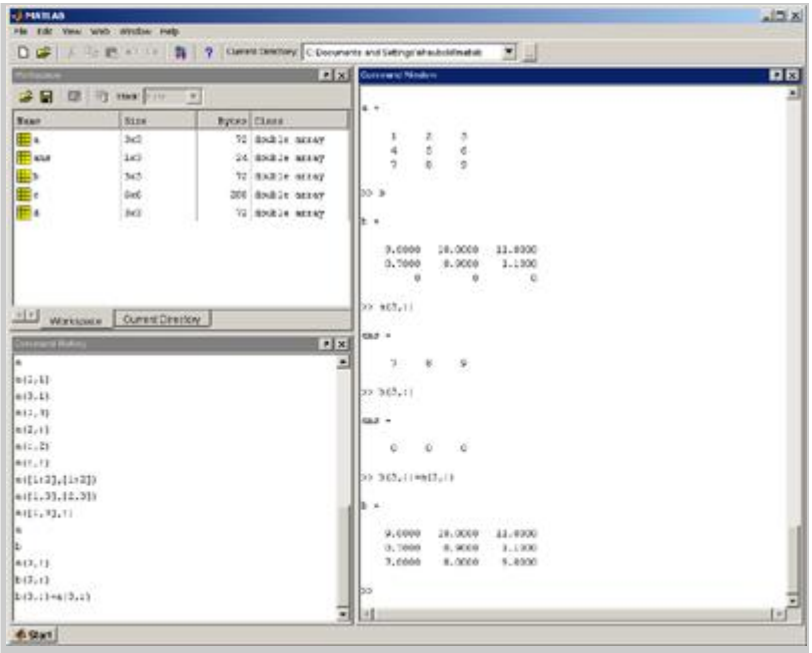
5	1	-9	8	91	7
.9	-1	34	3	19	8
5	0	0	61	81	9
-1	8	91	7	4	2

```
A (2:3,3:5) = 34  3  19
               0  61 81
```

Figure 1.28

Not only can we extract a sub-matrix, but we can also replace portions of a matrix. The same form of indexing is used, as shown in Figure 1.29.

The concept of replacing sub-matrices is pictorially described in Figure 1.30.


MATLAB
File Edit View Web Window Help
Current Directory: C:\Documents and Settings\...\matlab
Workspace
Stack
Workspace
Current Directory
Command History
Command Window
Start

Name	Size	Bytes	Class
a	3x3	72	double array
ans	1x3	24	double array
b	3x3	72	double array
c	6x6	288	double array
d	3x3	72	double array

```
a
a(2,1)
a(3,1)
a(:,3)
a(2,:)
a(:,2)
a(:,:)
a([1:2],[1:2])
a([1,3],[2,3])
a([1,3],:)
a
b
a(3,:)
b(3,:)
b(3,:)=a(3,:)
```

```
a =

     1     2     3
     4     5     6
     7     8     9

>> b

b =

    9.0000   10.0000   11.0000
    0.7000    8.9000    1.1300
         0         0         0

>> a(3,:)

ans =

     7     8     9

>> b(3,:)

ans =

     0     0     0

>> b(3,:)=a(3,:)

b =

    9.0000   10.0000   11.0000
    0.7000    8.9000    1.1300
    7.0000    8.0000    9.0000

>>
```

Figure 1.29
Click image to enlarge, or click here to open

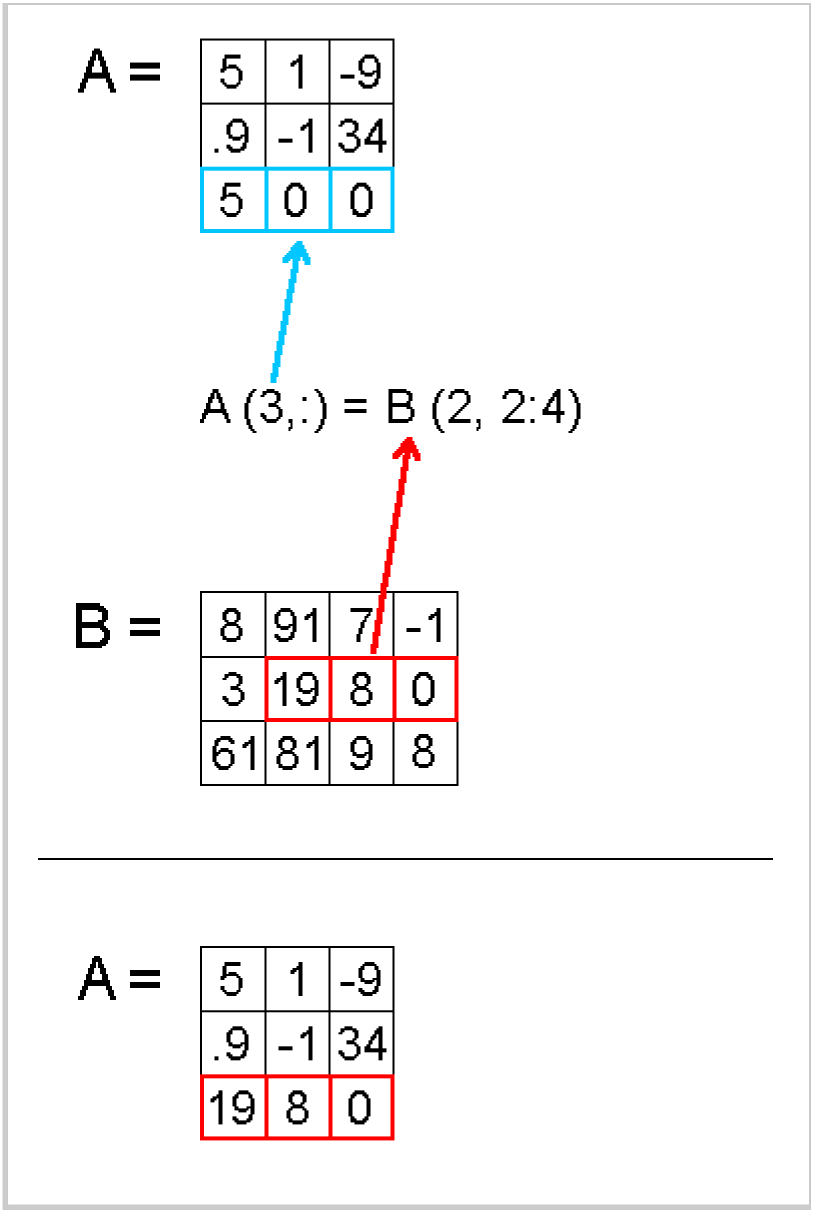
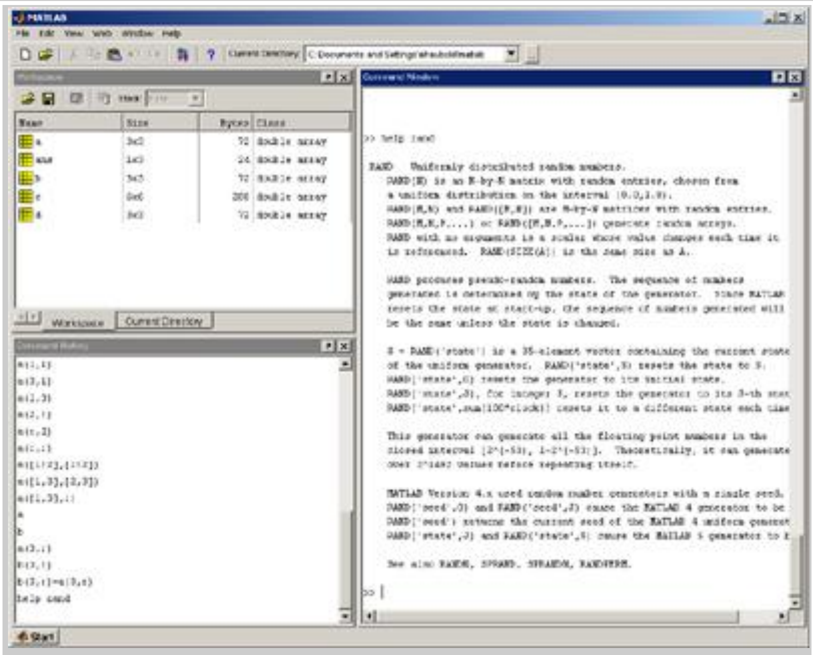


Figure 1.30

To get quick help on any Matlab function, we type `help function` where **function** is the name of the function. `help rand` , for example, gives detailed help on function **rand**.


MATLAB
File Edit View Web Window Help
Current Directory: C:\Documents and Settings\...\matlab
Workspace
Current Directory
Command History
Command Window
Start

Name	Size	Bytes	Class
a	3x3	72	double array
ans	1x3	24	double array
b	3x3	72	double array
c	6x6	288	double array
d	3x3	72	double array

```
>> help rand

 RAND   Uniformly distributed random numbers.
    RAND(N) is an N-by-N matrix with random entries, chosen from
    a uniform distribution on the interval (0.0,1.0).
    RAND(M,N) and RAND([M,N]) are M-by-N matrices with random entries.
    RAND(M,N,P,...) or RAND([M,N,P,...]) generate random arrays.
    RAND with no arguments is a scalar whose value changes each time it
    is referenced.  RAND(SIZE(A)) is the same size as A.

    RAND produces pseudo-random numbers.  The sequence of numbers
    generated is determined by the state of the generator.  Since MATLAB
    resets the state at start-up, the sequence of numbers generated will
    be the same unless the state is changed.

    S = RAND('state') is a 35-element vector containing the current state
    of the uniform generator.  RAND('state',S) resets the state to S.
    RAND('state',0) resets the generator to its initial state.
    RAND('state',J), for integer J, resets the generator to its J-th stat
    RAND('state',sum(100*clock)) resets it to a different state each time

    This generator can generate all the floating point numbers in the
    closed interval [2^(-53), 1-2^(-53)].  Theoretically, it can generate
    over 2^1492 values before repeating itself.

    MATLAB Version 4.x used random number generators with a single seed.
    RAND('seed',0) and RAND('seed',J) cause the MATLAB 4 generator to be
    RAND('seed') returns the current seed of the MATLAB 4 uniform generat
    RAND('state',J) and RAND('state',S) cause the MATLAB 5 generator to b

    See also RANDN, SPRAND, SPRANDN, RANDPERM.

>> |
```

Figure 1.31
Click image to enlarge, or click here to open

To save a collection of expressions, for example for in-class exercises and homework assignments, you will need to use m-files. M-files are no more than simple text files that can be executed from within Matlab to reproduce the statements in the m-file.

To create a new m-file, go to *File -> New -> M-file*, as shown in Figure 1.32.

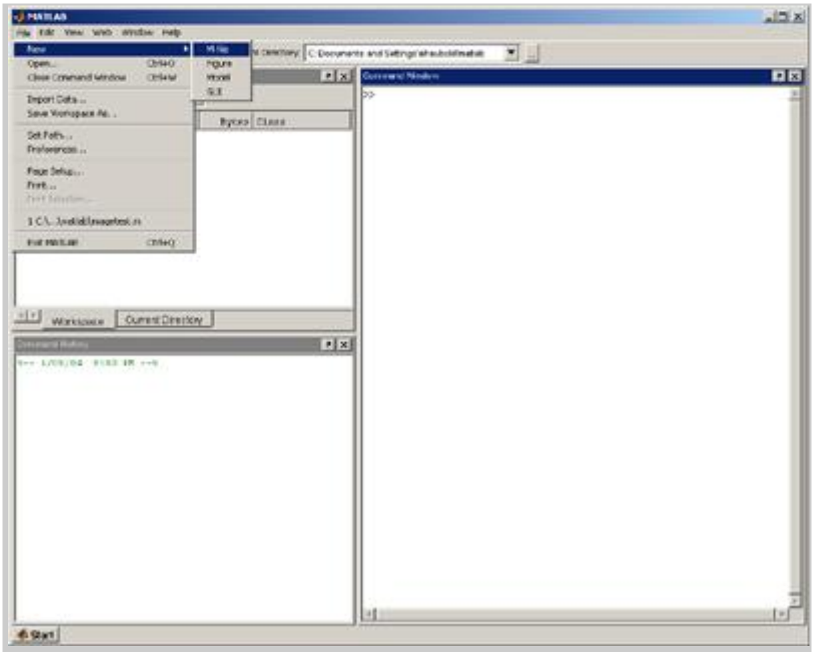
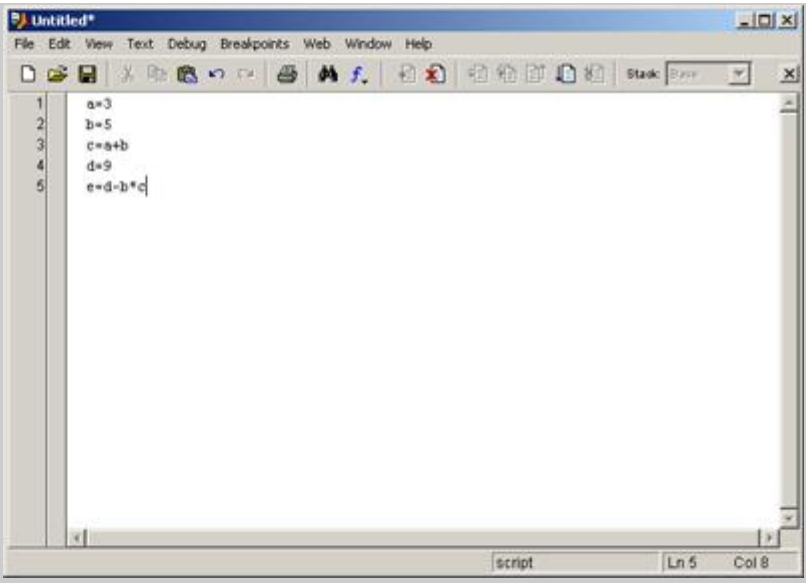


Figure 1.32
Click image to enlarge, or click here to open

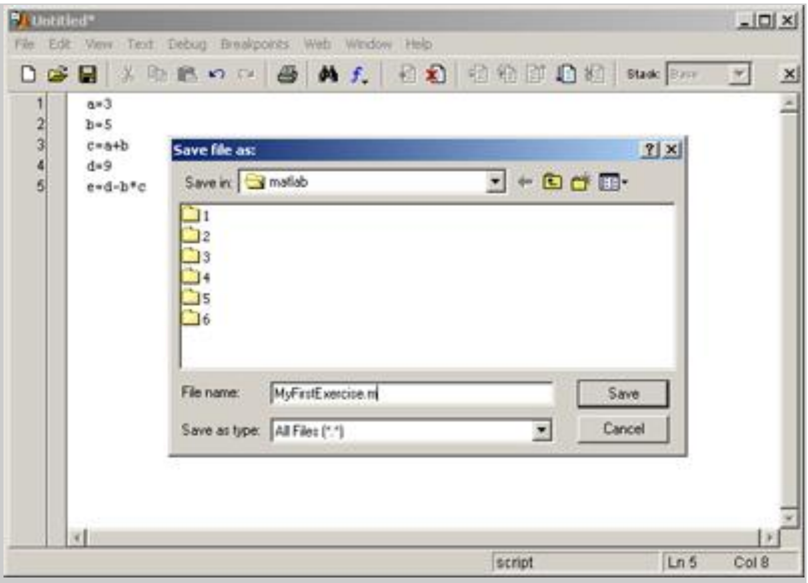
We can now enter a number of statements, one statement per line. In a later lecture, we will see how to create functions using m-files.



```
a=3
b=5
c=a+b
d=9
e=d-b*c
```

Figure 1.33
Click image to enlarge, or click here to open

Save the m-file in the *matlab* directory which you have earlier chosen as the *Current Directory*. Make sure that the extension of this file is **.m**, i.e. make sure that the file ends with **.m**

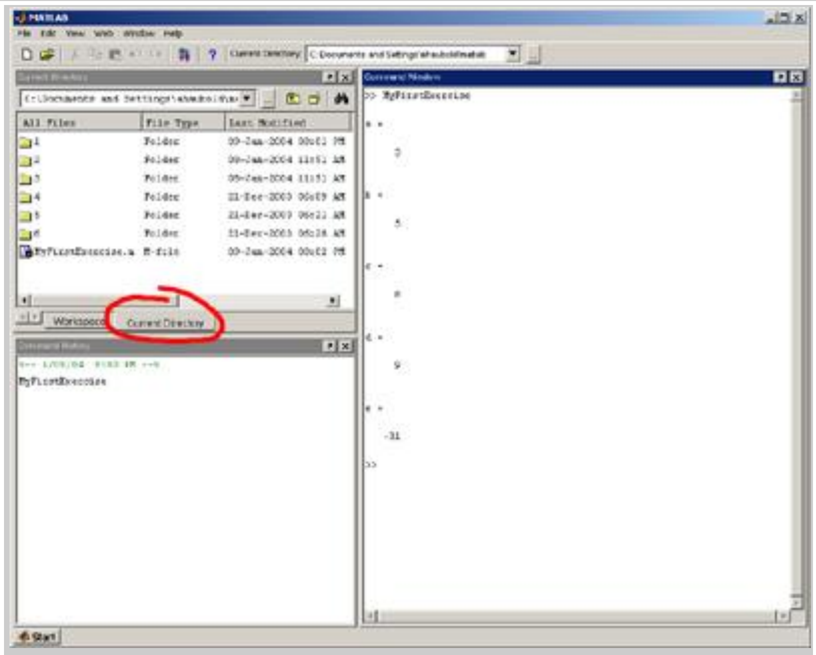


```
a=3
b=5
c=a+b
d=9
e=d-b*c
```

File name: MyFirstExercise.m
Save as type: All Files (*.*)

Figure 1.34
Click image to enlarge, or click here to open

You can now execute the statements in the m-file by typing in the name of the m-file in the *Command Window*. Make sure to omit the **.m** extension.


MATLAB
File Edit View Web Window Help
Current Directory: C:\Documents and Settings\ahaubold\matlab
Current Directory
Command Window
Workspace
Current Directory
Command History
Start

```
C:\Documents and Settings\ahaubold\ma
All Files            File Type   Last Modified
1                    Folder      09-Jan-2004 00:02 PM
2                    Folder      09-Jan-2004 11:51 AM
3                    Folder      09-Jan-2004 11:51 AM
4                    Folder      21-Dec-2003 06:09 AM
5                    Folder      21-Dec-2003 06:21 AM
6                    Folder      21-Dec-2003 06:28 AM
MyFirstExercise.m    M-file      09-Jan-2004 00:02 PM
```

```
-- 1/09/04  9:53 AM --
MyFirstExercise
```

```
>> MyFirstExercise

a =

     3


b =

     5


c =

     8


d =

     9


e =

   -31

>>
```

Figure 1.35
Click image to enlarge, or click here to open

Links

http://202.5.195.17/emust/web/

http://uranchimeg.com/Education/?page_id=1534

http://www.aquaphoenix.com/lectures/matlab1/page3.html