9 MATLAB

9.1 Introduction

MATLAB is a high-level technical computing language and interactive environment for algorithm development, data visualization, data analysis, and numeric computation. Using the MATLAB product, we can solve technical computing problems faster than with traditional programming languages, such as C, C++ and FORTRAN

9.2 Overview of the MATLAB Environment

MATLAB is a tool for numerical computation and visualization. The basic data element is a matrix, so if we need a program that manipulates array-based data it is generally fast to write and run in MATLAB (unless we have very large arrays or lots of computations, in which case we are better offf using C or Fortran.)

We can use MATLAB in a wide range of applications, including signal and image processing, communications, control design, test and measurement, financial modeling and analysis, and computational biology. Add-on toolboxes (collections of special-purpose MATLAB functions, available separately) extend the MATLAB environment to solve particular classes of problems in these application areas.

MATLAB provides a number of features for documenting and sharing our work. We can integrate our MATLAB code with other languages and applications, and distribute our MATLAB algorithms and applications. Features include:

1. High-level language for technical computing
2. Development environment for managing code, files, and data
3. Interactive tools for iterative exploration, design, and problem solving
4. Mathematical functions for linear algebra, statistics, fourier analysis, filtering, optimization, and numerical integration
5. 2-D and 3-D graphics functions for visualizing data

6. Tools for building custom graphical user interfaces functions for integrating MATLAB based algorithms with external applications and languages, such as C, C++, Fortran, Java™, COM, and Microsoft® Excel®

9.3 The MATLAB System

The MATLAB system consists of the following main parts:

I. *Desktop Tools and Development Environment*

This part of MATLAB is the set of tools and facilities that help us use and become more productive with MATLAB functions and files. Many of these tools are graphical user interfaces. It includes: the MATLAB desktop and Command Window, an editor and debugger, a code analyzer, and browsers for viewing help, the workspace, and folders.

II. *Mathematical Function Library*

This library is a vast collection of computational algorithms ranging from elementary functions, like sum, sine, cosine, and complex arithmetic, to more sophisticated functions like matrix inverse, matrix eigenvalues, Bessel functions, and fast fourier transforms.

III. *The Language*

The MATLAB language is a high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features. It allows both "programming in the small" to rapidly create quick programs us do not intend to reuse. We can also do "programming in the large" to create complex application programs intended for reuse.

IV. *Graphics*

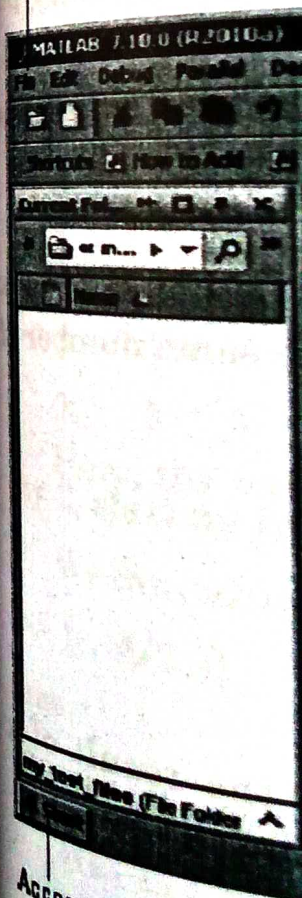
MATLAB has extensive facilities for displaying vectors and matrices as graphs, as well as annotating and printing these graphs. It includes high-level functions for two-dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics. It also includes low-level functions that allow us to fully customize the appearance of graphics as well as to build complete graphical user interfaces on our MATLAB applications.

V. External Inter
The external int
Fortran programs
facilities for calling
for calling MATLA
and writing MAT-f

VI. About the De
In general, whe
the MATLAB desk
user interfaces or
applications associ

The first time w
the default layout,

Menus change depending on the tool you are using.



Access tools and more.

9.4 Use of Matlab

When we start MATLAB, the command prompt ">>" appears. We will tell MATLAB what to do by typing commands at the prompt.

Construction of Matrices

The basic data element in MATLAB is a matrix. A scalar in MATLAB is a 1x1 matrix, and a vector is a 1xn (or nx1) matrix. The simplest way to create a matrix in MATLAB is to use the matrix constructor operator, []. Create a row in the matrix by entering elements (shown as E below) within the brackets. Separate each element with a comma or space:

row = [E1, E2, ..., Em] or, row = [E1 E2 ... Em]

For example, to create a one row matrix of five elements, type

A = [12 62 93 -8 22];

To start a new row, terminate the current row with a semicolon:

A = [row1; row2; ...; rown]

This example constructs a 3 row, 5 column (or 3-by-5) matrix of numbers. Note that all rows must have the same number of elements:

```
A = [1 6 9 -8 2; 1 2 8 4 9; -4 1 -7 9 6]
A =
   1   6   9   -8   2
   1   2   8   4   9
  -4   1  -7   9   6
```

If we don't want MATLAB to display the result of a command, put a semicolon at the end:

```
>> A = [1 1 1; 2 2 2; 3 3 3];
```

Matrix A has been created but MATLAB doesn't display it. The semicolon is necessary when we are running long scripts and don't want everything written out to the screen!

Suppose we want to access a particular element of matrix A:

```
>> A(1,2)

ans =

1
```

Suppose we want to access a particular row of A:

```
>> A(2,:)
ans =
2     2     2
```

The ":" operator we have just used generates equally spaced vectors. We can use it to specify a range of values to access in the matrix:

```
>> A(2,1:2)
ans =
2     2
```

We can also use it to create a vector:

```
>> y = 1:3
y =1  2     3
```

The default increment is 1, but we can specify the increment to be something else:

```
>> y = 1:2:6

   1     3     5
```

Here, the value of each vector element has been increased by 2, starting from 1, while less than 6.

We can easily concatenate vectors and matrices in MATLAB:

```
>> [y, A(2,:)]

ans =

   1   3   5   2   2   2
```

We can also easily delete matrix elements. Suppose we want to delete the 2nd element of the vector y:

```
>> y(2) = []
y =

      1   5
```

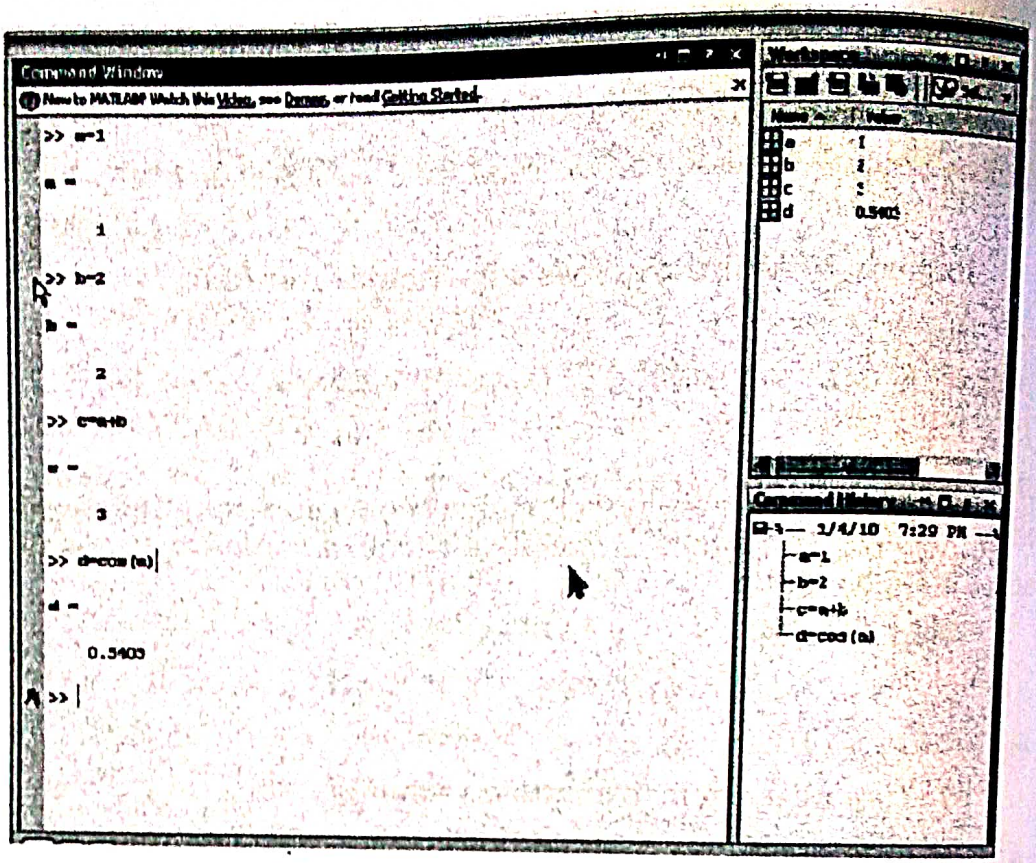


Fig2: Matlab Work space

MATLAB has several built-in matrices that can be useful. For example, zeros(n,n) makes an nxn matrix of zeros.

```
>> B = zeros(2,2)
B =
     0     0
     0     0
```

A few other useful matrices are:

zeros – *create a matrix of zeros*

ones – *create a matrix of ones*

rand – *create a matrix of random numbers*

eye – *create an identity matrix*

Matrix operations

An important thing to remember is that since MATLAB is matrix-based, the multiplication operator "*" denotes matrix multiplication. Therefore, A*B is *not* the same as multiplying each of the elements of A times the elements of B.

However, we'll probably find that at some point we want to do element-wise operations (array operations). In MATLAB we denote an array operator by playing a period in front of the operator. The difference between "*" and ".*" is demonstrated in this example:

```
>> A = [1 1 1; 2 2 2; 3 3 3];
>> B = ones(3,3);
>> A*B

ans =

   3   3   3
   6   6   6
   9   9   9

>> A.*B

ans =

   1   1   1
   2   2   2
   3   3   3
```

MATLAB provides many useful functions for working with matrices. It also has many scalar functions that will work element-wise on matrices (e.g., the function sqrt(x) will take the square root of each element of the matrix x). Below is a brief list of useful functions.

Useful matrix functions:

A' – *transpose of matrix A.*
det(A) – *determinant of A*
eig(A) – *eigenvalues and eigenvectors*
inv(A) – *inverse of A*
svd(A) – *singular value decomposition*
norm(A) – *matrix or vector norm*

find(A) – *find indices of elements that are nonzero. Can also pass an expression to this function, e.g.* find(A > 1) *finds the indices of elements of A greater than 1.*

A few useful math functions:

sqrt(x) – *square root*

sin(x) – *sine function. See also* cos(x), tan(x), *etc.*

exp(x) – *exponential*

log(x) – *natural log*

log10(x) – *common log*

abs(x) – *absolute value*

mod(x) – *modulus*

factorial(x) – *factorial function*

floor(x) – *round down. See also* ceil(x), round(x).

min(x) – *minimum elements of an array. See also* max(x).

besselj(x) – *Bessel functions of first kind*

MATLAB also has a few built-in constants, such as pi (π) and i (imaginary number).

Symbolic math

Although MATLAB is primarily used for numerical computations,we can also do symbolic math with MATLAB. Symbolic variables are created using the command "sym."

```
>> x = sym('x');
```

Here we have created the symbolic variable x. If it seems kind of lame to us to have to type in all this just to create "x", we 're in luck—MATLAB provides a shortcut.

```
>> syms x
```

This is a shortcut for x = sym('x').

Symbolic variables can be used for solving algebraic equations. For example, suppose we want to solve the equation "x^4 + 3*x^2 + 3 = 5":

Below are a few useful things we can do with symbolic variables in MATLAB. For more, look at the "Symbolic Math Toolbox" section of the MATLAB help.

solve – *symbolic solution of systems of algebraic equations*
int(f,x) – *indefinite integral of f with respect to x*
int(f,x,a,b) – *definite integral of f with respect to x from a to b.*
diff(f,'x') – *differentiate f with respect to x*
taylor – *Taylor series expansion of symbolic expression*
subs – *substitute values into a symbolic expression*

9.5. M-files : Scripts and Functions

Matlab can also be used as a programming language. To program in Matlab we simply create a text file containing Matlab commands exactly as we would type them interactively in the Matlab window. The file may have any legal unix name, and should end with a .m extension. These files may be placed in our root directory, or a directory named /matlab. (Any other directories would have to be explicitly added to our Matlabpath.) There are two types of m-files in matlab. One is called a script. This is simply a list of Matlab commands with no header. The other type is a function. Functions have a header line that may look something like:

```
function
    y=fun1(x)
```

Functions may be passed arguments, and may return results. To invoke a script or a function we simply type the filename (without the .m extension) into the Matlab window. We will also notice that m-files which we create are included in the help listing. If we perform a help on a specific m-file, help will return any comments which appear

Function Functions

before the first line of actual code in the m-file.

We can also use m-files to create our own functions. For example, suppose we want to make a function that increments the value of each element of a matrix by some constant. And suppose we want to call the function "incrementor." We would make an m-file called "incrementor.m" containing the following:

```
function f = incrementor(x,c)
% Incrementor adds c to each element in the matrix x. f=x+c;
```

9.6 . Making plots

Now that we can load data into MATLAB and easily manipulate it, we'll want to be able to display the results in a meaningful (and hopefully aesthetically pleasing) way. Let's create some sample data for an example of a simple linear plot:

```
>> x = 1:10;
>> y1 = x.^2;
>> y2 = 2.*y1;
```

We can plot it using the "plot" command:

```
>> plot(x,y1)
```

Suppose we want to plot only the data points, without the line between them:

```
>> plot(x,y1,'ko')
```

This will plot the data with black o's ("k" for the color black, and "o" to plot o's) instead of the default blue line. To see all the options for plotting colors/characters, type "help plot".

However, if we are just making a plot from the command line, it may be easier to make changes directly in the figure window (for example, to change the y-limits go to the "Edit" menu and choose "Axes properties"). If we are planning on becoming an advanced user of MATLAB, it would be worthwhile to learn about how handles work in MATLAB. Check out the help sections on "get", "set", "gca", and "gcf".

Suppose we want to make a second plot, of y2. If we type "plot(x,y2)" it will overwrite the plot that we already have. If we want to be able to look at both plots at once, we can plot y2 in a new window. Type "figure" to open a new figure window:

```
>> figure
>> plot(x,y2)
```

However, we may want to plot y1 and y2 on the same set of axes. Use the "hold" command to hold the current plot and axes properties. First, return to figure 1:

```
>> figure(1)
>> hold on
>> plot(x,y2)
```

We can put multiple individual plots in the same figure window using the "subplot" command. Type "help subplot" for more information. Once we're done with our plot, we'll probably want to label the axes:

```
>> xlabel('x')
>> ylabel('y')
```

We can also give it a title:

```
>> title('My plot')
```

9.7 Advanced operations

There's a lot more that we can do with MATLAB than is listed in this handout. Check out the MATLAB help or one of the "Other Resources" if we want to learn more about the following more advanced tools:

- Numerical integration (quad)
- Discrete Fourier transform (fft, ifft)
- Statistics (mean, median, std, var)
- Curve fitting (cftool)
- Signal processing (sptool)
- Numerical integration of systems of ODEs (ode45)

Illustrative Examples

Example 1. Some problems on matrices

First let us write the matrix A = [1 2 0; 2 5 -1; 4 10 -1] as

A =

```
    1    2    0
    2    5   -1
    4   10   -1
```

Then we can easily find the transpose of the matrix A i.e., B = A'

Thus

B =

```
    1    2    4
    2    5   10
    0   -1   -1
```

Now let's multiply these two matrices together. Note again that MATLAB doesn't require us to deal with matrices as a collection of numbers. MATLAB knows when we are dealing with matrices and adjusts our calculations accordingly. C = A * B

C =

```
    5   12   24
   12   30   59
   24   59  117
```

Instead of doing a matrix multiply, we can multiply the corresponding elements of two matrices or vectors using the .* operator. C = A .* B

C =

```
    1    4    0
    4   25  -10
    0  -10    1
```

Let's find the inverse of a matrix A i.e., X = inv(A)

X =

```
    5    2   -2
   -2   -1    1
    0   -2    1
```

and then we verify the fact that a matrix times its inverse is the identity matrix i.e., I = inv(A) * A.

```
I =
     1     0     0
     0     1     0
     0     0     1
```

MATLAB has functions for nearly every type of common matrix calculation. There are functions to obtain eigenvalues of A i.e., eig(A)

```
ans =
    3.7321
    0.2679
    1.0000
```

as well as the singular value decomposition . svd(A)

```
ans =
   12.3171
    0.5149
    0.1577
```

The "poly" function generates a vector containing the coefficients of the characteristic polynomial. The characteristic polynomial of a matrix A is p = round(poly(A))

```
p =
     1    -5     5    -1
```

We can easily find the roots of a polynomial using the roots function. These are actually the eigenvalues of the original matrix. roots(p)

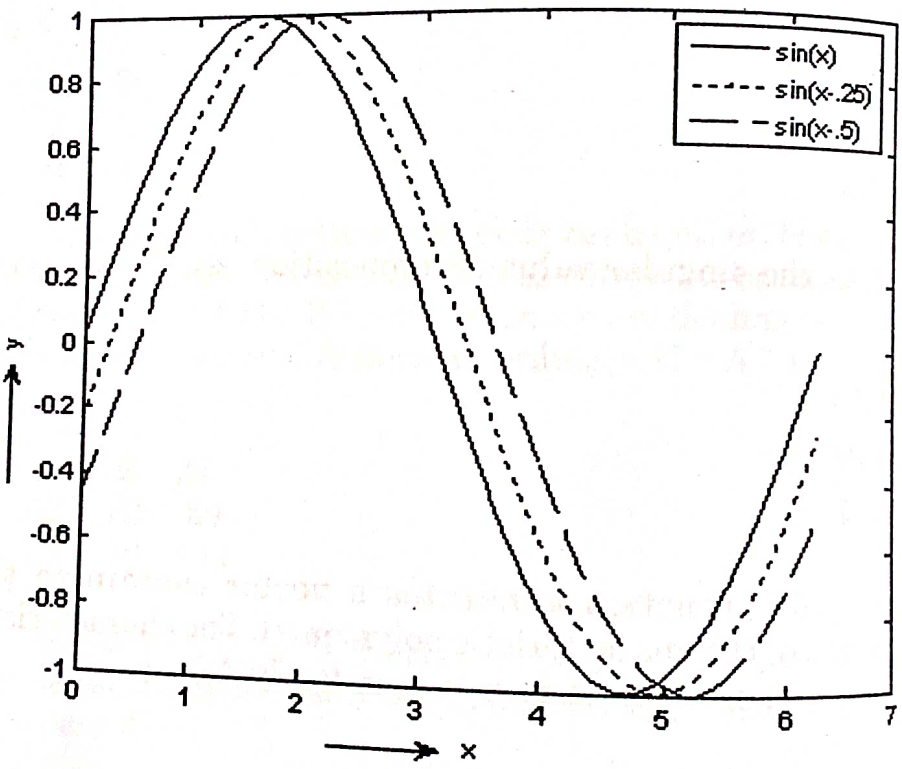
```
ans =
    3.7321
    1.0000
    0.2679
```

Example 2. Graph of sin (x), sin(x-.25), sin(x-.5);

Programme

```
x = 0:pi/100:2*pi;
y = sin(x);
y2 = sin(x-.25);
y3 = sin(x-.5);
plot(x,y,x,y2,x,y3)
legend('sin(x)','sin(x-.25)','sin(x-.5)')
```

The output is

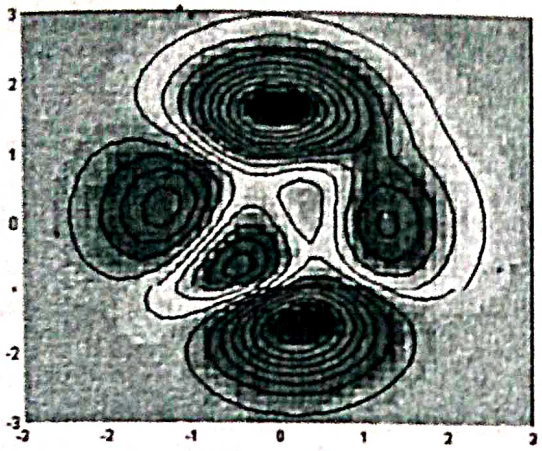


Example 3. A contour plot of the peaks function

Programme

```
[x,y,z] = peaks;
pcolor(x,y,z)
shading interp
hold on
contour(x,y,z,20,'k')
hold off
```

The output is



Example 4. Three dimensional figure of sin (R)/R, where $R = (x^2+y^2)^{1/2}$

Programme:

```
[X,Y] = meshgrid(-8:.5:8);
R = sqrt(X.^2 + Y.^2) + eps;
Z = sin(R)./R;
mesh(X,Y,Z,'EdgeColor','black')
```

The output is