

1. Introduction

1.1. What is MATLAB (Matrix Laboratory)?

MATLAB is an interactive computer language used for technical and scientific computing. It is available for many computer systems including MS windows, Linux and Macintosh. MATLAB integrates computation, visualization and programming from a user friendly (easy to use) environment. The user can solve problems and obtain their results through interactive sessions in a familiar mathematical notation.

MATLAB was first developed in the 1970's for the applications involving matrices, linear algebra and numerical analysis. Throughout, the years the program's capabilities have been greatly extended and today some of the typical uses of MATLAB include

- Maths and computation
- Scientific and Engineering graphics
- Algorithm development
- Modelling and simulation
- Data analysis and visualization
- Application development

Among many others, there are two very important features for using MATLAB. First, the basic data element of MATLAB is an array element that does not require dimensioning. This allows us to solve problems especially those involving matrix and vector formulations in a very short period of time, compared to developing a program to solve the same problem in traditional programming languages such as C, Fortran and Pascal. Second, there is no distinction among real, complex and integer numbers. All numbers are in double precision, which means, MATLAB connects all kinds of numbers continuously and any variable can take any type of number without special declaration in programming. This feature makes programming development faster.

These notes cover MATLAB version 6 for MS windows which was released in the fall of 2000.

1.2. Starting MATLAB and Quitting MATLAB

Starting MATLAB: On MS windows double click on MATLAB icon

Quitting MATLAB: Select **Exit Matlab** from the **File Menu** in the desktop, or type **quit** in the Command Window

2. Matlab Desktop

When you open Matlab program the MATLAB desktop will be displayed as shown in the figure below. The desktop contains separate tools that are used for file operations, variables, applications, visualizations associated with MATLAB. Note that, Figure 1 is the default appearance of MATLAB desktop, which contains three separate

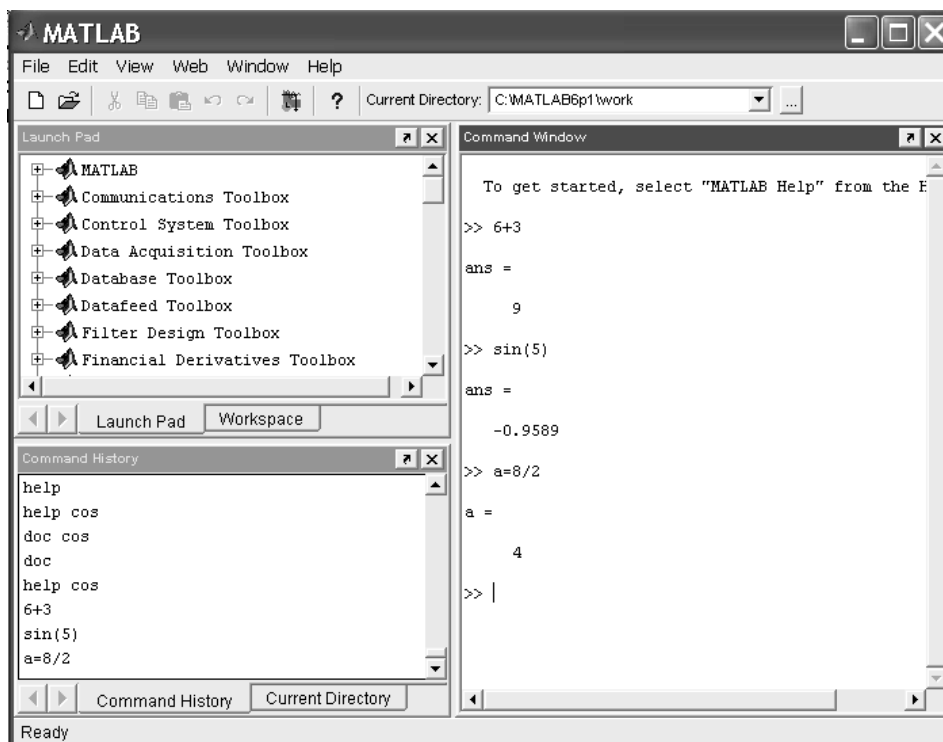


Figure 1: The default Matlab desktop

windows(Command Window, Launch Pad, Command History). The appearance of the desktop can be changed according to your needs. We can resize, open, move, close the windows of desktop. The windows appearing in the desktop can also be moved outside the desktop (undocking).

EXAMPLE 1. To undock (move window outside MATLAB desktop) the Command Window click on the button containing arrow on the upper right corner of Command Window (See Figure 2).

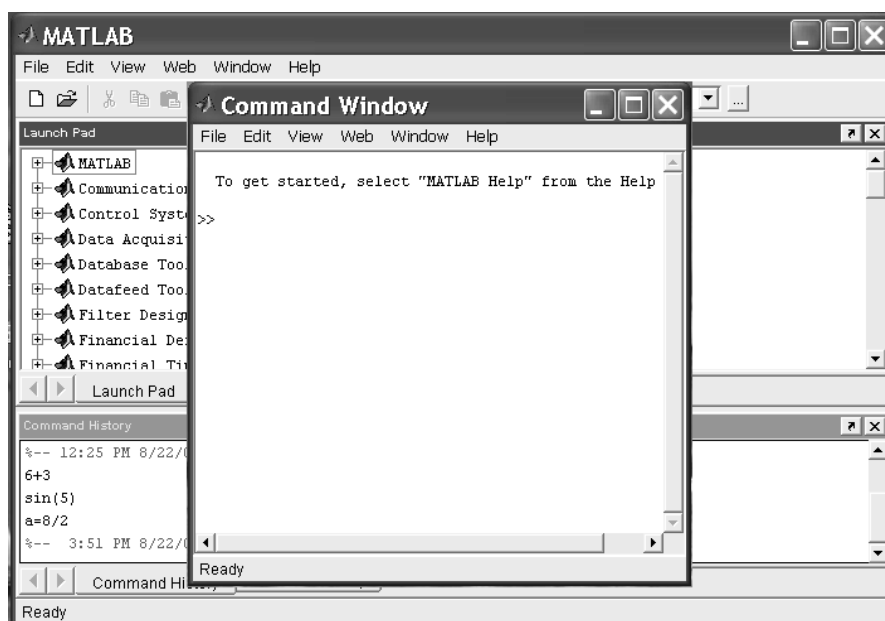


Figure 2: Undocking Command window

If you want to move an undocked window inside the desktop (docking), choose **Dock *name* Window** from the **View** menu where *name* is the name of Window that you are docking.

EXAMPLE 2. To dock the Command Window, click the **Dock Command Window** from the view menu.

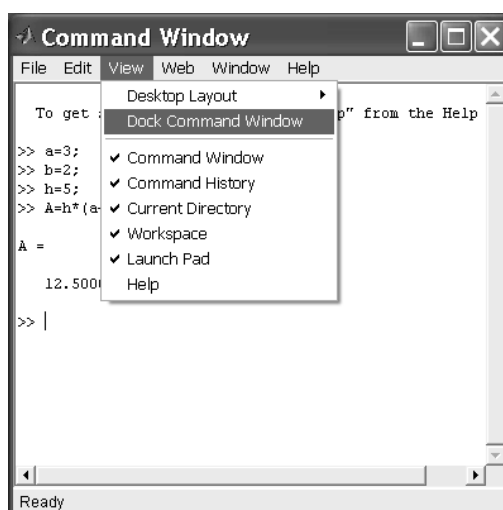


Figure 3: Docking Command Window

The MATLAB desktop tools are listed below

- Command Window
- Command History
- Current Directory Browser
- Workspace Browser
- Launch Pad
- Help Browser
- Array Editor
- Editor/Debugger

The first 6 items can be displayed from the **View** menu as shown in Figure 4. When an item is selected a check

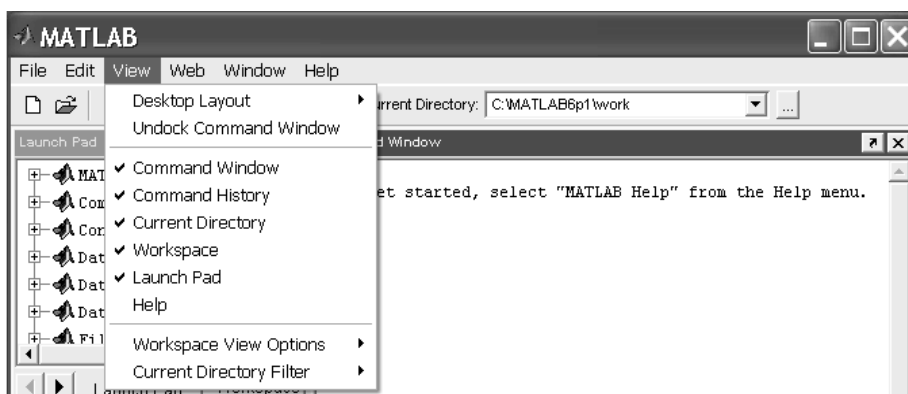


Figure 4: Displaying Desktop tools

appears at the left hand side, which means the item is currently in use. You can also select predefined appearances of MATLAB desktop from the **Desktop Layout** in the **View** menu. Choosing the **Default** Layout from (as seen in Figure below) Desktop returns the appearance shown in Figure 1.

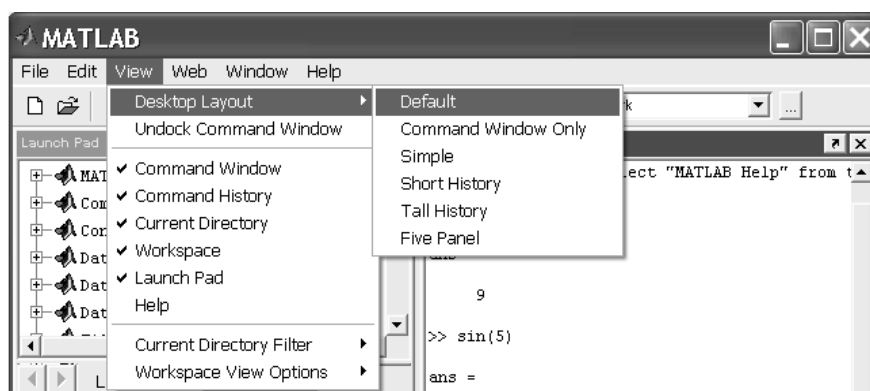


Figure 5: Selecting Desktop appearances

2.1. Command Window

Command Window is used to communicate with MATLAB by typing commands, entering variables, running functions. MATLAB displays the prompt `>>` to indicate that it is ready to take input. To multiply 2 and 6, type `2*6` and press `Enter` (be sure the cursor is at the prompt in the Command window).

```
>>2*6
ans =
    12
```

The answer is stored into a variable called `ans`, which is the variable name for MATLAB to hold the answer to latest calculation. We can use `ans` in further calculations such as

```
>>3+ans
ans =
    15
```

Note that the value stored in `ans` is changed to 15. You can also assign mathematical expressions to variable names that you choose. For example, type `x=5+2*7` (`=` is the assignment operator)

```
>>x=5+2*7
ans =
    19
```

MATLAB stores the scalar value 19 to a variable name `x`. You can use `x` in your further calculation such as

```
>>x=38/x
ans =
     2
```

In MATLAB, variable names are case sensitive as in C. The names `xsum` and `Xsum` correspond to different variables. Variable names have to start with a letter and can contain up to 31 characters. If the name contains more than 31 characters, the characters beyond the 31 st are ignored.

If you make a typing mistake when you are entering commands, you can correct the command by using `Backspace`, `Delete` keys and by using arrow keys `←` and `→`. The previous keystores can be retained with the arrow keys `↑` and `↓`.

2.2. Command History

In a session each line you typed is stored in Command History window. You can view the previously executed lines from the command history and copy, execute, these lines.

EXAMPLE 3. To copy a previously executed line, first, select the line from the Command History and right click on it (see Figure 3). A pop up menu opens. From that menu select copy. Next, click on the Command window and paste it into the desired location.

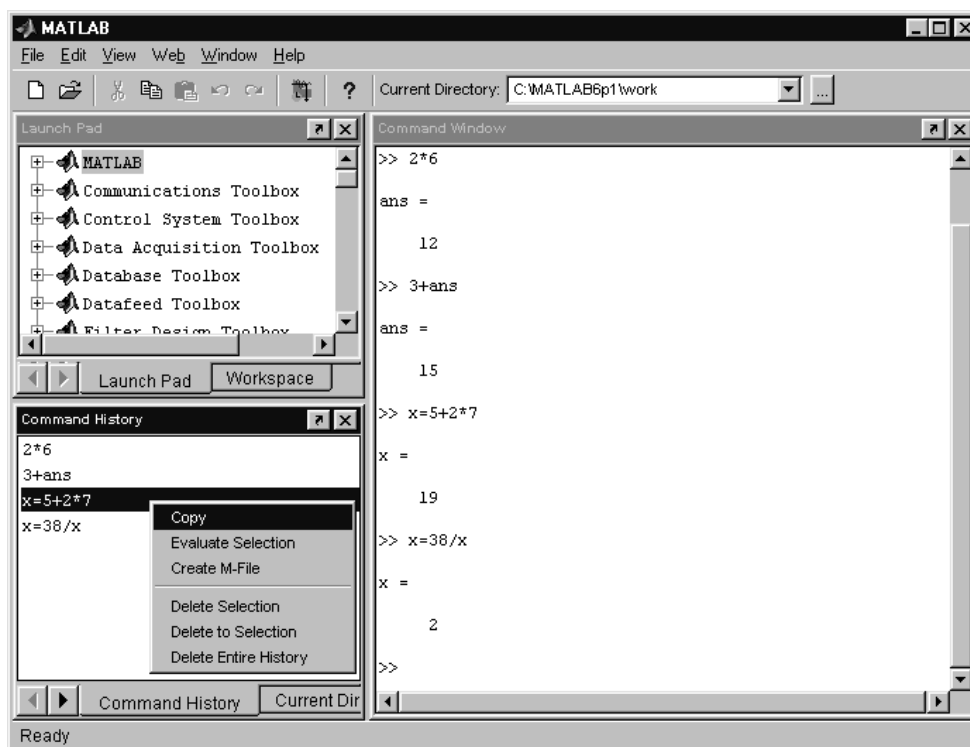


Figure 6: Copying a previously executed line from Command History

If you double click on a line in the Command History, it will automatically be executed in the Command Window.

Scalar arithmetic operations. A scalar is a single number. A scalar variable is variable that contains a single number. The arithmetic operations in mathematical expressions that MATLAB performs on scalars, are given in the table below. Mathematical expressions are always evaluated from left to right. You can use paranthesis in the expressions.

Mathematical Operation	MATLAB Symbol	MATLAB Form
Exponentiation a^b	\wedge	$a\wedge b$
Multiplication ab	$*$	$a*b$
Right division $a/b = \frac{a}{b}$	$/$	a/b
Left division $a\backslash b = \frac{b}{a}$	\backslash	$a\backslash b$
Addition $a + b$	$+$	$a+b$
Subtraction $a - b$	$-$	$a-b$

The next table shows the order of precedence of mathematical expressions from highest to lowest

Paranthesis beginning from the innermost pair	
^	:Exponentiation
*, /	:Multiplication, division
+,-	:Addition, subtraction

EXAMPLE 4. Evaluate $\frac{5 \times 6}{2} + 17$

```
>>5*6/2 + 17
```

```
ans =
```

```
32
```

EXAMPLE 5. Evaluate $8^2 \times 7 - \frac{3 \times 11 + 7}{6 - 1}$

```
>>8^2*7 - (3*11 + 7)/(6 - 1)
```

```
ans =
```

```
440
```

EXAMPLE 6. Evaluate $27^{1/3} \times 2 - 81^{0.5} + 3(17 + 18 + 19)$

```
>>27^(1/3)*2 - 81^0.5 + 3*(17+18+19)
```

```
ans =
```

```
159
```

EXAMPLE 7. Evaluate $\frac{0.5((2^3+1)^2+15)^2}{8}$

```
>>0.5*( (2^3+1)^2 + 15)^2/8
```

```
ans =
```

```
576
```

Note that, in writing the expressions for the above examples, we use spaces to improve readability. MATLAB ignores the spaces before and after the assignment operator '=' when making its calculations. The expressions

```
>>x=8/2-3*(7+1)
```

```
>>x = 8/2 - 3*(7 + 1)
```

give the same result, however reading the second expression is much more easier than the first.

Assignment operator. In MATLAB, sign = is used as an assignment operator. Typing $x = 12$ assigns the value 12 to the variable name x. MATLAB assigns the value which is calculated on the right hand side of operator = to a variable name on the left hand side. The opposite is not valid, that is, writing $12 = x$ produces an error in MATLAB. We can use expressions like $x = x + 12$ provided that x has been assigned a value before writing this expression. There must be only one variable on the left side of the assignment operator. Writing $x + 12 = x$ is invalid in MATLAB.

EXAMPLE 8. Calculate the area of a trapezoid which is given by the formula $A = h\frac{(a+b)}{2}$, where a and b are the side lengths and h is the base length of the trapezoid. In our problem we choose $a = 3$, $b = 2$ and $h = 5$.

```
>> a = 3;
>> b = 2;
>> h = 5;
>> A = h*(a+b)/2
A =
    12.5000
```

Notice that, we use semicolon ';' in the end of the first three expressions. In MATLAB, semicolon is used to suppress printing on the screen. In Example 8, the values 3, 2 and 5 are assigned to the variables a , b and h respectively, but the results are not printed on the screen. We can write several commands on the same line if we separate them with comma ',' and semicolon ';'. The previous example can also be entered in the form

```
>> a = 3; b = 2; h = 5; A = h*(a+b)/2
A =
    12.5000
```

If we use commas instead of semicolons the output will be

```
>> a=3, b=2, h=5, A=h*(a+b)/2
a =
     3
b =
     2
h =
     5
A =
    12.5000
```

2.3. Workspace browser

In MATLAB, the term workspace refers to the names and values of variables that are stored in memory during the current working session. The information about each variable in the workspace can be obtained from the Workspace Browser. To open the Workspace Browser, select **View** menu in the desktop and choose **Workspace**. When the Workspace Browser is opened, you see the names, sizes and classes of each variable that MATLAB uses in the current session. The next figure shows the appearance of Workspace Browser, assuming that the workspace contains the variables for the previous example.

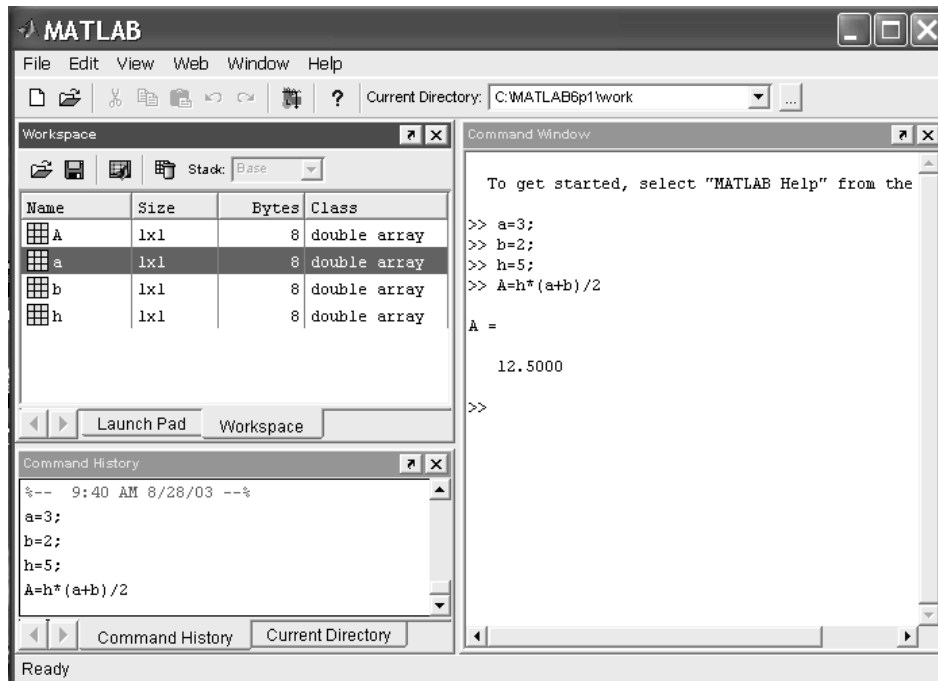


Figure 7: Workspace browser

You can change the values of the variables or delete them from the memory using Workspace Browser. We will cover the use of Workspace browser in more detail in the following chapters.

Managing the working Session in MATLAB. MATLAB uses some commands for managing the working session. In the following table, we introduce some of these commands and give examples on how to use them.

Command	Description
<code>who</code>	:Lists variables that are currently stored in the memory
<code>whos</code>	:Lists variables that are currently stored in the memory by giving their sizes and classes
<code>clear</code>	:Removes all the variables that are stored in the memory
<code>clear var1 var2 var3</code>	:Removes the variables <code>var1</code> <code>var2</code> and <code>var3</code> from the memory
<code>clc</code>	:Clears the command window (the variables are still in the memory)

EXAMPLE 9. List the current variables for Example 8 using `who` command

```
>> who
Your variables are:
A a b h
```

EXAMPLE 10. List the current variables and get their details for Example 8 using `whos` command

```
>> whos
  Name      Size      Bytes  Class

  A         1x1         8  double array
  a         1x1         8  double array
  b         1x1         8  double array
  h         1x1         8  double array
```

Grand total is 4 elements using 32 bytes

EXAMPLE 11. Remove variable `a` from the memory and then list the current variables

```
>> clear a
>> who
Your variables are:
A b h
```

2.4. Current Directory browser

While working with MATLAB, we need to save, open, search and view files. Therefore, it is important to know the files you use with MATLAB. MATLAB file operations use *current directory* and *search path* as reference points. Any file you must run must be either in the *current directory* or on the *search path*. *Current directory* can be seen from Current Directory field on top of the desktop or on top of Current Directory Browser as shown in Figure 8. MATLAB uses Current Directory Browser for file operations. When you open Current directory browser, you see a list of files that exist in the current directory. Suppose you want to view files in the directory named **Mywork**. First, you have to change the current directory to **Mywork**. This can be done either typing `cd C:\Mywork` in the Command Window or by using browser on the right hand side of Current Directory field. You can also display the current directory by typing `pwd` from the command window. The table below summarizes some directory commands.

<code>pwd</code>	Displays the current directory
<code>dir</code>	Lists all files in the current directory
<code>dir dirname</code>	Lists all files in the directory <code>dirname</code>
<code>cd dirname</code>	Changes the current directory to <code>dirname</code>

The search path is a set of directories that MATLAB uses to find MATLAB related files. The use of the search path and its related commands will be covered in later chapters.

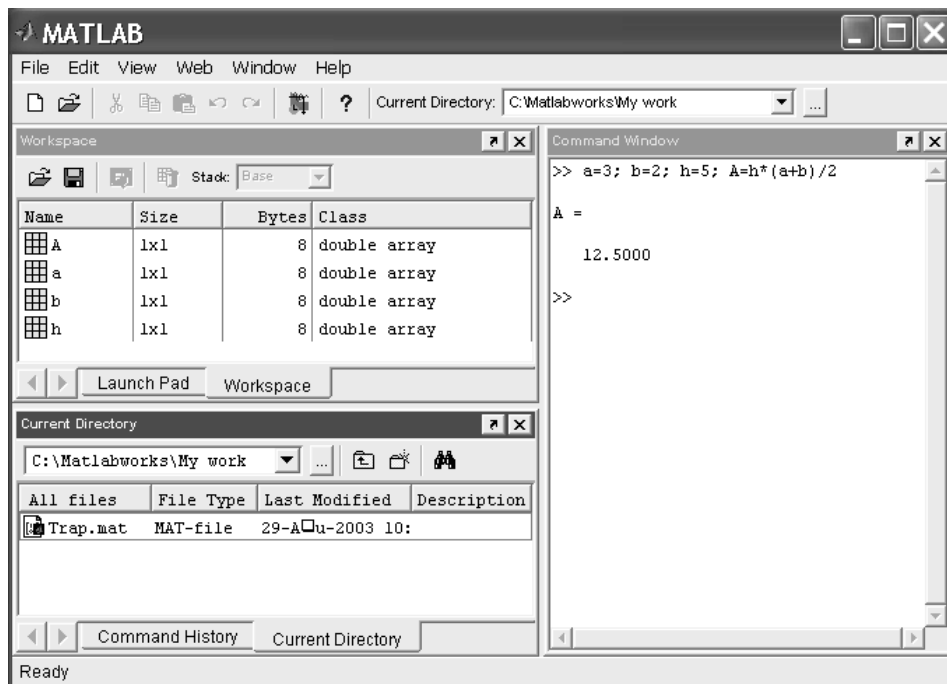


Figure 8: Current Directory Browser

2.5. Getting help from MATLAB

One of the most unique features of MATLAB is its help system. There are two ways to get help from MATLAB. The first one is to use help commands from the Command Window and the other one is to use Help Browser. If you want to get a concise explanation of command, type `help commandname` in the Command Window.

EXAMPLE 12. *Getting help about the cos (cosines) function from MATLAB using help command*

```
>> help cos
```

```
COS    Cosine.
       COS(X) is the cosine of the elements of X.
```

```
Overloaded methods
       help sym/cos.m
```

More detailed information about a command or subject can be obtained from the Help Browser. To open Help Browser, select **Matlab Help** from the **Help** menu on the desktop. The help browser contains 2 windows: the Display Pane and Help Navigator. There are 4 Tabs on the Help Navigator:

- Contents Tab is used to list the titles and table of contents for all MATLAB documentation
- Index Tab is used to find keywords from all MATLAB documentation
- Search Tab is used to find all MATLAB documents containing a specified phrase
- Favorites Tab is used to view a list of documents you previously designated as favorites

EXAMPLE 13. To get help about cosine function from MATLAB using help menu, first open the **Help Browser**, and click **Index** tab in the Help Navigator window. Next, type cosine into the search index box. As you type, the index displays the matching items. Select **cosine** by clicking on it. As the item is selected, a detailed documentation about that item is displayed in the display pane. (See the figure below)

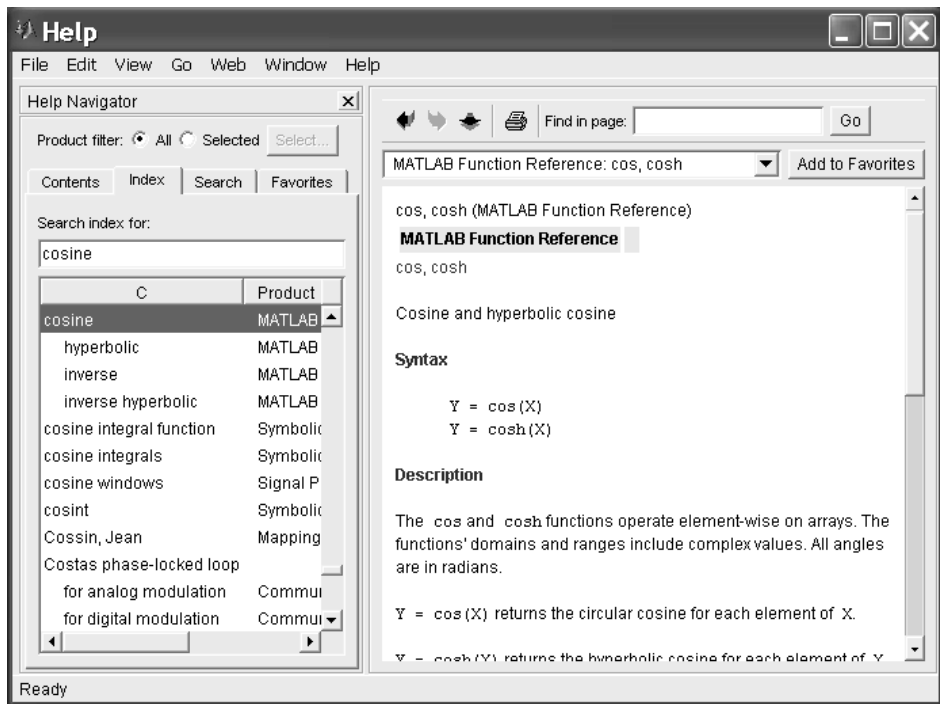


Figure 9: Help Browser

The documentation about a command can also be displayed by typing `doc commandname` in the command window. If you type

```
>> doc cos
```

the Help Browser opens and displays the same documentation as in the case of the previous example. Notice that, in order to use `help` command you have to know the exact spelling of the command. If you type

```
>> help cosine
```

MATLAB gives you a warning message as below

```
>> help cosine
```

```
cosine.m not found.
```

because there is no function definition spelled as cosine in MATLAB.

Another important function is `lookfor` function. This function allows us to search functions based on a keyword. When you type `lookfor keyword` in the command window, MATLAB displays all functions that contain the keyword.

EXAMPLE 14. Search all functions that contain the keyword "tangent" using `lookfor` function

```
>> lookfor tangent
ACOT   Inverse cotangent.
ACOTH  Inverse hyperbolic cotangent.
ATAN   Inverse tangent.
ATAN2  Four quadrant inverse tangent.
ATANH  Inverse hyperbolic tangent.
COT    Cotangent.
COTH   Hyperbolic cotangent.
TAN    Tangent.
TANH   Hyperbolic tangent.
DTANSIG Hyperbolic tangent sigmoid transfer derivative function.
TANSIG Hyperbolic tangent sigmoid transfer function.
ACOT   Symbolic inverse cotangent.
ACOTH  Symbolic inverse hyperbolic cotangent.
ATAN   Symbolic inverse tangent.
ATANH  Symbolic inverse hyperbolic tangent.
COT    Symbolic cotangent.
COTH   Symbolic hyperbolic cotangent.
TAN    Symbolic tangent function.
TANH   Symbolic hyperbolic tangent.
BLKATAN This block defines an output angle that is the arctangent of two inputs.
BLKCOSATAN This block defines the cosine of an angle whose tangent is u1/u2.
BLKSINATAN This block defines the sine of an angle whose tangent is u1/u2.
```

An efficient way, for getting help from MATLAB can be performed in two steps

1. Use `lookfor` function to list all functions that contain a certain keyword.
2. From that list, select the function name that you are looking for and then use `help` or `doc` command to get detailed information about that function or command name.

2.6. Menus and Toolbar

As we discussed before MATLAB has different type of windows such as *Command window*, *Command History*, *Launch Pad*, *Workspace Browser* and *Current Directory browser*. On the top of Desktop there is a menu bar and a toolbar which contains icons and current directory field. There are also other windows in MATLAB that appear when you plot graphs or when you write your MATLAB programs. Each window type has a menu bar and the contents of these menus can be changed depending on which window you are using.

Below, we summarize the use of desktop menus when the *Command Window* is active. The desktop contains 6 different menus if you activate the Command Window. These are

- **File**
- **Edit**

- View
- Web
- Window
- Help

menus. You can activate one of these menus by clicking on it. After clicking, MATLAB brings a list of items. You can select one of the items by clicking on it and MATLAB performs the operation. When you examine the items that appear when a menu is opened, you see that, some of the item names are followed by three dots "...". This means that, if you select that item a *submenu* or another window will be opened.

File menu in MATLAB 6. The file menu is used to perform file operations in MATLAB. Each item and its use are listed in the table below.

New	Opens a dialog box that allows you to create a new program file, a new Figure or Model File.
Open	Opens a dialog box that allows you to select a file for editing.
Close Command Window	Closes the command window.
Import Data	Starts the Import Wizard that allows you to import data easily.
Save Workspace As...	Opens a dialog box that enables you to save a file.
Set Path...	Opens a dialog box that enables you to set MATLAB search path.
Preferences...	Opens a dialog box that enables you to set preferences for such items as fonts, colors, tab spacing...
Print...	Opens a dialog box that enables you to print all of the Command Window.
Print Selection...	Opens a dialog box that enables you to print selected portions of the Command Window.
File List	Contains a list of previously used files, in the order of most recently used.
Exit MATLAB	Closes MATLAB.

Edit menu in MATLAB 6. The items and their use in the edit menu are given in the table below

Undo	Reverses the previous editing action.
Redo	Reverses the previous Undo operation.
Cut	Removes the selected text and stores it for pasting later.
Copy	Copies the selected text for pasting later, without removing it.
Paste	Inserts any text on the clipboard at the current location of the cursor.
Paste Special...	Inserts the contents of the clipboard into the workspace as one or more variables.
Select All	Highlights all text in the Command Window.
Delete	Clears the highlighted text from the Command Window.
Clear Command Window	Removes all text from the Command Window.
Clear Command History	Removes all text from the Command History window.
Clear Workspace	Removes the values of all variables from the workspace.

View menu in MATLAB 6. View menu is used to control the appearance of MATLAB. The table below list the items and summarizes their use.

Desktop Layout	used to select the predefined appearances of MATLAB, where there are six options: Default, Command Window Only, Simple, Short History, Tall History and Five Panel.
Undock Command Window	Undocks Command Window (See Example 1).
Command Window	Opens or closes Command Window
Command History	Opens or closes Command History
Current Directory	Opens or closes Current Directory Browser
Workspace	Opens or closes Workspace Browser
Launch Pad	Opens or closes Launch Pad
Help	Opens or closes Help Browser
Workspace View Options	Arranges the appearance of Workspace Browser

Note that when you select a window name from the **View** menu, a check appears in front of the name of window.

Web menu in MATLAB 6. Web menu is used to access the MathWorks (the owner of MATLAB) support site. You can get information about the other products of MathWorks and also you can find technical support and updates about MATLAB.

Window menu in MATLAB 6. The Window menu has one or more items depending on what you have done in your session. Click the name of the window that appears on the menu to open it.

Help menu in MATLAB 6. The Help menu is explained in the previous sections.

The use of these menus will be explained in a more detailed form by giving some example in the following sections.

2.7. Saving and Loading your Workspace variables

Recall that, the term workspace refers to the names and values of variables that are stored in memory during the current working session. You can save your workspace variables in files called *mat*. files (The extension of these files are *.mat*) . *Mat* files are binary files that can only be read by MATLAB and they contain the names, types, sizes and values of your workspace variables. Saving workspace variables allows you to continue your working session at a later time. There are two ways for saving workspace variables

1. Using **save** command from the Command Window

- * If you type **save** from the Command Window, MATLAB saves your workspace variables in a file called **matlab.mat**. You can also specify the name of file by typing **save filename** where **filename** is the name of the file. It should be recalled that MATLAB saves these files in *current directory* which can be seen from the *Current Directory Field*.

EXAMPLE 15. *Save your workspace variables into a file named "Myfile"*

```
>> save Myfile
```

2. Selecting the item **Save Workspace As...** from the **File** menu

- * When you select **Save Workspace As...** item, a window appears from which you can specify the name and the location of the file that is to be saved.

When needed, an existing *.mat* file can be opened and the workspace variables of that *.mat* file can be loaded into your session. Before loading your workspace variables, be sure that the location of current directory is correct. There are three ways of loading workspace variables.

1. Using `load` command from the Command Window

- * If you type `load` from the Command Window, MATLAB loads your workspace variables from the file `matlab.mat`. Recall that, `matlab.mat` is the default filename that MATLAB gives when you use `save` command without specifying filename. Typing `load filename` loads the workspace variables in the file `filename.mat`.

EXAMPLE 16. *Load the workspace variables in the file named "Myfile"*

```
>> load Myfile
```

2. Selecting the **Open** item from the file menu

- * Click **Open**, and go to the location of the *mat* file you want to load. Select the *mat* file you want to open and press the Open button or double click on the file.

3. Using Current Directory Browser

- * From the list in the Current Directory Browser, if you double click on the a *mat* file, MATLAB loads the workspace variables of that file to your working session.

2.8. Recording your session

In your working session, you can record all your activities in a file. Typing `diary on` from the Command Window, starts recording your keystrokes and most of the screen output to a file named `diary`. It should be noted that, this record will not include your graphs. Typing `diary off` terminates recording your activities.

In MATLAB, this kind of recording is called keeping *diary*. The status of the *diary* can be `on` or `off` depending on whether MATLAB is recording your activities or not. When we type `diary on`, the status of *diary* becomes `on` and recording starts. On the other hand, when we type `diary off`, the status of the *diary* becomes `off` and MATLAB terminates recording. If you type `diary off` to stop recording and later type `diary on`, MATLAB appends the remainder of the session to the end of file.

You can change the status of the *diary* by typing `diary` itself. This means, if the status of the *diary* is `on`, typing `diary` makes the status of the *diary* `off`, or if the status of the *diary* is `off`, typing `diary` makes the status of *diary* `on`.

You can also keep your diary in a file with a filename you choose by typing `diary filename`.

To see the status of the *diary*, type `get(0,'diary')` and MATLAB will indicate the diary status by returning either `on` or `off`.

An Important Note

You can not view your current recordings while the diary status is `on`. In order to view these recordings first you must change the diary status to `off` state.

EXAMPLE 17. *Getting the diary status*

```
>> get(0,'diary')
ans =
off
```

To see the diary filename type `get(0,'diaryfile')`, and MATLAB returns the name of the diary filename

EXAMPLE 18. *Getting the diary filename*

```
>> get(0,'diaryfile')
ans =
diary
```

Note that, the *diary* files are ASCII text files which means that, with any type of word processor, you can read, edit and print these files. At this point, we introduce MATLAB's text editor which is called *Editor/Debugger*. *Editor/Debugger* is a text editor which is used to write, edit and debug MATLAB programs. It can also be used to write and edit text files such as *diary* files. If you want to open a *diary* file from MATLAB, you have two options

- Select **Open** from the **File** menu and go to the location of the *diary* file. Change **All MATLAB files** to **All Files (*.*)** in the **File Type** field. Now you can see all the files that are stored in the directory. Select the *diary* file and press **Open**.
- Double click the name of the *diary* file from the list in the Current Directory Browser.

EXAMPLE 19. *Lets assume we record the activities shown below in a file named diary and we want to view this file using MATLAB Editor/Debugger*

```
>> diary
>> a=1

a =

    1

>> b=2

b =

    2

>> c=a+b

c =

    3

>> diary off
>>
```

When you use one of the procedures explained above, the *Editor/Debugger Window* will be opened as shown below. By using this window, you can edit the diary file and perform file operations such as printing.

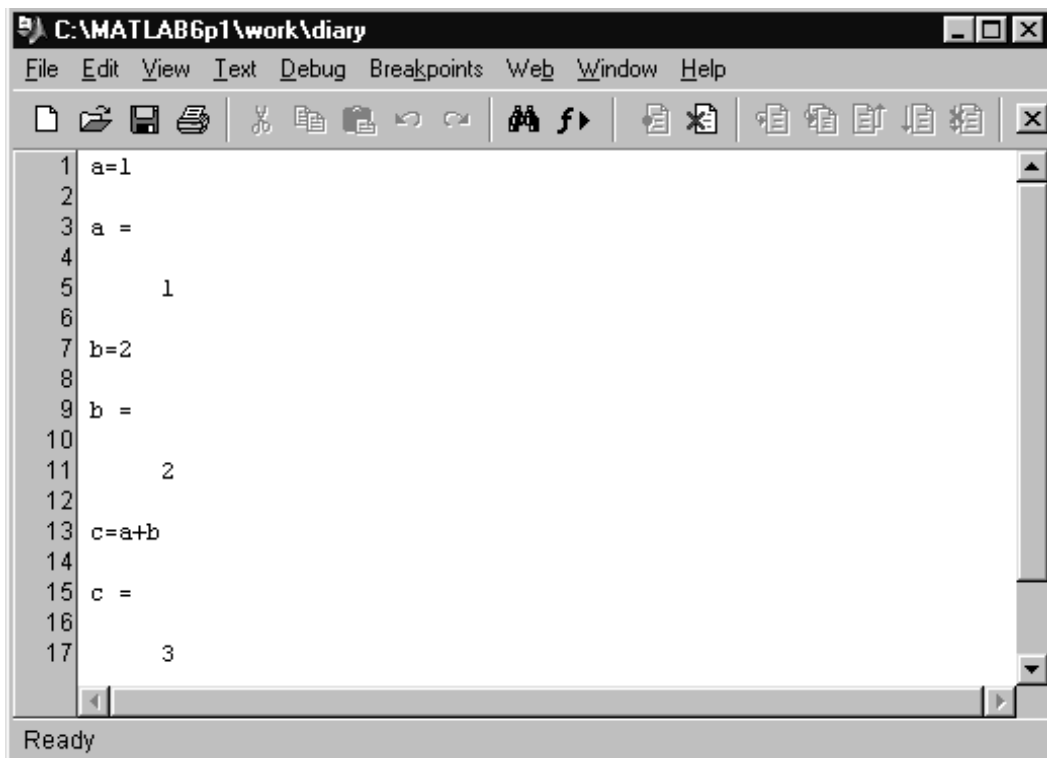


Figure 10:

3. Special variable names and predefined constants in Matlab

There are some special variable names and predefined constants in MATLAB to represent special numbers and cases. We list some of these by giving simple examples.

- **ans**: is a variable name containing the most recent answer. Using **ans**, you can carry the latest result in your further calculations.

EXAMPLE 20. First calculate $\frac{12.63 \times 72.74}{20}$ then divide the result by 0.456

```
>> 12.63*72.74/20.0
```

```
ans =
```

```
45.9353
```

```
>> ans/0.456
```

```
ans =
```

```
100.7353
```

```
>>
```

- **Inf**: is a variable name which stands for ∞ (infinity). If you type $2/0$, MATLAB generates the answer **Inf**. Also MATLAB uses **Inf** for the numbers so large that it can't represent.

EXAMPLE 21. Try typing $\frac{2}{0}$

```
>> 2/0
```

```
Warning: Divide by zero.
```

```
ans =
```

```
Inf
```

```
>>
```

Notice that, in this example MATLAB first gives a division by zero warning than generates the answer **Inf**.

EXAMPLE 22. Try calculating 5^{987}

```
>> 5^987
```

```
ans =
```

```
Inf
```

```
>>
```

Although there is an answer for this calculation, MATLAB produces **Inf**. This is because the answer exceeds the largest number that MATLAB represents.

- **NaN**: indicates an undefined numerical result such as $0/0$, ∞/∞ and $0 \times \infty$.

EXAMPLE 23. Type $\frac{0}{0}$ and ∞/∞

```
>> 0/0
```

```
Warning: Divide by zero.
```

```
ans =
```

```
NaN
```

```
>> Inf/Inf
```

```
ans =
```

```
NaN
```

```
>>
```

- **eps**: is the smallest number in which, when added to 1 by MATLAB, creates a number greater than 1. That is, **eps** represents the accuracy of computations.

- `pi`: stands for the number π .

EXAMPLE 24. Type `pi`

```
>> pi
```

```
ans =
```

```
3.1416
```

- `i` or `j`: denote the imaginary number where $i = j = \sqrt{-1}$

The variables symbols explained above are the MATLAB's predefined constants. In your calculations and programs, it is better not using them as your variable names.

4. Elementary mathematical functions

MATLAB has a wide collection of build-in Mathematical functions. These mathematical functions can be classified into 3 categories

- Trigonometric functions
- Other Elementary functions
- Functions that do chores

While using these functions, you can use complex arguments as well as real arguments. This is a great advantage over the other programming languages from the fact that, in languages like C and Pascal, the programmer must code or use functions seperately for real and complex arguments. Another distinction of MATLAB's mathematical functions is; these functions also work for vector and matrix arguments.

In the tables shown below, we list some of MATLAB's elementary functions that fall into first two categories shown above. Functions that do chores will be explained later.

Trigonometric Functions	Explanation
<code>sin(x)</code>	is the sine of the elements of x .
<code>cos(x)</code>	is the cosine of the elements of x .
<code>tan(x)</code>	is the tangent of the elements of x
<code>asin(x)</code>	is the arcsine of the elements of x . Complex results are obtained if $\text{abs}(x) > 1.0$ for some element
<code>acos(x)</code>	is the arccosine of the elements of x . Complex results are obtained if $\text{abs}(x) > 1.0$ for some element
<code>atan(x)</code>	is the arctangent of the elements of x . $-\pi/2 \geq \text{atan}(x) \geq \pi/2$
<code>atan2(y, x)</code>	Four quadrant inverse tangent. <code>atan2(y, x)</code> is the four quadrant arctangent of the real parts of the elements of x and y . $-\pi \leq \text{atan2}(y, x) \leq \pi$
<code>sinh(x)</code>	is the hyperbolic sine of the elements of x
<code>cosh(x)</code>	is the hyperbolic cosine of the elements of x
<code>tanh(x)</code>	is the hyperbolic tangent of the elements of x
<code>asinh(x)</code>	is the inverse hyperbolic sine of the elements of x
<code>acosh(x)</code>	is the inverse hyperbolic cosine of the elements of x
<code>atanh(x)</code>	is the inverse hyperbolic tangent of the elements of x

Other Elementary Functions	Explanation
<code>abs(x)</code>	is the absolute value of the elements of x . When x is complex, $\text{abs}(x)$ is the complex modulus (magnitude) of the elements of x .
<code>angle(x)</code>	returns the phase angles, in radians, of a matrix with complex elements If $x = \text{real}$, $\text{angle} = 0$; if $x = \sqrt{-1}$ $\text{angle} = \pi/2$
<code>sqrt(x)</code>	is the square root of the elements of x . Complex results are produced if x is not positive.
<code>real(x)</code>	is the real part of x
<code>imag(x)</code>	is the imaginary part of x
<code>conj(x)</code>	is the complex conjugate of x . For a complex x , $\text{conj}(x) = \text{real}(x) - i*\text{imag}(x)$
<code>round(x)</code>	rounds the elements of x to the nearest integers
<code>fix(x)</code>	rounds the elements of x to the nearest integers towards zero
<code>floor(x)</code>	rounds the elements of x to the nearest integers towards minus infinity ($-\infty$)
<code>ceil(x)</code>	rounds the elements of x to the nearest integers towards infinity (∞)
<code>sign(x)</code>	For each element of x , $\text{sign}(x)$ returns 1 if $x < 0$; returns -1 if $x > 0$ For the nonzero elements of complex x , $\text{sign}(x) = x ./ \text{abs}(x)$
<code>mod(x,y)</code>	Remainder upon division: $x - y*\text{fix}(x/y)$
<code>rem(x,y)</code>	Remainder upon division: $x - y*\text{fix}(x/y)$. Different from <code>mod</code> if $y \leq 0$
<code>exp(x,y)</code>	is the exponential of the elements of x , e^x . For complex $z = x + i * y$, $\text{exp}(z) = \text{exp}(x) * (\cos(y) + i*\sin(y))$
<code>log(x,y)</code>	is the natural logarithm of the elements of x , $\ln(x)$. Complex results are produced if x is not positive
<code>log10(x,y)</code>	is the base 10 logarithm of the elements of x . Complex results are produced if x is not positive
<code>factor(x,y)</code>	factorize x into prime numbers
<code>isprime(x)</code>	1 if x is a prime number, 0 if not
<code>factorial(x)</code>	$x!$

IMPORTANT NOTE

All functions with arguments that are angles expect the angles to be expressed in radians

5. Formatting the output of your calculations

MATLAB gives the output of your computations with 4 decimals by default. You can change the format of the output by using `format` command. Remember that all computations in MATLAB are done by using double precision. Using `format` command do not change the type of your variables, it only controls how the numbers appear on the screen. For example, when you compute `sin(2.5)`, MATLAB displays the result in 4 decimal places as shown below;

```
>> sin(2.5)
ans =
  0.5985
```

Although MATLAB calculates the result up to 15 decimal places, it only displays 4 decimal places. This is because the format of the screen display for numbers is set to 4 decimal places by default. If you type `format long` and perform the same computation again,

```
>> format long
>> sin(2.5)

ans =
```

```
  0.59847214410396
```

you notice that this time the result is displayed in 15 decimal places. There are various format options (types) for displaying numbers in MATLAB. These options are activated by `format` command and some of these are listed in the table below;

Format types	Explanation
<code>format</code>	Same as <code>format short</code>
<code>format short</code>	Displays the number with 4 decimal places (5 digits)
<code>format long</code>	Displays the number with 15 decimal places (16 digits)
<code>format short e</code>	Displays the number with 4 decimal places (5 digits) plus exponent
<code>format long e</code>	Displays the number with 15 decimal places (16 digits) plus exponent

It should be noted that, when you change the display format to a certain type, all the results of your following calculations will be displayed at that type. There is one exception to this rule: if the result of your computation is an integer, then MATLAB does not display decimals.

EXAMPLE 25. Using all format types given in the table above, calculate $e^{15.13}$

```
>> format short
>> exp(15.13)

ans =

  3.7228e+006

>> format long
>> exp(15.13)
```

```
ans =
```

```
3.722849769352520e+006
```

```
>> format short e
```

```
>> exp(15.13)
```

```
ans =
```

```
3.7228e+006
```

```
>> format long e
```

```
>> exp(15.13)
```

```
ans =
```

```
3.722849769352520e+006
```

There are also other formatting types which can be used for special purposes. The detailed information of these types can be obtained from the Help Browser.

6. Arrays and Matrices in MATLAB

MATLAB is especially designed for array and matrix operations. In this section, we will see how to perform array and matrix operations. One dimensional arrays are called vectors and they can be defined in rowwise or columnwise (row vector or column vector). If a vector is a *row vector* and it has n elements and the size of this vector is denoted by $1 \times n$. On the other hand, if a vector is a *column vector* and it has n elements and the size of this vector is denoted by $n \times 1$. The vector \mathbf{v} can also be represented by $[v_i]$ to indicate its elements v_i . The subscript " i " is called an indice, and indicates the location of element v_i

$$\mathbf{v} = \underbrace{\begin{bmatrix} v_1 & v_2 & \cdots & v_i & \cdots & v_n \end{bmatrix}}_{\substack{1 \times n \\ \text{row vector}}} \quad ; \quad \mathbf{v} = \underbrace{\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_i \\ \vdots \\ v_n \end{bmatrix}}_{\substack{n \times 1 \\ \text{column vector}}}$$

Two dimensional arrays are called matrices. Their sizes of matrices are denoted by $m \times n$, where m is the number of rows and n is the number of columns. Notice that, a row vector has a size of $1 \times n$, where n is the number of columns and a column vector has a size of $n \times 1$, where n is the number of rows. The matrix \mathbf{A} can also be represented by $[a_{ij}]$ to indicate its elements a_{ij} . The indices " i " and " j " shows the location of element a_{ij} . The first indice always denotes the row number, whereas the second indice always denotes the column number of the element a_{ij} .

$$\mathbf{A} = \underbrace{\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1j} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2j} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots & & \vdots \\ a_{i1} & a_{i2} & \cdots & a_{ij} & \cdots & a_{in} \\ \vdots & \vdots & & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mj} & \cdots & a_{mn} \end{bmatrix}}_{\substack{m \times n \\ \text{matrix}}}$$

6.1. Creating vectors and matrices

Creating row vectors in MATLAB. In order to create a vector or an array in MATLAB, we will use the brackets "[" and "]". To create a row vector, just type the elements by seperating with a space inside the brackets [and]. For example, to create a variable \mathbf{a} which contains a row vector with 4 elements as shown below;

$$\mathbf{a} = \left[1 \ 2 \ 3 \ 4 \right]_{1 \times 4}$$

type the following and press enter in the Command Window

```
>> >> a=[1 2 3 4]
```

a =

1 2 3 4

You can also seperate the elements by using comma ","

```
>> a=[1,2,3,4]
```

```
a =
```

```
    1    2    3    4
```

Creating column vectors in MATLAB. In order to create column vectors, you can either separate the elements by semicolons ";" or you can press enter after each element. For example, to create a column vector **b** as below

$$\mathbf{b} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}_{4 \times 1}$$

type the following in the Command Window

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

```
b =
```

```
    1
    2
    3
    4
```

or type the elements by pressing enter after each element.

```
>> b=[1
```

```
2
```

```
3
```

```
4]
```

```
b =
```

```
    1
    2
    3
    4
```

The other option for creating a column vector is to use to type a row vector and then use a transpose notation "'". This converts a row vector into a column vector. In the previous example vector **b** can be expressed as;

$$\mathbf{b} = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}^T$$

where subscript T denotes for transpose and in MATLAB this statement is to be typed as;

```
>> b=[1 2 3 4]'
```

```
b =
```

1
2
3
4

If you examine the above examples, you should notice that MATLAB displays the row vectors horizontally and column vectors vertically.

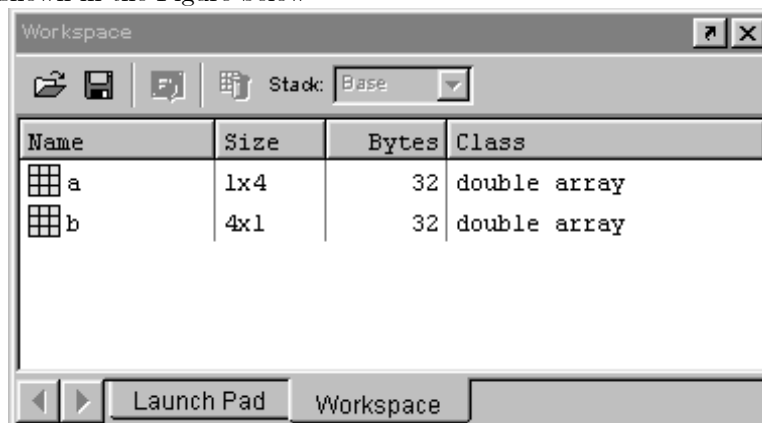
After entering the variables **a** and **b**, if you type **whos** command, MATLAB displays the properties of workspace variables.

```
>> whos
Name      Size      Bytes  Class

a         1x4         32  double array
b         4x1         32  double array
```

Grand total is 8 elements using 64 bytes

Notice that, the sizes of **a** and **b** are displayed as 1×4 and 4×1 respectively, denoting that variable **a** contains one row and four columns whereas variable **b** contains 4 rows and one column. This can also be seen from the Workspace browser as shown in the Figure below



In some applications, you need vectors containing very large number of elements where the values of the elements are regularly spaced. For this type of vectors, it is not practical to enter all the elements. In MATLAB we use colon ":" operator for creating vectors with regularly spaced elements. If you create a vector, say **x** by typing

```
>> x=[iv:inc:fv]
```

MATLAB creates a vector **x** of values with a spacing **inc** (*increment*). The initial value of **x** is **iv** and the final value of **x** is **fv**.

For example, suppose you want to create a row vector with 10 elements where the values of the first and last elements are 2 and 20 respectively. Also assume that, the values of the elements are equally spaced.

$$\mathbf{c} = [2 \ 4 \ 6 \ 8 \ 10 \ 12 \ 14 \ 16 \ 18 \ 20]$$

To create vector **c**, type

```
>> c=[2:2:20]
```

c =

2 4 6 8 10 12 14 16 18 20

Here the `iv= 2`, `inc= 2` and `fv= 20`. Suppose you give the final value `fv= 19` in the previous example by typing

`c=[2:2:19]`

c =

2 4 6 8 10 12 14 16 18

In this case, the final element of the vector becomes 18. This is because after the value 18 is created, MATLAB tries to add 2 to 18, but this exceeds to final limit 19 so MATLAB stops. Note that, the increments can also be negative.

EXAMPLE 26. Create a row vector $\mathbf{a} = [0.0 \ 0.1 \ 0.2 \ 0.3 \ 0.4 \ 0.5]$

`>> a=[0.0: 0.1: 0.5]`

a =

0 0.1000 0.2000 0.3000 0.4000 0.5000

EXAMPLE 27. Create a column vector $\mathbf{b} = \begin{bmatrix} 10 \\ 8 \\ 6 \\ 4 \\ 2 \\ 0 \end{bmatrix}$

`>> b=[10:-2:0]'`

b =

10
8
6
4
2
0

In addition to above procedure, MATLAB has two more commands for creating row vectors with regularly spaced elements:

- `linspace(x1,x2,n)`

`linspace` command creates a row vector containing `n` elements between the lower limit `x1` and upper limit `x2`

EXAMPLE 28. Create a row vector $\mathbf{c} = [-2 \ -1 \ 0 \ 1 \ 2]$ using `linspace` command

```
>> c=linspace(-2,2,5)
```

```
c =
```

```
-2 -1 0 1 2
```

- `logspace(a,b,n)`

`logspace` command creates a row vector containing `n` elements between 10^a and 10^b

Creating matrices in MATLAB. The easiest way to create a matrix is to type the matrix row by row, separating the rows by semicolon ";" or by pressing enter. In entering each row, you separate the elements by space or by comma ",". The following example illustrates this.

EXAMPLE 29. Create the matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$$

a) Separating the rows with semicolons

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

```
A =
```

```
1     2     3     4
5     6     7     8
9    10    11    12
```

b) Separating the rows by pressing enter

```
>> A=[1,2,3,4
```

```
5,6,7,8
```

```
9,10,11,12]
```

```
A =
```

```
1     2     3     4
5     6     7     8
9    10    11    12
```

You can also use type the elements of each row inside the brackets [and] to improve readability. The matrix A in the previous example can also be entered

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

```
A =
```

```
1     2     3     4
5     6     7     8
9    10    11    12
```

or

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

A =

```
    1     2     3     4
    5     6     7     8
    9    10    11    12
```

Appending vectors and matrices to each other. In MATLAB new vectors and matrices can be created by appending preexisting vectors and matrices to each other. This is a very powerful property which can be used in vector and matrix operations. Let us suppose you have created two row vectors $x = [1, 2, 3]$ and $y = [4, 5, 6]$. and you want to create another row vector with the form $z = [1, 2, 3, 4, 5, 6]$. If you type $z=[x, y]$, MATLAB forms a new row vector z by appending the vectors x and y .

```
>> x=[1,2,3];y=[4,5,6];
>> z=[x y]
```

z =

```
    1     2     3     4     5     6
```

you can also use a space instead of comma in the above example, that typing $z=[x y]$ gives the same result. We can also create a matrix from row or column vectors. You can also create column vectors in a similar way; let us assume this time we have two column vectors $x = [1, 2, 3]^T$ and $y = [4, 5, 6]^T$. To create a column vector $z = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix}^T$ just type $z=[x;y]$

```
>> x=[1;2;3];y=[4;5;6];
>> z=[x;y]
```

z =

```
    1
    2
    3
    4
    5
    6
```

Notice that this time, we append the column vectors by using a semicolon. The same result can be obtained by pressing enter instead of using semicolon

```
>> z=[x
y]
```

z =

1
2
3
4
5
6

We can append vectors with matrices and matrices with other matrices also. We will try to illustrate this property by giving some examples

Appending row vectors with a matrix. Any number of row vectors can be appended to a matrix \mathbf{A} , as long as these row vectors contain the same number of columns as \mathbf{A} . This means that, if the size of \mathbf{A} is $m \times n$, the sizes of the row vectors must be $1 \times n$. This kind of operation produces a new matrix with the size

$$(m + \text{number of row vectors appended}) \times n$$

and the operation is called *rowwise appending* (we increase the number of rows).

EXAMPLE 30. *Appending row vectors with a matrix. Given \mathbf{r}_1 , \mathbf{r}_2 and \mathbf{A}*

$$\mathbf{r}_1 = \begin{bmatrix} -1 & -2 & -3 & -4 \end{bmatrix} \quad ; \quad \mathbf{r}_2 = \begin{bmatrix} -10 & -20 & -30 & -40 \end{bmatrix} \quad ; \quad \mathbf{A} = \begin{bmatrix} 10 & 20 & 30 & 40 \\ 50 & 60 & 70 & 80 \\ 90 & 100 & 110 & 120 \end{bmatrix}$$

create the matrix \mathbf{B}

$$\mathbf{B} = \begin{bmatrix} -1 & -2 & -3 & -4 \\ 10 & 20 & 30 & 40 \\ 50 & 60 & 70 & 80 \\ 90 & 100 & 110 & 120 \\ -10 & -20 & -30 & -40 \end{bmatrix}$$

```
>> r1=[-1,-2,-3,-4];
>> r2=[-10,-20,-30,-40];
>> A=[10,20,30,40;50,60,70,80;90,100,110,120];
>> B=[r1;A;r2]
```

B =

```
-1    -2    -3    -4
 10    20    30    40
 50    60    70    80
 90   100   110   120
-10   -20   -30   -40
```

Note that, if we try to create matrix \mathbf{B} by typing $\mathbf{B}=[\mathbf{r}_1,\mathbf{A},\mathbf{r}_2]$ (using commas instead of semicolons) MATLAB will give us an error message. This is because, the number of rows of \mathbf{r}_1 and \mathbf{r}_2 are not equal to the number of rows of \mathbf{A} .

Appending column vectors with a matrix. Any number of column vectors can be appended to a matrix \mathbf{A} , as long as these column vectors contain the same number of rows as \mathbf{A} . This means that, if the size of \mathbf{A} is $m \times n$, the sizes of the column vectors must be $m \times 1$. This kind of operation produces a new matrix with the size

$$m \times (n + \text{number of column vectors appended})$$

and the operation is called *columnwise appending* (we increase the number of columns).

EXAMPLE 31. *Appending column vectors with a matrix. Given \mathbf{c}_1 , \mathbf{c}_2 and \mathbf{A}*

$$\mathbf{c}_1 = \begin{bmatrix} -1 \\ -2 \\ -3 \end{bmatrix} \quad ; \quad \mathbf{c}_2 = \begin{bmatrix} -10 \\ -20 \\ -30 \end{bmatrix} \quad ; \quad \mathbf{A} = \begin{bmatrix} 10 & 20 & 30 & 40 \\ 50 & 60 & 70 & 80 \\ 90 & 100 & 110 & 120 \end{bmatrix}$$

create the matrix \mathbf{B}

$$\mathbf{B} = \begin{bmatrix} -1 & 10 & 20 & 30 & 40 & -10 \\ -2 & 50 & 60 & 70 & 80 & -20 \\ -3 & 90 & 100 & 110 & 120 & -30 \end{bmatrix}$$

```
>> r1=[-1;-2;-3];
>> r2=[-10;-20;-30];
>> A=[10,20,30,40;50,60,70,80;90,100,110,120];
>> B=[r1,A,r2]
```

B =

```
-1    10    20    30    40   -10
-2    50    60    70    80   -20
-3    90   100   110   120   -30
```

Note that, if we try to create matrix \mathbf{B} by typing $\mathbf{B}=[\mathbf{r1};\mathbf{A};\mathbf{r2}]$ (using semicolons instead of commas) MATLAB will give us an error message. This is because, the number of columns of \mathbf{c}_1 and \mathbf{c}_2 are not equal to the number of columns of \mathbf{A} .

Appending matrices with other matrices. Matrices \mathbf{A} and \mathbf{B} can be appended to each other to produce new matrix \mathbf{C} as long as they have compatible sizes. In order to append matrices \mathbf{A} and \mathbf{B} , at least one their sizes must be same. If \mathbf{A} is a $m \times n$ matrix and \mathbf{B} is a $k \times n$ matrix, then, we can only append them in a *rowwise* sense which produces a matrix with the size $(m + k) \times n$ (the number rows are increased). On the other hand, If \mathbf{A} is a $m \times n$ matrix and \mathbf{B} is a $m \times k$ matrix, then, we can only append them in a *columnwise* sense which produces a matrix with the size $m \times (n + k)$ (the number of columns are increased). Of courses if \mathbf{A} and \mathbf{B} are square matrices with the same size, then they can be appended to each other both rowwise and columnwise.

EXAMPLE 32. *Given the matrices \mathbf{A} and \mathbf{B} as shown below. Append them to form matrix \mathbf{C} .*

$$\mathbf{A} = \begin{bmatrix} 10 & 20 \\ 30 & 40 \\ 50 & 60 \end{bmatrix} \quad ; \quad \mathbf{B} = \begin{bmatrix} -1 & -2 & -3 \\ -4 & -5 & -6 \\ -7 & -8 & -9 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} 10 & 20 & -1 & -2 & -3 \\ 30 & 40 & -4 & -5 & -6 \\ 50 & 60 & -7 & -8 & -9 \end{bmatrix}$$

These matrices can only be appended in a *rowwise* sense because only the row sizes are same

```
>> A=[10,20;30,40;50,60];
>> B=[-1,-2,-3;-4,-5,-6;-7,-8,-9];
>> C=[A,B]
```

C =

```
10    20    -1    -2    -3
30    40    -4    -5    -6
50    60    -7    -8    -9
```

EXAMPLE 33. Given the matrices **A** and **B** as shown below. Append them to form matrix **C**.

$$\mathbf{A} = \begin{bmatrix} 10 & 20 \\ 30 & 40 \\ 50 & 60 \end{bmatrix} \quad ; \quad \mathbf{B} = \begin{bmatrix} -1 & -2 \\ -3 & -4 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} -1 & -2 \\ -3 & -4 \\ 10 & 20 \\ 30 & 40 \\ 50 & 60 \end{bmatrix}$$

These matrices can only be appended in a *columnwise* sense because only the column sizes are same

```
>> A=[10,20;30,40;50,60];
>> B=[-1,-2;-3,-4];
>> C=[B;A]
```

C =

```
-1    -2
-3    -4
10    20
30    40
50    60
```

6.2. Using Array Editor

It should be recalled that, the term workspace refers to the names and values of variables that are stored in memory during the current working session. We have said that the information about each variable in the workspace can be obtained from the Workspace Browser. Using Workspace Browser you can also save, edit and clear workspace variables. To open Workspace browser you can either select **Workspace** from the **View** menu or type `workspace` from the Command Window. Below we list the operations that can be performed using Workspace Browser. Here we discuss the first 5 items, items 6 and 7 will be discussed later.

1. Viewing the workspace variables

All the variables that are loaded in memory during the current working session can be seen from the Workspace Browser. The name, size, type and class of each variable can be observed.

2. Clearing workspace variables

By selecting a variable and then pressing delete key, deletes the variable from the working session. You can also use the trash icon seen below



for deleting a variables after selecting.

3. Saving workspace variables

You can save your workspace variables by using the **Save** icon from the the Workspace Browser

4. Loading a saved workspace

You can open *mat* files which contain workspace variables that are previously saved by clicking the **Open** icon

5. Editing workspace variables

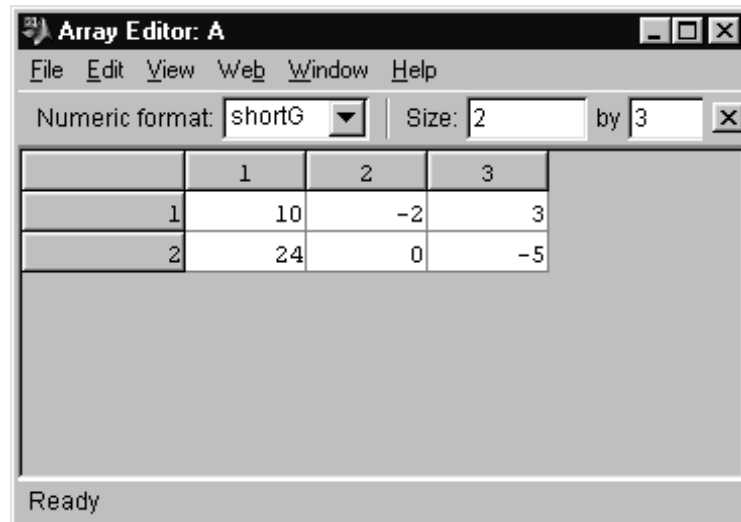
Workspace Browser contains a special graphical interface for working with arrays, namely *Array Editor*. To open *Array editor* first select a variable from the list in the workspace browser and then click open icon seen below.



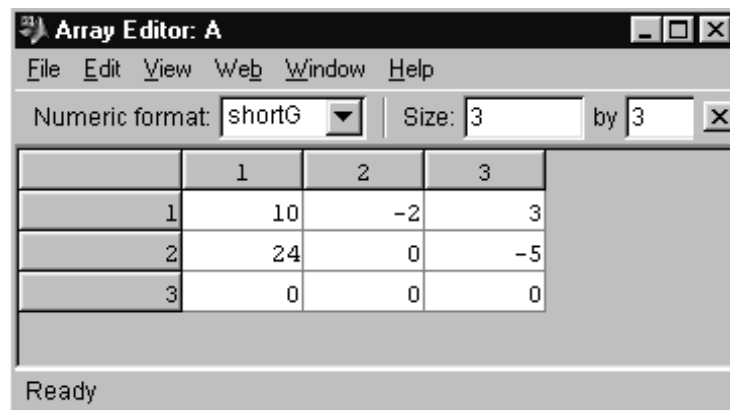
Assume that we have an 2×3 array **A** with the form

$$\mathbf{A} = \begin{bmatrix} 10 & -2 & 3 \\ 24 & 0 & -5 \end{bmatrix}$$

when you select variable **A** and click the open icon, MATLAB opens the *Array Editor* for variable *A* as shown below



Array Editor shows the variable name on the top of window and shows the elements of the array in a tabular form. By double clicking one of the elements, an edit field opens in which you can change the value of the element. There are two fields, **Numeric format** and **Size**. **Numeric Format** field shows the display format of the variable in the working session while **Size** field shows the size of the variable. In case of variable **A**, number of rows is equal to 2 and number of columns is equal to 3. The size of a variable can be changed from the Size field by changing the row and column numbers. For example, if you want to add a row to variable **A**, just type 3 in the first field of the **Size** field. This will add a row filled with zeros at the end of second row of **A**.



The second way to open *Array Editor* for a variable is double clicking on it from the list in the Workspace Browser. You can also open Array Editor for a variable by typing `open('var')` from the command window, where `var` is the name of variable.

6. Plotting workspace variables

7. Viewing base and function workspaces by using a stack

6.3. Working with Array elements

In the beginning of this Chapter, we use indices to indicate the elements of vectors and matrices. A vector \mathbf{v} can be represented by $[v_i]$ to indicate the elements v_i where indice i is used to denote i 'th element of a vector. In case of matrices, we have said that a matrix \mathbf{A} can be represented by $[a_{ij}]$ to indicate its elements a_{ij} where indices i and j denotes the row and column number of the element a_{ij} . In MATLAB, the elements of vectors and matrices are addressed similarly. For example, consider a 1×5 vector \mathbf{v} and a 3×4 matrix \mathbf{A} defined as

$$\mathbf{v} = \begin{bmatrix} -6 & 3 & 1 & -4 & 0 & 5 \end{bmatrix} \quad ; \quad \mathbf{A} = \begin{bmatrix} 2 & 8 & -1 & 9 \\ -4 & 6 & 3 & 5 \\ 1 & 0 & 3 & 2 \end{bmatrix}$$

and we want to get $v_3 = 1$ and $a_{23} = 3$. Typing `v(3)` and `A(2,3)` from the Command Window, gives us the values 1 and 3 respectively.

EXAMPLE 34. Vectors \mathbf{a} , \mathbf{b} and matrix \mathbf{C} is defined below. Obtain the values a_2 , b_4 and c_{35} from MATLAB

$$\mathbf{a} = \begin{bmatrix} 1.2 & -3.4 & 5.2 & 7.7 & -6.8 & 0.3 \end{bmatrix} \quad ; \quad \mathbf{b} = \begin{bmatrix} -0.2 \\ 3.1 \\ 7.8 \\ -8.5 \\ 9.0 \end{bmatrix} \quad ; \quad \mathbf{C} = \begin{bmatrix} 3.4 & 7.0 & -1.6 & -9.2 & 0.3 \\ 6.2 & 3.7 & 8.2 & 5.1 & -2.1 \\ 0.4 & 2.8 & -3.4 & 8.4 & -2.4 \end{bmatrix}$$

```
>> a=[1.2, -3.4, 5.2, 7.7, -6.8, 0.3];
>> b=[-0.2; 3.1; 7.8; -8.5; 9.0];
>> C=[3.4, 7.0, -1.6, -9.2, 0.3
6.2, 3.7, 8.2, 5.1, -2.1
0.4, 2.8, -3.4, 8.4, -2.4];
>> a(2)
```

ans =

-3.4000

```
>> b(4)
```

ans =

-8.5000

```
>> C(3,5)
```

ans =

-2.4000

You can change the values of the elements of an array by assigning a new values.

EXAMPLE 35. Change the value of the element c_{23} to -6.4 of matrix \mathbf{C} in the previous example.

>> C(2,3)=-6.4

C =

```

3.4000    7.0000   -1.6000   -9.2000    0.3000
6.2000    3.7000   -6.4000    5.1000   -2.1000
0.4000    2.8000   -3.4000    8.4000   -2.4000

```

In MATLAB, colon ":" operator is used to select a range of elements in arrays. We will explain the use of colon operator by giving examples.

- $\mathbf{v}(:)$ represents all the row or column elements of the vector \mathbf{v}
- $\mathbf{v}(2:5)$ represents the second through fifth elements in vector \mathbf{v} , that is $\mathbf{v}(2)$, $\mathbf{v}(3)$, $\mathbf{v}(4)$, $\mathbf{v}(5)$
- $\mathbf{A}(:,3)$ denotes all the elements in the third column of the matrix \mathbf{A} .
- $\mathbf{A}(:,2:5)$ denotes all elements in the second through fifth columns of \mathbf{A} .
- $\mathbf{A}(2:3, 1:3)$ denotes all the elements in the second and third rows that are also in the first through third columns.

EXAMPLE 36. Vector \mathbf{a} and matrix \mathbf{C} is defined below

$$\mathbf{a} = \begin{bmatrix} 10 & 20 & 30 & 40 & 50 \end{bmatrix} ; \quad \mathbf{F} = \begin{bmatrix} 11 & 12 & 13 & 14 & 15 \\ 21 & 22 & 23 & 24 & 25 \\ 31 & 32 & 33 & 34 & 35 \\ 41 & 42 & 43 & 44 & 45 \\ 51 & 52 & 53 & 54 & 55 \end{bmatrix}$$

Using colon ":" operator create the following vectors \mathbf{b} , \mathbf{c} , \mathbf{d} from vector \mathbf{a}

$$\mathbf{b} = \begin{bmatrix} 10 & 20 & 30 \end{bmatrix} ; \quad \mathbf{c} = \begin{bmatrix} 40 & 50 \end{bmatrix} ; \quad \mathbf{d} = \begin{bmatrix} 20 & 30 & 40 \end{bmatrix}$$

Using colon ":" operator create the following arrays (subarrays) and vectors from the matrix \mathbf{F}

$$\mathbf{M}_1 = \begin{bmatrix} 22 & 23 & 24 \\ 32 & 33 & 34 \end{bmatrix} ; \quad \mathbf{M}_2 = \begin{bmatrix} 13 & 14 \\ 23 & 24 \end{bmatrix} ; \quad \mathbf{v}_1 = \begin{bmatrix} 14 \\ 24 \\ 34 \\ 44 \\ 54 \end{bmatrix} ; \quad \mathbf{v}_2 = \begin{bmatrix} 42 & 43 & 44 \end{bmatrix}$$

```
>> a=[10,20,30,40,50];  
>> b=a(1:3)
```

```
b =
```

```
    10    20    30
```

```
>> c=a(4:5)
```

```
c =
```

```
    40    50
```

```
>> d=a(2:4)
```

```
d =
```

```
    20    30    40
```

```
>> F=[11,12,13,14,15
```

```
21,22,23,24,25
```

```
31,32,33,34,35
```

```
41,42,43,44,45
```

```
51,52,53,54,55];
```

```
>> m1=F(2:3,2:4)
```

```
m1 =
```

```
    22    23    24
```

```
    32    33    34
```

```
>> m2=F(1:2,3:4)
```

```
m2 =
```

```
    13    14
```

```
    23    24
```

```
>> v1=F(:,4)
```

```
v1 =
```

```
    14
```

```
    24
```

```
    34
```

```
    44
```

```
    54
```

```
>> v2=F(4,2:4)
```

v2 =

42 43 44

7. Some basic array functions

In the table given below, we summarize some of the basic array functions that can be used in MATLAB

Command	Description
<code>linspace(a,b,n)</code>	Creates a <i>row vector</i> of n regularly spaced values between a and b
<code>logspace(a,b,n)</code>	Creates a <i>row vector</i> of n logarithmically spaced values between a and b
<code>max(A)</code>	Returns algebraically largest element in \mathbf{A} if \mathbf{A} is a vector. Returns a row vector containing the largest elements in each column if \mathbf{A} is a matrix.
<code>min(A)</code>	Returns algebraically smallest element in \mathbf{A} if \mathbf{A} is a vector. Returns a row vector containing the smallest elements in each column if \mathbf{A} is a matrix.
<code>size(A)</code>	Returns a row vector $[m,n]$ containing the sizes of $m \times n$ array \mathbf{A}
<code>sum(A)</code>	Sums the elements in each column of the array \mathbf{A} and returns a row vector containing the sums

8. Some basic array operations

In the table given below, we summarize some of the basic array operations

Assume c is a scalar, A and B are arrays			
Symbol	Operation	Form	Example
+	Scalar array addition	$A+c$	$[6,3]+3=[9,6]$
-	Scalar array subtraction	$A-c$	$[17,-5]-7=[10,-12]$
*	Scalar array multiplication	$A*c$	$5*[3,-2]=[15,-10]$
/	Scalar array division	A/c	$[3,-2]/2=[1.5,-1.0]$
+	Array addition	$A + B$	$[5,2]+[3,-4]=[8,-2]$
-	Array subtraction	$A - B$	$[5,2]-[3,-4]=[2,6]$
.*	Array multiplication	$A.*B$	$[3,5].*[4,8]=[12,40]$
./	Array right division	$A./B$	$[2,5]./[4,8]=[2/4,5/8]$
.\	Array left division	$A.\B$	$[2,5).\[4,8]=[2\4,5\8]$
.^	Array exponentiation	$A.^B$	$[3,5).^2=[3^2,5^2]$ $2^[3,5]=[2^5,2^5]$ $[3,5).^[4,5]=[3^4,4^5]$
*	Matrix multiplication	$A*B$	$\begin{bmatrix} 2 & 3 \end{bmatrix} \begin{bmatrix} 5 \\ 7 \end{bmatrix} \quad [2,3]*[5;7]=31$ $\begin{bmatrix} 5 \\ 7 \end{bmatrix} \begin{bmatrix} 2 & 3 \end{bmatrix} \quad [5;7]*[2,3]= \begin{matrix} 10 & 15 \\ 14 & 21 \end{matrix}$ $\begin{bmatrix} 2 & 9 \\ 5 & -7 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \end{bmatrix} \quad [2,9;5,-7]*[2;3]= \begin{matrix} 31 \\ -11 \end{matrix}$ $\begin{bmatrix} 2 & 9 \\ 5 & -7 \\ 1 & 9 \end{bmatrix} \begin{bmatrix} 6 & 1 \\ 3 & 2 \end{bmatrix} \quad [2,9;5,-7;1,9]*[6,1;3,2]= \begin{matrix} 39 & 20 \\ 9 & -9 \\ 33 & 19 \end{matrix}$

9. Special matrices

Using MATLAB you can easily create special matrices such as identity or null matrices. The table below shows MATLAB commands for creating special matrices

Command	Description
<code>eye(n)</code>	Creates an $n \times n$ identity matrix
<code>eye(size(A))</code>	Creates an identity matrix the same size as the matrix A
<code>ones(n)</code>	Creates an $n \times n$ matrix of ones
<code>ones(m,n)</code>	Creates an $m \times n$ matrix of ones
<code>ones(size(A))</code>	Creates an array of ones the same size as the array A
<code>zeros(n)</code>	Creates an $n \times n$ matrix of zeros
<code>zeros(m,n)</code>	Creates an $m \times n$ matrix of zeros
<code>zeros(size(A))</code>	Creates an array of zeros the same size as the array A

10. Matrix Commands for solving linear equations

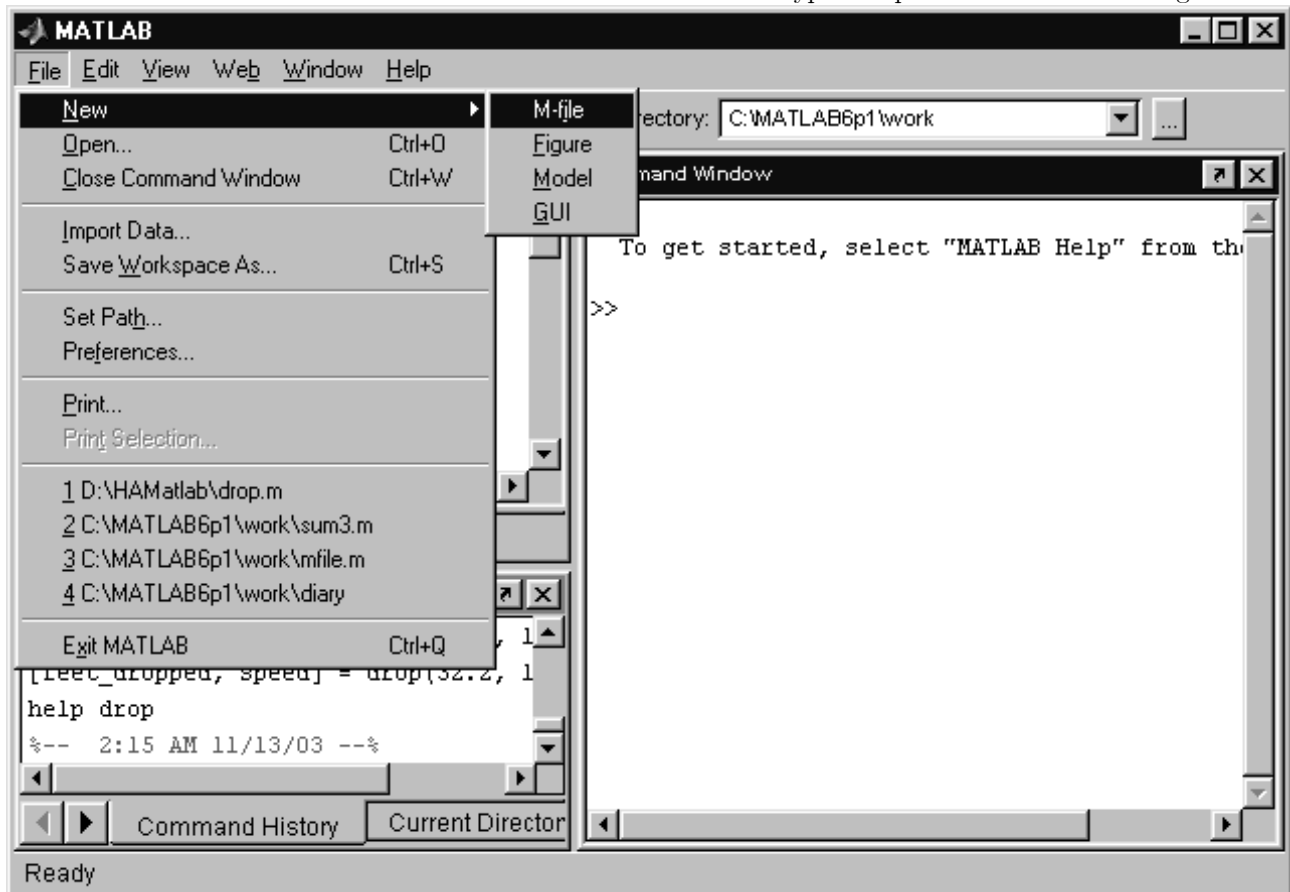
Command	Description
<code>det(A)</code>	Computes the determinant of the array \mathbf{A}
<code>inv(A)</code>	Computes the inverse of the matrix \mathbf{A}
<code>pinv(A)</code>	Computes the pseudoinverse of the matrix \mathbf{A}
<code>rank(A)</code>	Computes the rank of the matrix \mathbf{A}
<code>rref([A b])</code>	Computes the reduced row echolen form corresponding to the augmented matrix $[\mathbf{A} \ \mathbf{b}]$
<code>x = inv(A)*b</code>	Solves the matrix equation $\mathbf{Ax} = \mathbf{b}$, using matrix inverse
<code>x = A\b</code>	Solves the matrix equation $\mathbf{Ax} = \mathbf{b}$, using left division (Gauss Elimination)
<code>x = d/C</code>	Solves the matrix equation $\mathbf{Ax} = \mathbf{b}$, using right division.

11. M-Files, User defined functions

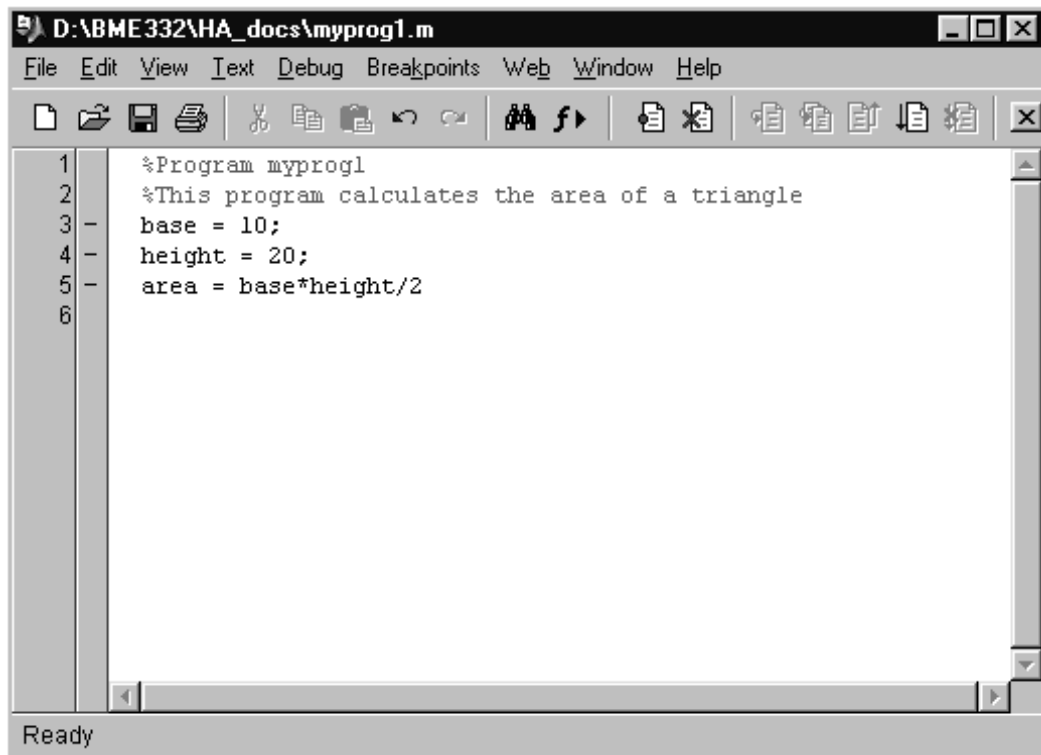
In the preceding sections, we have used diary files for storing our keystrokes in a working session and use mat files for saving our workspace variables. Another important file type of MATLAB is M-files. M-files have the extension `.m` such as `myfile.m`. There are two types of M-files in MATLAB: *script files* and *function files*. A *script file* contains a list of MATLAB commands while a *function file* contains the definition of MATLAB functions as well as the functions that you create. When you type the name of the M-file (without typing its extension `.m`) from the Command Window, all the commands in this file are executed.

11.1. Creating an M-file and using it as a script file

M-files can be created by using any text editors because they are ASCII files. It should be recalled that, ASCII files are the files which can be recognized nearly by all computer systems. The simplest way to create an M-file is to use MATLAB's *Editor/Debugger*. To create an M-file using MATLAB's *Editor/Debugger*, select **New** from the **File** menu in the Command Window. You will see that a list of file types is opened as shown in the figure below.



If you select **M-File** from that list, MATLAB's *Editor/Debugger* window will be opened. First, you have to save this file. For that, select **Save** from the **File** menu in the *Editor/Debugger* and replace the default name (`untitled`) provided by MATLAB with `myprog1`. Before saving this file, do not forget to check the location of *Current directory*. When you save the file, *MATLAB* will automatically provide the extension `.m` to the end of the file name `myprog1`, that is, the file will be saved into your *Current Directory* as `myprog1.m`. Notice that, immediately after saving the file, the new file name and its location appears at the top of the dialog box of *Editor/Debugger*. Inside the *Editor/Debugger*, type exactly the same as shown below in the figure.



After typing, save the changes and close the *Editor/Debugger*. Once you have saved your M-file, you can type the name of this M-file in the command window and press enter. For our example, type `myprog1` and press enter in the command window. You see the following output

```
>> myprog1
```

```
area =
```

```
100
```

This type of usage of an M-file is called a *script file*. As we have said, when you type the name of the M-file without its extension, the commands in the file are executed one by one. If you look to the Workspace Browser, you will see the variables that you typed in the script file `myprog1.m`. This shows that, the variables that you have been created in a script file are **global**.

When working with M-files, there are some important things that you have to keep in mind.

- Do not forget to save the M-file before calling it from the Command Window.
- The naming of your M-file, has to obey the rules for naming variables in MATLAB, that is the filename must start with a letter and can contain up to 31 characters.
- Although, it is possible to use capital letters in the name of your M-file, do not use them. That is, always use small letters in naming your M-files (Example: use `myprog1.m` instead of `MYprog1.m`). Also recall that, in MATLAB, variable names are case sensitive as in C. The names `xsum` and `Xsum` correspond to different variables.
- Do not use the names of MATLAB's built in commands and functions for your filename.
- Do not use a filename that you have given already a variable in a working session. For example, if you have already defined a variable named `sum2` in the working session, you can not give this name to an M-file. In order

to check whether a variable name exists in the working session or not, type `exist('name_variable')` from the Command Window, where `name_variable` is the name of the variable that you are checking. For example, to check whether variable `sum2` exists or not, type `exist('sum2')`. This will return with answer 0 or 1. If the answer is 0, it means that the variable name does not exist. If the answer is 1, it means that the variable name already exists. You can also check whether an M-file exists or not by typing `exist('filename.m','file')` from the Command Window where `filename.m` is the name of the M-file you are checking. The answer is either 0 or 1 depending on whether the `filename.m` exists or not in the working session. Finally, to check whether your variable name or filename is the same as MATLAB's built in commands and functions, type `exist('name','builtin')` from the command window. Again, the answer is either 0 or 1 depending on whether the `name` exists as a built in command or a function name in MATLAB.

You can use `type` command to view an M-file without opening it. For example type the following

```
>> type myprog1
%Program myprog1
%This program calculates the area of a triangle
base = 10;
height = 20;
area = base*height/2
```

11.2. User defined functions

Another type of M-files are function files as we stated above. Function files are useful, when a set of commands are repeated several times. They are similar to the functions in C. To create a user defined function in MATLAB, you have to open a new M-file and type

```
function [output variables] = function_name(input variables);
```

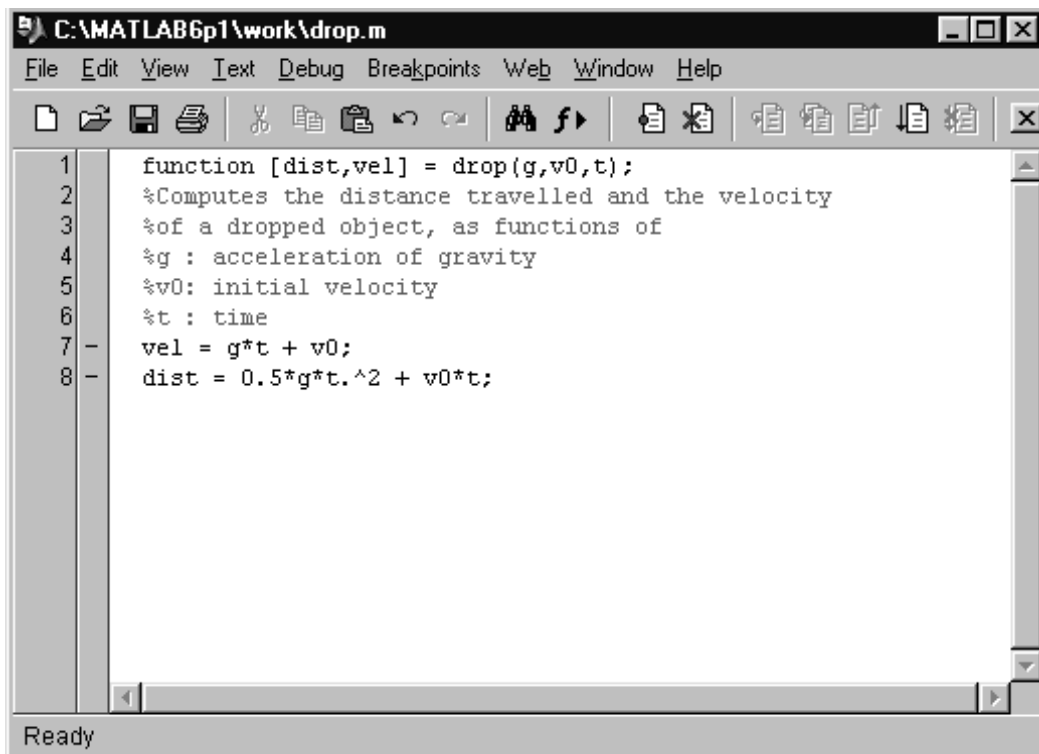
Here, `function` is MATLAB keyword that is used to define MATLAB functions. After the `function` keyword `output variables` must be listed inside the square brackets. Following the assignment operator '=', `function_name` is to be specified. Finally, the names of your `input variables` must be listed inside parenthesis.

Note that, your `function_name` must be the same as the filename of the M-file in which it is to be saved. Also the name of your function `function_name` must be lowercase. This is because, even though MATLAB is case sensitive by default, your operating system may not. Suppose that you have created two functions with the names `Drop` and `drop`. As stated above, the name of your M-files must be `Drop.m` and `drop.m`. If your operating system is not case sensitive `Drop.m` and `drop.m` might be treated as the same files and this may lead to confusion in MATLAB when calling these functions.

EXAMPLE 37. Create a user defined function named `drop` which computes a falling objects velocity and distance. The velocity and distance of a falling object is given by

$$\begin{aligned}v &= v_0 + gt \\d &= \frac{1}{2}gt^2 + v_0t\end{aligned}$$

Here, v and d are the final velocity and distance of a falling object after time t respectively. v_0 is the initial velocity of the object and g is the acceleration of gravity. Note that, the output variables must be the velocity v and distance d , while the input variables must be time t , initial velocity v_0 and acceleration of gravity g . First, open a new M-file from the **File** menu and type the following in your M-file



```

1 function [dist,vel] = drop(g,v0,t);
2 %Computes the distance travelled and the velocity
3 %of a dropped object, as functions of
4 %g : acceleration of gravity
5 %v0: initial velocity
6 %t : time
7 - vel = g*t + v0;
8 - dist = 0.5*g*t.^2 + v0*t;

```

Next, save the M-file as `drop.m` and exit from the Editor/Debugger. Recall, that `drop.m` is saved to your Current Directory. Notice that, in writing the distance formula, we have used array exponentiation `".^"` in order to enter time t as array. Now you can call your new function from the Command Window by typing its name. In the examples below we show various ways to call our function `drop`.

1. The variable names used in writing our function may, but need not, be used when the function is called.

```

>> a=9.81;
>> initial_speed =10;
>> time=5;
>> [meters_dropped, velocity] = drop(a,initial_speed,time)

```

meters_dropped =

172.6250

velocity =

59.0500

2. The input variables need not be assigned values outside the function prior to the function call

```

>> [meters_dropped, velocity] = drop(9.81,10,5)

```

meters_dropped =

172.6250

```
velocity =
```

```
59.0500
```

3. Since array exponentiation `.^` was used in the distance formula, you can also enter time as an array

```
>> [meters_dropped, velocity] = drop(9.81,10,[0:1:5])
```

```
meters_dropped =
```

```
0 14.9050 39.6200 74.1450 118.4800 172.6250
```

```
velocity =
```

```
10.0000 19.8100 29.6200 39.4300 49.2400 59.0500
```

The output variables are also arrays each containing 6 elements as the input time array.

4. If no output variable is specified in calling the function, the function returns the calculation of the first output variable by storing the result into `ans`

```
>> drop(9.81,10,5)
```

```
ans =
```

```
172.6250
```

Above, we introduced the general syntax of the function definition line as

```
function [output variables] = function_name(input variables);
```

If there is one output variable, the square brackets are optional, that is

```
function [area_square] = square(side);
```

can be written as

```
function area_square = square(side);
```

There can also be functions with no output variables such as a function that generates a plot. Such a function definition is written as

```
function sqplot(side);
```

The names of the input variables given in the function definition line are local to that function. This means that other variable names can be used when you call the function as in the previous examples. All variables inside the function are erased after the function finishes executing, except when the same variable name appears in the output variable list used in the function call.