MATLAB MODULE 1

MATLAB Window Environment and the Base Program

Getting Help

MATLAB provides hundreds of built-in functions covering various scientific and engineering computations. With numerous built-in functions, it is important to know how to look for functions and how to learn to use them.

For those who want to look around and get a feel for the MATLAB computing environment by clicking and navigating through what catches their attention, a window- based help is a good option. To activate the Help window, type **helpwin** or **helpdesk** on command prompt or start the Help Browser (Fig. M1.13) by clicking the  icon from the desktop toolbar.

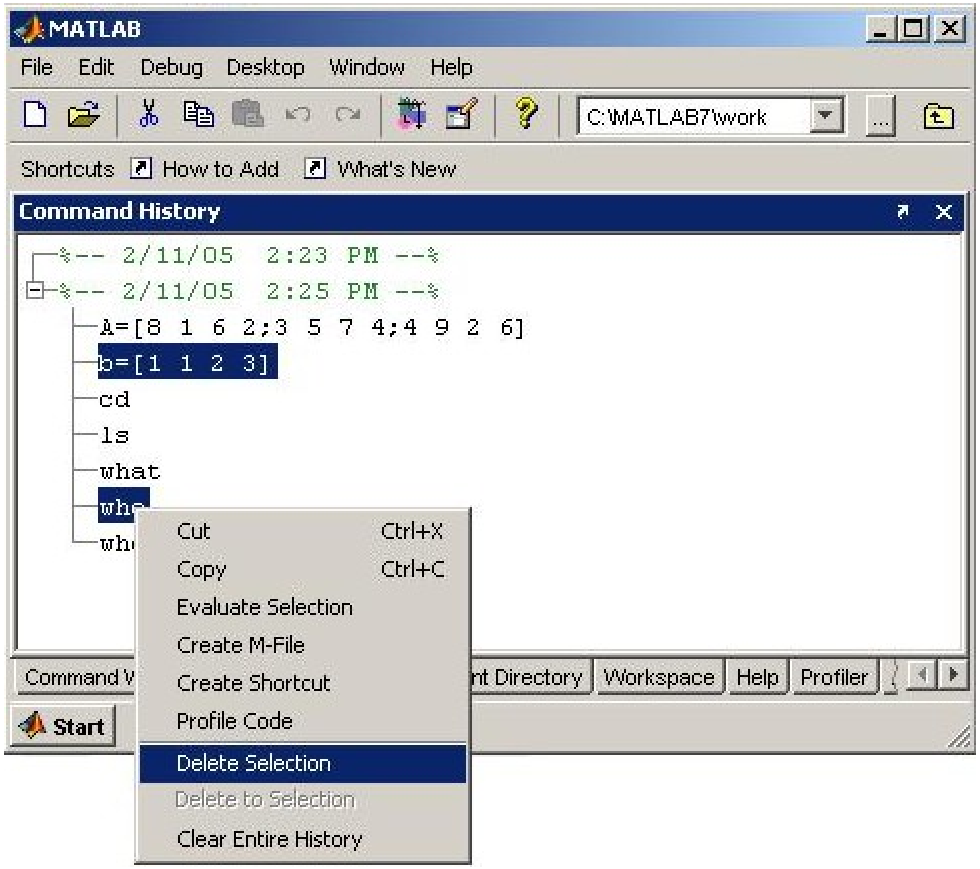

Fig. M1.12 *Command history window with two commands being deleted*

If you know the exact name of a command, type **help *commandname*** to get detailed task-oriented help. For example, type **help helpwin** in the command window to get the help on the command **helpwin** .

If you don't know the exact command, but (atleast !) know the keyword related to the task you want to perform, the **lookfor** command may assist you in tracking the exact command. The **help** command searches for an exact command name matching the keyword, whereas the **lookfor** command searches for quick summary information in each command related to the keyword. For example, suppose that you were looking for a command to take the inverse of a matrix. MATLAB does not have a command named **inverse**; so the command **help inverse** will not work.

In your MATLAB command window try typing **lookfor inverse** to see the various commands available for the keyword **inverse**.

MATLAB has a wonderful demonstration program that shows its various features through interactive graphical user interface. Type **demo** at the MATLAB prompt to invoke the demonstration program (Fig. M1.14) and the program will guide you throughout the tutorials.

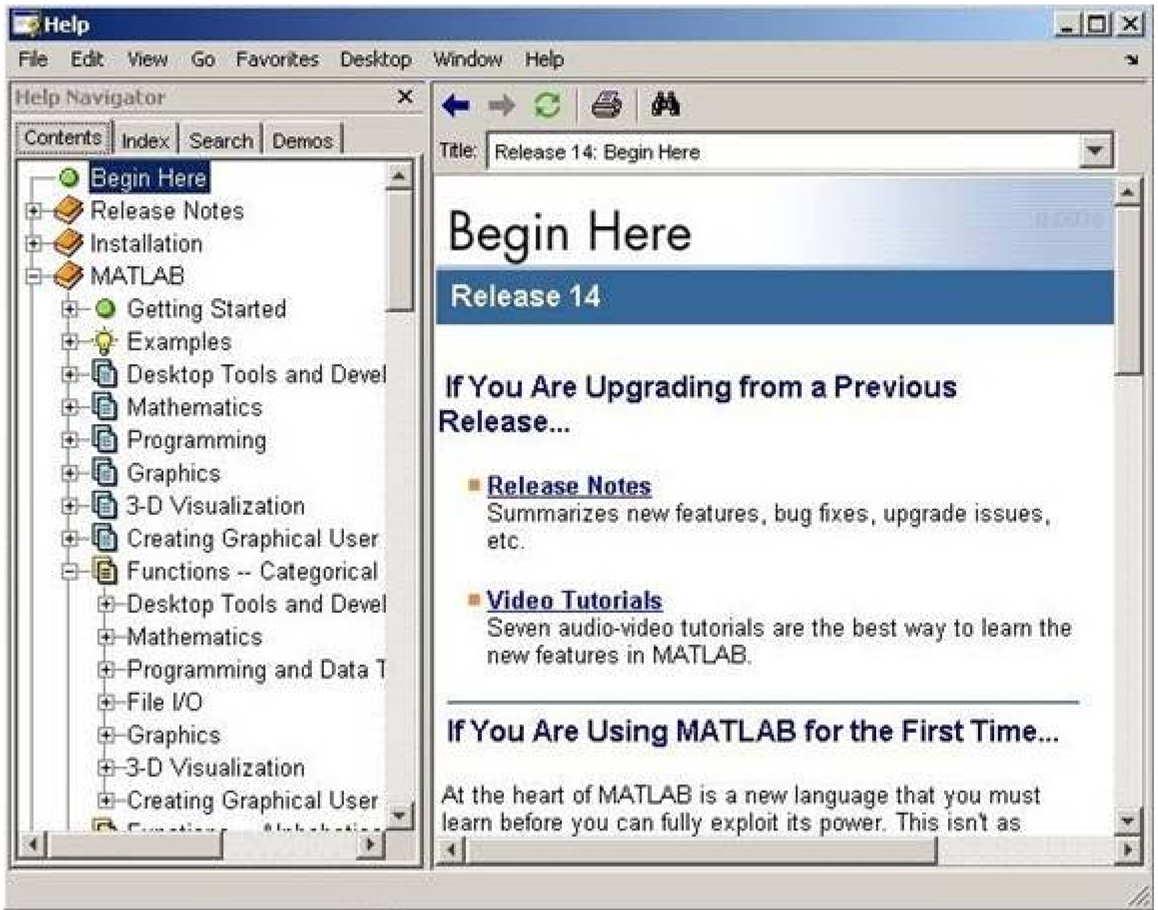

Fig. M1.13 *Help browser*

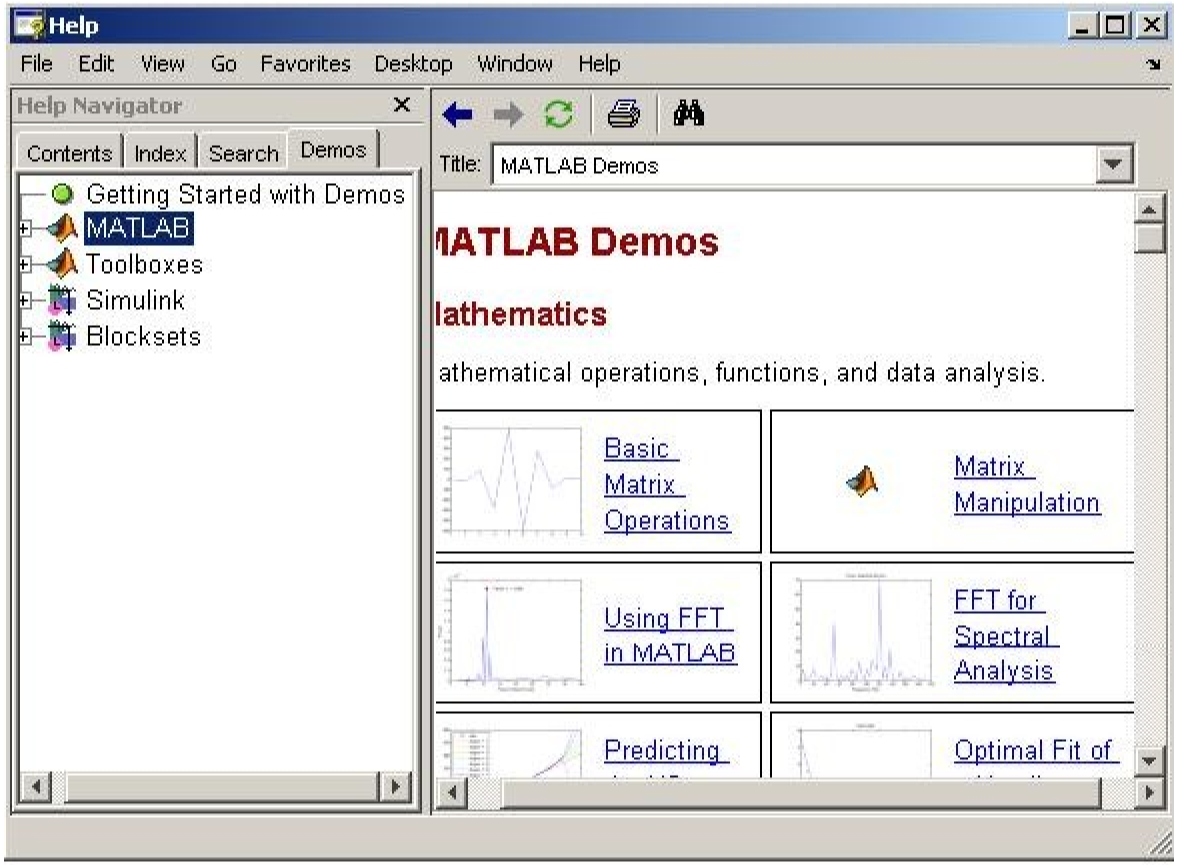

Fig. M1.14 *Demonstration Window*

Elementary Matrices

Basic data element of MATLAB is a matrix that does not require dimensioning. To create the matrix variable in

MATLAB workspace, type the statement (note that any operation that assigns a value to a variable, creates the variable, or overwrites its current value if it already exists).

>> A=[8 1 6 2;3 5 7 4;4 9 2 6] ↵

The blank spaces (or commas) around the elements of the matrix rows separate the elements. Semicolons separate the rows. For the above statement, MATLAB responds with the display

```
A =

8   1   6   2

3   5   7   4

4   9   2   6
```

Vectors are special class of matrices with a single row or column. To create a column vector variable in MATLAB workspace, type the statement

>> b=[1; 1; 2; 3] ↵

```
b =

   1

   1

   2

   3
```

To enter a row vector, separate the elements by a space or comma ' **,** '. For example:

>> b=[1,1,2,3] ↵

b =

1 1 2 3

We can determine the size of the matrices (number of rows, number of columns) by using the **size** command.

>> size(A) ↵

ans =

3 4

The command **size**, when used with the scalar option, returns the length of the dimension specified by the scalar. For example, **size (A,1)** returns the number of rows of **A** and **size(A,2)** returns the number of columns of **A**.

>> size(A,1) ↵

ans =

3

>> size(A,2) ↵

ans =

4

For matrices, the **length** command returns either number of rows or number of columns, whichever is larger. For example,

>> length(A) ↵

ans =

4

For vectors, **length** command can be used to determine its number of elements.

```
>> length(b) ↵

ans =

    4
```

The use of colon (**:**) operator plays an important role in MATLAB. This operator may be used to generate a row vector containing the numbers from a given starting value **xi**, to the final value **xf**, with a specified increment **dx**, e.g., **x=[xi:dx:xf]**

```
>> x=[0:0.1:1] ↵

x =

  Columns 1 through 7

       0    0.1000    0.2000    0.3000    0.4000    0.5000    0.6000

  Columns 8 through 11

    0.7000    0.8000    0.9000    1.0000
```

By default, the increment is taken as unity.

To generate linearly equally spaced samples between *x1* and *x2*, use the command **linspace(x1,x2)** . By default, 100 samples will be generated. The command **linspace (x1,x2, N)** allows the control over number of samples to be generated. See the example below.

```
>> x=linspace(0,1,11)

x =

  Columns 1 through 6

       0    0.1000    0.2000    0.3000    0.4000    0.5000
```

Columns 7 through 11

0.6000 0.7000 0.8000 0.9000 1.0000

Learn how to generate logarithmically spaced vector using the command **logspace** .

The colon operator can also be used to subscript matrices. For example, **A(:,j)** is the j^{th} column of **A**, and **A(i,:)** is the i^{th} row of **A**. Observe the following MATLAB session.

```
>> A=[8 1 6 2;3 5 7 4;4 9 2 6]; ↵

>> A(2,:) ↵

ans =

    3     5     7     4

>> A(3,2:4) ↵

ans =

    9     2     6

>> A(1,3) ↵

ans =

    6

>> B=A(1:3,2:3) ↵

B =

    1     6
```

5 7

9 2

>> A(:,3)=[ ] ↵

A =

8 1 2

3 5 4

4 9 6

Manipulating matrices is almost as easy as creating them. Try the following operations:

>> A+3 ↵

>> A-3 ↵

>> A*3 ↵

>> A/3 ↵

When you add/subtract/multiply/divide a vector/matrix by a number (or by a variable with a number assigned to it), MATLAB assumes that all elements of vector/matrix should be individually operated on.

Table M1.3 provides the list of basic operations on any two arbitrary matrices **A** and **B** and their dimensional requirements.

Table M1.3 *Basic matrix operations*

Operation	Operator	Example	Notes

Plus	+	**A+B**	Must be of same dimensions
Minus	-	**A-B**	Must be of same dimensions
Multiply	*	**A*B**	Must be of compatible dimensions
Multiply (element-by-element)	.*	**A.*B**	Must be of same dimensions; multiplies element a_{ij} with element b_{ij}
Divide (element-by-element)	./	**A./B**	Must be of same dimensions; divides element a_{ij} by element b_{ij}
Divide (element-by-element)	.\	**A.\B**	Must be of same dimensions; divides element b_{ij} by element a_{ij}
Matrix power	^	**A^k**	k must be a constant, A must be a square matrix
Matrix power (element-by-element)	.^	**A.^k**	k is a constant, A can be of any dimensions; gives $(a_{ij})^k$

Example M1.1

To find the solution of the following set of linear equations:

$$
\begin{aligned}
2x_1 + 5x_2 - 3x_3 &= 6 \\
3x_1 - 2x_2 + 4x_3 &= -2 \\
x_1 + 6x_2 - 4x_3 &= 3
\end{aligned}
$$

we write the equations in the matrix form as

$$\mathbf{Ax} = \mathbf{b}$$

where

$\mathbf{A}$ is the matrix of coefficients of x_1, x_2 and x_3

$\mathbf{x}$ is the column vector which will contain the solutions x_1, x_2 and x_3

$\mathbf{b}$ is the column vector of values on the right-hand side

$$\mathbf{A} = \begin{bmatrix} 2 & 5 & -3 \\ 3 & -2 & 4 \\ 1 & 6 & -4 \end{bmatrix}; \mathbf{x} = \begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix}; \mathbf{b} = \begin{bmatrix} 6 \\ -2 \\ 3 \end{bmatrix}$$

The solution vector

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b} = \frac{\mathbf{A}^{+}\mathbf{b}}{|\mathbf{A}|}$$

where $\mathbf{A}^{+}$ stands for adjoint of matrix $\mathbf{A}$ and $|\mathbf{A}|$ stands for determinant of $\mathbf{A}$

The determinant of $n \times n$ matrix

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

is a scalar-valued function of $\mathbf{A}$. It is found through the use of minors and cofactors.

The *minor* m_{ij} of the element a_{ij} is the determinant of a matrix of order $(n-1)\times(n-1)$ obtained from $\mathbf{A}$ by removing the row and column containing a_{ij}. The *cofactor* c_{ij} of the element a_{ij} is defined by the equation

$$c_{ij} = (-1)^{i+j} m_{ij}$$

Determinants can be evaluated by an expansion that reduces the evaluation of an $n\times n$ determinant down to the evaluation of a string of $(n-1)\times(n-1)$ determinants, namely the cofactors. Selecting an arbitrary row *k* of matrix $\mathbf{A}$ or arbitrary column *l* of matrix $\mathbf{A}$, we have

$$|\mathbf{A}| = \sum_{j=i}^{m} a_{kj} c_{kj}$$

or

$$|\mathbf{A}| = \sum_{l=i}^{n} a_{il} c_{il}$$

The adjoint of $n\times n$ matrix $\mathbf{A}$ is found by replacing each element a_{ij} of matrix $\mathbf{A}$ by its cofactor and then transposing.

$$\mathbf{A}^{+} = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix}^{T} = \begin{bmatrix} c_{11} & c_{21} & \cdots & c_{n1} \\ c_{12} & c_{22} & \cdots & c_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ c_{1n} & c_{2n} & \cdots & c_{nn} \end{bmatrix}$$

Following MATLAB commands solve the given set of simultaneous linear equations.

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

>> b = [6; -2; 3];

>> x = inv(A) * b

x =

4.8333

-4.5833

-6.4167

Exercise M1.4

Consider three matrices **A, B,** and **C** given below. Perform the following operations: **A+B, B-C, A*C, A.*B, A./C, A.\B, A./B, (B*C)^3,** and **C.^3.** Countercheck MATLAB answers manually. Try to interpret errors, if any.

$$\mathbf{A}=\begin{bmatrix}8 & 1 & 6\\ 3 & 5 & 7\\ 4 & 9 & 2\end{bmatrix};\mathbf{B}=\begin{bmatrix}6 & -1 & 4\\ 1 & 3 & 5\\ 2 & 7 & 0\end{bmatrix};\mathbf{C}=\begin{bmatrix}3\\ 5\\ 8\end{bmatrix}$$

Exercise M1.5

Create a vector **t** with 10 elements 1,2,.,10. Calculate for $\omega(t)$ and $\tau = 2$, w $k = 0.75$

$$\omega(t) = k\left(1 - e^{\frac{-t}{\tau}}\right).$$

Exercise M1.6

Create a vector **t** with initial time $t_0 = 0$ and final time $t_f = 5,$ with an interval of 0.05. Calculate

i. $y(t) = e^{-t}\sin(\pi t)$

ii. $y(t) = \dfrac{\sin(\pi t^2)}{t^2}$

Flow Control Functions

There are many flow control functions in MATLAB. The **for** function in MATLAB provides a mechanism for repeatedly executing a series of statements a given number of times. The **for** function connected to an **end** statement sets up a repeating circulation loop. An important point is that each **for** must be matched with an **end** . The **break** statement provides exit jump out of loop.

The **while** function in MATLAB allows a mechanism for repeatedly executing a series of statements an indefinite number of times, under control of a logical condition.

The function **if** evaluates a logical expression and executes a group of statements based on the value of the expression. The **else** statement further conditionalizes the **if** statement.