

# Appendix A:

## Introduction to MATLAB

In this Appendix we introduce the MATLAB features that are essential to understanding the basics of the software developed throughout this text. Several good MATLAB self-study tutorial books are available; see Hanselman and Littlefield (2005), Hahn (2002), and Lyshevski (2003). Also, the manuals that accompany MATLAB are recommended as additional references. The MATLAB program has extensive built-in help and tutorials.

We strongly recommend that the readers go through this appendix and practice each command while sitting at a computer running MATLAB. This exercise may fail to convey its intended lessons if the user is not practicing these commands at the computer. The symbol “>>” is MATLAB’s prompt, which automatically appears in the command window. You do not need to type it. In this Appendix, and throughout this book, any line that starts with the symbol “>>” is a Command Window line.

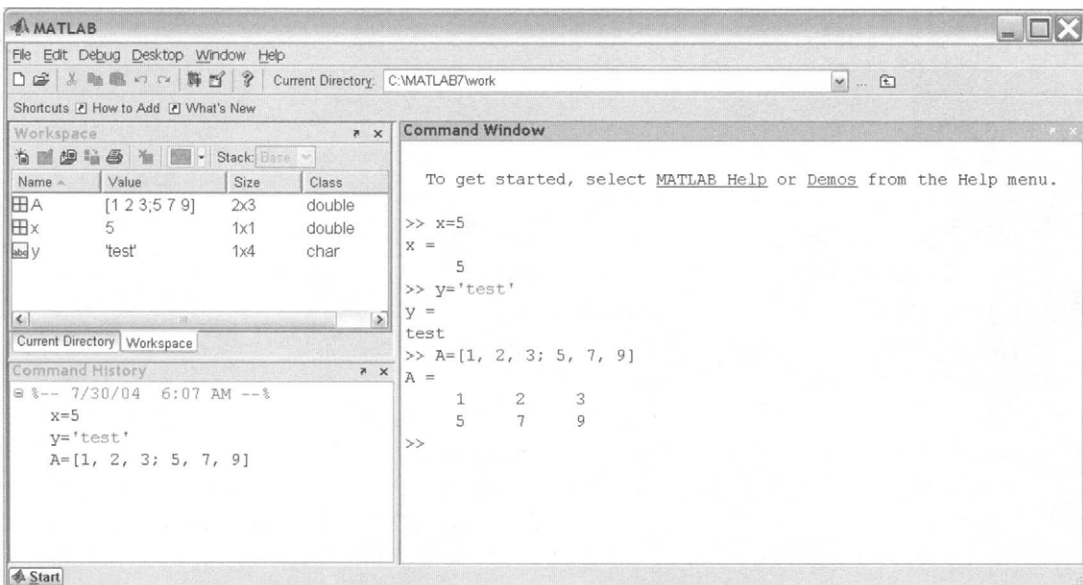
## A.1 The MATLAB Environment

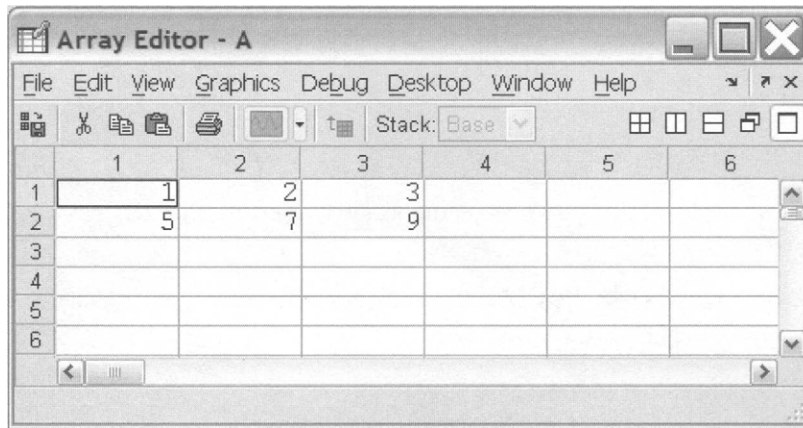
The default working environment in MATLAB version 7.0 (R14) displays the four panels shown in Fig. A.1. These are:

**Workspace:** This browser displays the variables created and used during a MATLAB session. Variables are added to the workspace by defining them in the command window, using functions, running M-files (MATLAB language programs), and/or loading saved workspaces. To view the workspace and information about each variable, use the workspace browser, or use the commands `who` and `whos` in the Command Window. To view and/or edit any variable, double-click on the variable of interest in the workspace browser. This will open the Array Editor (see Fig. A.2) to show you the contents of that variable and enable you to edit it.

**Current Directory** (not open in Fig. A.1): This panel displays the contents of the current directory.

**Command History:** This panel displays all the commands that have been entered in the present and previous MATLAB sessions. Point the mouse on any of these commands and click the right button to see the options available: Copy, Evaluate Selection, Create M-file, and Delete. Any of the commands in the Command History window may be executed again by dragging the command from the Command History and dropping it on the Command Window.



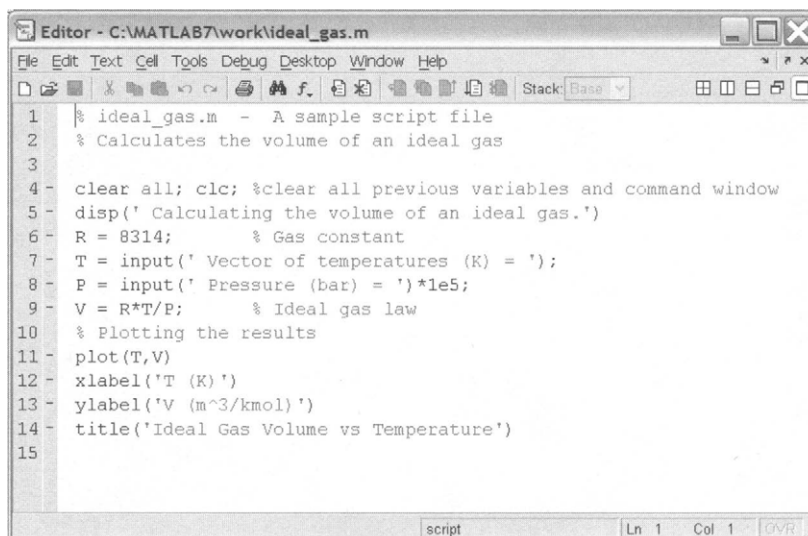


**Figure A.2** The Array Editor, showing the values of array A.

**Command Window:** This is the window in which all MATLAB commands are entered and results are displayed. It is used to enter variables, execute functions, and run M-files.

In addition to the above, the following windows may be opened:

**Editor:** The MATLAB Editor (see Fig. A.3) is used to create, edit, debug, and execute MATLAB programs (M-files). The command `edit` in the Command Window opens the MATLAB Editor.



**Figure A.3** The MATLAB Editor.

**Profiler:** MATLAB includes the Profiler to help you improve the performance of your M-files. Run a MATLAB statement or an M-file in the Profiler and it produces a report of where the time is being spent. Access the Profiler from the Desktop menu, or use the `profile` function.

**Start button:** The MATLAB Start button, located at the lower left corner of the MATLAB window (Fig. A.1), has a menu interface that provides easy access to all of the above items (and more).

### A.1.1 Customizing the MATLAB environment

The MATLAB environment may be customized to suit the user's needs. To do so, go to the menu item File/Preferences. This opens the window shown on Fig. A.4, which gives the user a large number of options from adjusting the size and color of fonts to specifying the format used in displaying the numbers and the compactness of the display.

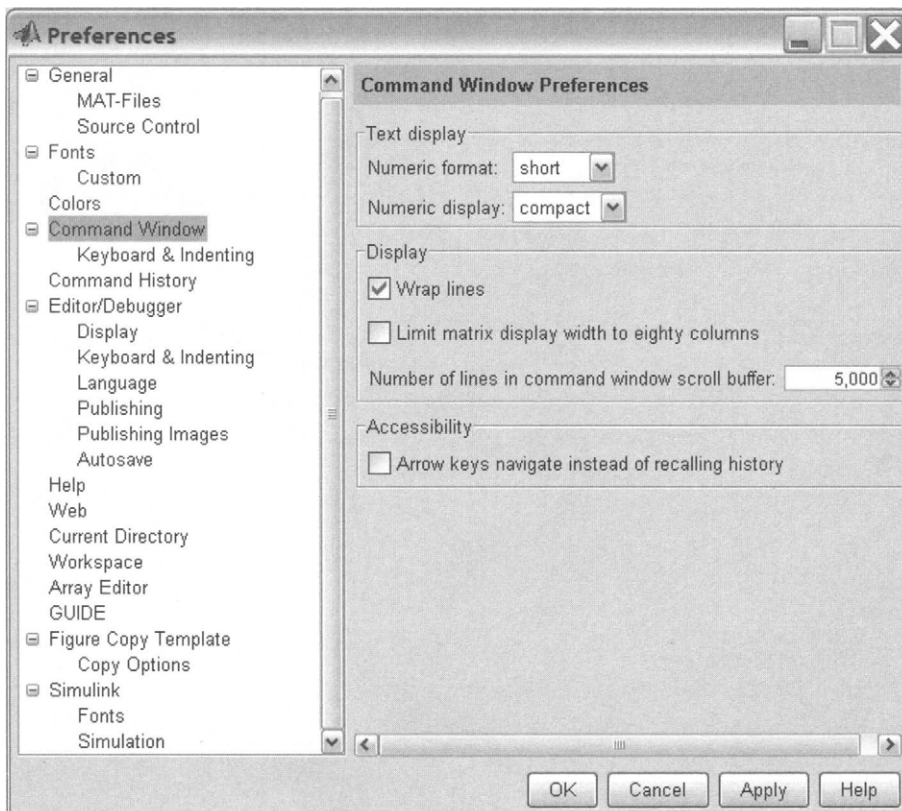


Figure A.4 Preferences for customizing the MATLAB environment.

### A.1.2 The MATLAB path

When a command is executed or the name of an M-file is invoked in the Command Window, MATLAB searches through a list of folders and subfolders to locate that command or M-file, in order to execute it. The standard MATLAB search path is set when MATLAB is installed. Users can add additional folders that contain their own files. To add to the path, go to File/Set Path and follow instructions, as shown on Fig. A.5.

Do **not** remove any of the standard MATLAB search path folders. These are needed for proper operation of MATLAB.

Caution should be exercised in naming new M-files. If a file is named with an identical name as one already existing in MATLAB, the file that is found first during the search of the path will be executed. It is good practice to avoid duplicating names, and to add the users' folders to the bottom of the path.

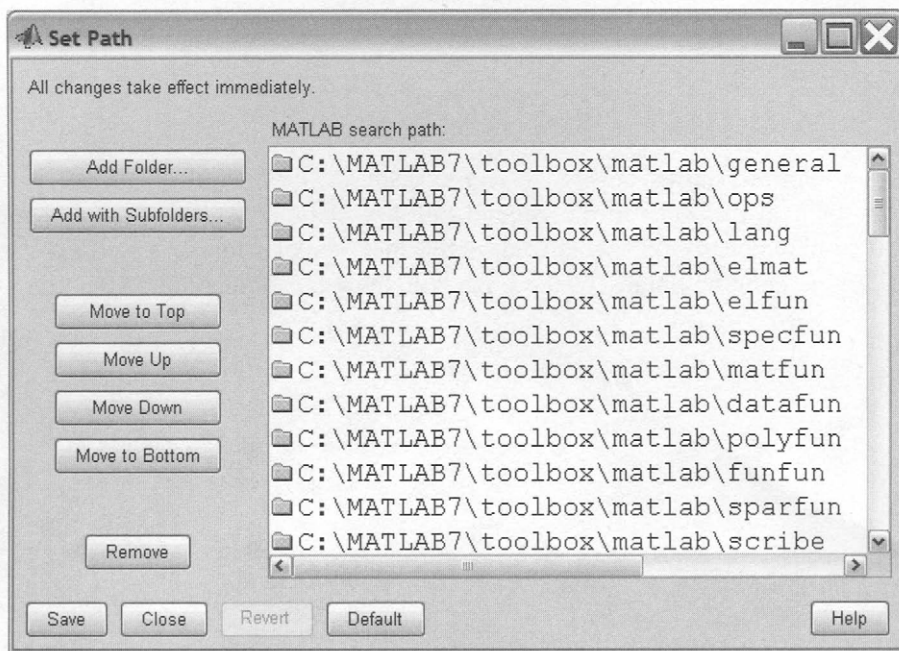


Figure A.5 Changing the MATLAB path.

### A.1.3 Where to find help for MATLAB

As a beginner, you may want to see a tutorial about MATLAB. Typing `demo` at the command line will display the available demonstrations. In the MATLAB demo window (Fig. A.6), you may choose the subject you are interested in and then follow the lessons.

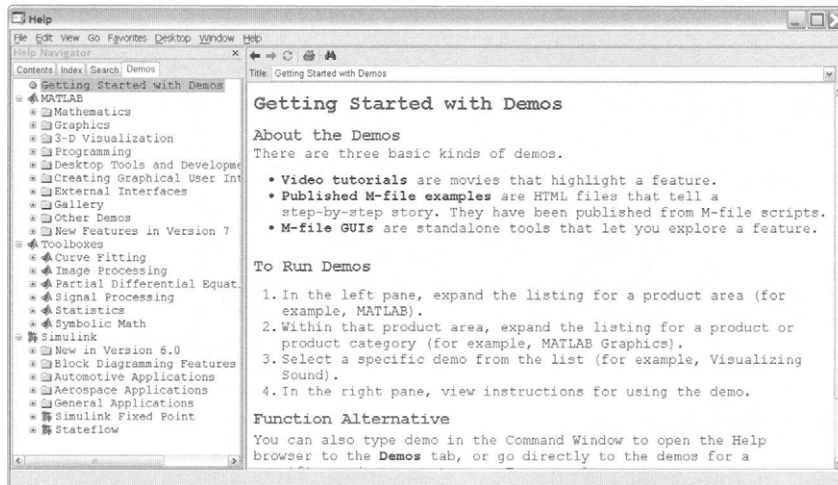


Figure A.6 The MATLAB demo window.

The command

```
>> helpwin
```

opens the window shown on Fig. A.7, and displays the overall organization of all available help. It has links to the general purpose commands; operators and special characters; elementary and specialized math functions; and matrix functions, among others. Single clicking on any of the links will show the available help for the given topic.

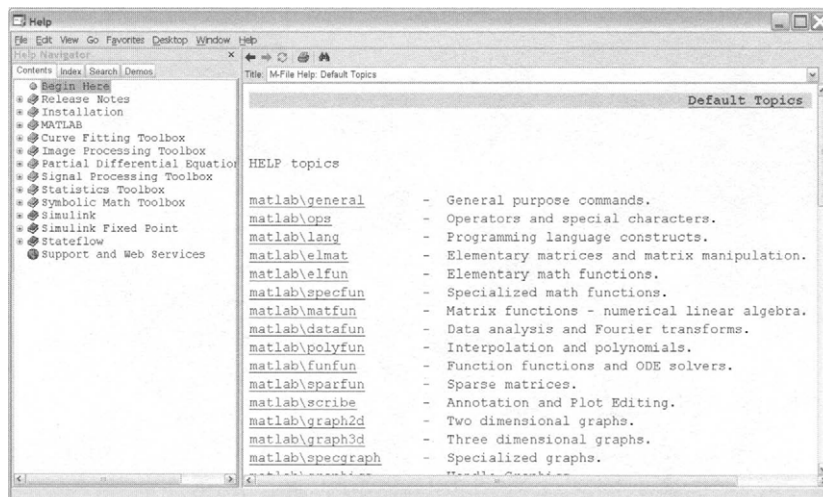


Figure A.7 The default help page.

Typing `help` alone lists the names of all directories in the MATLAB search path with a one-line description for each. Also, if you type a directory name following the command `help`, MATLAB lists the contents of the directory (if the directory is one in the MATLAB path search and contains a `contents.m` file). Try the following commands:

```
>> help
>> help general
>> helpdesk
>> doc
```

If you know the name of the function for which you need help, you can use the `help` command. For example, to get help for the command `sign`:

```
>> help sign

SIGN    Signum function.
        For each element of X, SIGN(X) returns 1 if the element
        is greater than zero, 0 if it equals zero and -1 if it is
        less than zero. For the nonzero elements of complex X,
        SIGN(X) = X ./ ABS(X).
```

The lines printed above are in the first block of comment lines in this function. These are designed to give a brief explanation of the function and its use. The percent symbol (%) is used to mark a line that contains comments:

```
% This is a comment statement in a program
```

If you are not sure of the function name, you can try to find the name using the `lookfor` command:

```
>> lookfor absolute
ABS      Absolute value.
IMABSDIFF Compute absolute difference of two images.
CIRCLEPICK Pick bad triangles using an absolute tolerance
MAD Mean/median absolute deviation.
ABS      Absolute value.
LOCATEFILE Resolve a filename to an absolute location.
applyabsolute.m: % out =
applyabsolute(in,cspace,source_wp,dest_wp)
```

Extensive MATLAB help and manuals may be found on the following Web sites:

<http://www.mathworks.com>  
<http://www.mathworks.com/academia/>

## A.2 Elementary Operations

The four elementary arithmetic operations in MATLAB—addition, subtraction, multiplication, and division—are represented by the operators `+`, `-`, `*`, and `/`, respectively:

```
>> (20+10-4*5)/2
ans =
    5
```

The symbol `^` stands for the power operator:

```
>> 5^2
ans =
   25
```

The operator `\` is for left division, where the denominator is at the left, rather than the right. Compare `/` and `\`:

```
>> 2/4
ans =
    0.5000
>> 2\4
ans =
    2
```

MATLAB can handle complex numbers

```
>> sqrt(-1),
ans =
    0 + 1.0000i
```

and infinite numbers

```
>> 1/0
Warning: Divide by zero.
(Type "warning off MATLAB:divideByZero" to suppress this warning.)
ans =
    Inf
```

The letters `i` and `j` represent the complex number  $\sqrt{-1}$  unless another value is assigned to them. Also, the variable `pi` represents the ratio of the circumference of a circle to its diameter (i.e., 3.141592653. . .). If an expression cannot be evaluated, MATLAB returns `NaN`, which stands for Not-a-Number:

```
>> 0*log(0)
Warning: Log of zero.
ans =
    NaN
```



The equality sign is used to represent the assignment of values to variables:

```
>> a = 2
a =
    2
>> b=3*a
b =
    6
```

If there is no assignment, then the result of the expression is saved in a variable named `ans`:

```
>> a+b
ans =
    8
```

If you want to suppress the display in the Command Window of the result of the assignment, put a semicolon at the end of the statement:

```
>>c = a+b;
```

You can see the value of the variable by simply typing the variable:

```
>> c
c =
    8
```

MATLAB is case sensitive, meaning that MATLAB distinguishes between variables with upper- and lower-case names:

```
>> A=10;
>> A
A =
    10
>> a
a =
    2
```

All computations in MATLAB are done in double precision (see Chapter 3). The precision of the calculation is independent of the formatted display. The results of calculation are normally displayed, or printed, with only five significant digits. The `format` command may be used to switch between different display formats:

```
>> d = exp(pi)
d =
    23.1407
>> format long, d
d =
    23.14069263277927
```

```
>> format short e, d
d =
    2.3141e+001
>> format long e, d
d =
    2.314069263277927e+001
>> format short, d
d =
    23.1407
```

All of the different variations of the `format` command may be viewed by getting the help for `format`:

```
>> help format
```

```
FORMAT Set output format.
```

```
All computations in MATLAB are done in double precision.
```

```
FORMAT may be used to switch between different output
display formats as follows:
```

```
FORMAT          Default. Same as SHORT.
FORMAT SHORT    Scaled fixed point format with 5 digits.
FORMAT LONG     Scaled fixed point format with 15 digits.
FORMAT SHORT E  Floating point format with 5 digits.
FORMAT LONG E   Floating point format with 15 digits.
FORMAT SHORT G  Best of fixed or floating point format with 5
                digits.
FORMAT LONG G   Best of fixed or floating point format with
                15 digits.
FORMAT HEX      Hexadecimal format.
FORMAT +        The symbols +, - and blank are printed
                for positive, negative and zero elements.
                Imaginary parts are ignored.
FORMAT BANK     Fixed format for dollars and cents.
FORMAT RAT      Approximation by ratio of small integers.
```

```
Spacing:
```

```
FORMAT COMPACT Suppress extra line-feeds.
```

```
FORMAT LOOSE   Puts the extra line-feeds back in.
```

In order to delete a variable from the memory, use the `clear` command:

```
>>clear a
```

Using `clear` or `clear all` command deletes all the variables from the workspace:

```
>> clear          or          >> clear all
```

Check to verify that all variables have been deleted from the workspace by looking at the Workspace window.

The `clc` command clears the Command Window and brings the cursor to the top of the window (home):

```
>> clc
```

The `clf` command clears the current figure:

```
>> clf
```

Remember that by using the up-arrow (or down-arrow) key you can scroll backwards (or forwards) through all the commands you have entered so far in each session. If you need to recall a specific command that has been used already, just type its first letter (or first few letters) and then use the up arrow key to locate that command.

Several operating system navigational commands may be executed from the Command Window, such as:

```
cd (for change directory),  
dir (for list the files in directory),  
mkdir (for make new directory),  
pwd (for print current working directory),  
ls (for list directory).
```

For example, to change to the work directory of MATLAB:

```
>> cd c:\matlab704\work
```

To change to Chapter1 directory of the Biosystems program directory:

```
>> cd 'C:\Program Files\Biosystems\Chapter1'
```

The single quotation mark ( ' ) is needed in the above command because of the presence of blank spaces in the name of the directory. For this reason, the student is strongly advised not to place blanks in names of directories and in the names of M-files. The underscore character ( \_ ) should be used to separate words in names of files.

## A.3 Vectors and Matrices

The word MATLAB is an abbreviation for Matrix Laboratory: MATLAB was designed to make operations on matrices as easy as possible. All variables in MATLAB are represented as arrays. A scalar number is a  $(1 \times 1)$  array, a row vector is a  $(1 \times n)$  array, and a column vector is a  $(n \times 1)$  array. The standard matrix notation  $M(\text{row}, \text{column})$  is used by MATLAB for two-dimensional arrays. Multidimensional matrices (i.e., matrices with more than two dimensions) may also be used in

MATLAB. The notation used is  $M(\text{row}, \text{column}, \text{page}, \dots)$ . The third dimension is called *page*, but no generic names are given to the higher dimensions.

Creating a matrix is also done by assigning values to a MATLAB variable, where elements of the matrix must be enclosed in square brackets:

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

or

```
>> M = [1, 2, 3
        4 5 6]
M =
     1     2     3
     4     5     6
```

The elements of a row may be separated either by a comma or a space, and the rows may be separated by a semicolon or carriage return (i.e., a new line). Elements of a matrix can be accessed or replaced individually, as in  $M(\text{row}, \text{column})$ :

```
>> M(1,3)
ans =
     3

>> M(2,1) = 7
M =
     1     2     3
     7     5     6
```

The `row` and `column` variables are used for indexing the position of an element within the matrix. Array indices must be integer numbers greater than or equal to 1. Zero is not acceptable as an index for arrays in MATLAB.

Matrices may be combined together to form new matrices:

```
>> N = [M; M]
N =
     1     2     3
     7     5     6
     1     2     3
     7     5     6

>> O = [M, M]
O =
     1     2     3     1     2     3
     7     5     6     7     5     6
```

The transposition of a matrix results from interchanging its rows and columns. This can be done by putting a single quote after a matrix:

```
>> M_trans=M'
M_trans =
     1     7
     2     5
     3     6
```

A convenient shorthand notation for a sequence is the colon (:) operator. A colon placed between two numbers generates the vector of numbers from the left limit to the right limit:

```
>> v = [-1:4]
v =
    -1     0     1     2     3     4
```

The default increment is 1, but the user can change it, if required:

```
W = [-1:0.5:2; 6:-1:0; 1:7]
W =
   -1.0000   -0.5000     0    0.5000    1.0000    1.5000    2.0000
    6.0000    5.0000    4.0000    3.0000    2.0000    1.0000     0
    1.0000    2.0000    3.0000    4.0000    5.0000    6.0000    7.0000
```

A very common use of the colon notation is to refer to rows, columns, or a part of the matrix:

```
>> W(:,5) % This command refers to all rows of the 5th column
ans =
     1
     2
     5

>> W(1,:) % This refers to the 1st row and all the columns
ans =
   -1.0000   -0.5000     0    0.5000    1.0000    1.5000    2.0000

>> W(2:3,4:7) % This refers to rows 2 to 3 and column 4 to 7
ans =
     3     2     1     0
     4     5     6     7

>> W(2,5:end) % This refers to row 2 and columns 5 to last column
ans =
     2     1     0
```

Multidimensional matrices (i.e., matrices with more than two dimensions) may also be handled in MATLAB. Let us add a third dimension to the matrix `W` to make it into `W(row, column, page)`:

```
>> W(:,:,2) = [2:2:14; 0.1 0.2 0.3 0.4 0.5 0.6 0.7; 2 2 2 2 2 2 2]
W(:,:,1) =
    -1.0000    -0.5000         0     0.5000     1.0000     1.5000     2.0000
     6.0000     5.0000     4.0000     3.0000     2.0000     1.0000         0
     1.0000     2.0000     3.0000     4.0000     5.0000     6.0000     7.0000
W(:,:,2) =
     2.0000     4.0000     6.0000     8.0000    10.0000    12.0000    14.0000
     0.1000     0.2000     0.3000     0.4000     0.5000     0.6000     0.7000
     2.0000     2.0000     2.0000     2.0000     2.0000     2.0000     2.0000
```

As mentioned earlier, the third dimension is called *page*.

### A.3.1 MATLAB construction functions for special arrays

MATLAB has many built-in array construction functions. For example:

```
>>ones(2)                % generates a (2 × 2) matrix of ones
ans =
     1     1
     1     1

>>ones(2,3)              % generates a (2 × 3) matrix of ones
ans =
     1     1     1
     1     1     1

>>zeros(2,3,2)           % generates a (2 × 3 × 2) array of zeros
ans(:,:,1) =
     0     0     0
     0     0     0
ans(:,:,2) =
     0     0     0
     0     0     0

>>eye(3)                 % generates a (3 × 3) identity matrix
ans =
     1     0     0
     0     1     0
     0     0     1

>>rand(4,2)              % generates a (4 × 2) matrix of random
                        % variables uniformly distributed on the
                        % interval (0.0,1.0)
ans =
     0.9501     0.8913
     0.2311     0.7621
     0.6068     0.4565
     0.4860     0.0185

>>linspace(-1,5,7)       % generates a 7-element row vector of
                        % equally spaced numbers between -1 and 5
ans =
    -1         0         1         2         3         4         5
```

```
>>logspace(-1,2,8)      % generates an 8-element row vector of
                        % logarithmically equally spaced points
                        % between  $10^{-1}$  and  $10^2$ 

ans =
    0.1000    0.2683    0.7197    1.9307    5.1795   13.8950
   37.2759  100.0000
```

Two very useful array functions are `size`, which gives the dimensions of the array, and `length`, which gives the maximum length of the array:

```
>> size(W)
ans =
     3     7     2
>> length(W)
ans =
     7
```

### A.3.2 Array arithmetic

Multiplying an array by a scalar multiplies each element of the array by the scalar:

```
>> A= [1, 2, 6; 2:4; 10:0.5:11], 2*A
A =
    1.0000    2.0000    6.0000
    2.0000    3.0000    4.0000
   10.0000   10.5000   11.0000
ans =
     2     4    12
     4     6     8
    20    21    22
```

Only arrays of the same size may be added or subtracted:

```
>> B = [4, 7, 14; 1, 5, 7; 7, 7, 5]
B =
     4     7    14
     1     5     7
     7     7     5
ans =
>> A-B
   -3.0000   -5.0000   -8.0000
    1.0000   -2.0000   -3.0000
    3.0000    3.5000    6.0000

>> c = [5; 2; 4];
>> A+c
??? Error using ==> +
Matrix dimensions must agree.
```

Adding a scalar to an array results in adding the scalar to each element of the array:

```
>> A+5
ans =
    6.0000    7.0000   11.0000
    7.0000    8.0000    9.0000
   15.0000   15.5000   16.0000
```

Vector and matrix multiplication requires that they are conformable (see Appendix C):

```
>> A*B      % conformable because both matrices are (3 × 3)
ans =
   48.0000   59.0000   58.0000
   39.0000   57.0000   69.0000
  127.5000  199.5000  268.5000

>> A*c      % conformable because A is (3 × 3) and c is (3 × 1)
ans =
    33
    32
   115

>> c*A      % nonconformable because c is (3 × 1) and A is (3 × 3)
??? Error using ==> *
Inner matrix dimensions must agree.

>> c'*A     % conformable because c' is (1 × 3) and A is (3 × 3)
ans =
    49    58    82
```

To perform element-by-element operations on matrices and vectors, use a period (.) before the operator symbol:

```
>> A.*B
ans =
    4.0000   14.0000   84.0000
    2.0000   15.0000   28.0000
   70.0000   73.5000   55.0000

>> A.^2
ans =
    1.0000    4.0000   36.0000
    4.0000    9.0000   16.0000
  100.0000  110.2500  121.0000
```



```

>> A^2          % compare this with A.^2 above
ans =
    65     71     80
    48     55     68
   141    167    223

>> 1./A
ans =
    1.0000    0.5000    0.1667
    0.5000    0.3333    0.2500
    0.1000    0.0952    0.0909

```

Some useful matrix functions are:

```

>> det(A)        % determinant of a square matrix
ans =
   -27

>> inv(A)         % inverse of a square matrix
ans =
    0.3333   -1.5185    0.3704
   -0.6667    1.8148   -0.2963
    0.3333   -0.3519    0.0370

>> rank(A)        % rank of a matrix (see Appendix C)
ans =
     3

>> [V,D]=eig(A)   % eigenvectors and eigenvalues of a square matrix
V =
   -0.3457   -0.7603    0.6014
   -0.2817   -0.1527   -0.7737
   -0.8951    0.6314    0.1993
D =
   18.1660         0         0
         0   -3.5811         0
         0         0    0.4150

```

The columns of matrix **V** above are the eigenvectors of **A**, and the diagonal elements of matrix **D** are the corresponding eigenvalues of **A**.

```

>> poly(A)        % coefficients of characteristic polynomial of A
ans =
    1.0000   -15.0000   -59.0000    27.0000

```

The above result indicates that the characteristic polynomial,  $\det(\mathbf{A} - \lambda \mathbf{I})$ , is

$$\lambda^3 - 15\lambda^2 - 59\lambda + 27$$

## A.4 MATLAB Built-in Functions

MATLAB provides standard elementary and advanced mathematical and visualization functions, such as `abs`, `sqrt`, `exp`, `sin`, `cos`, `eig`, `plot`, `plot3`.

To see a listing of all the elementary functions, give the command:

```
>> help elfun
Elementary math functions.

Trigonometric.
sin           - Sine.
sind          - Sine of argument in degrees.
sinh          - Hyperbolic sine.
asin          - Inverse sine.
asind         - Inverse sine, result in degrees.
asinh         - Inverse hyperbolic sine.
cos           - Cosine.
cosd          - Cosine of argument in degrees.
cosh          - Hyperbolic cosine.
acos          - Inverse cosine.
acod          - Inverse cosine, result in degrees.
acosh         - Inverse hyperbolic cosine.
tan           - Tangent.
tand          - Tangent of argument in degrees.
tanh          - Hyperbolic tangent.
atan          - Inverse tangent.
atand         - Inverse tangent, result in degrees.
atan2         - Four quadrant inverse tangent.
atanh         - Inverse hyperbolic tangent.
sec           - Secant.
secd          - Secant of argument in degrees.
sech          - Hyperbolic secant.
asec          - Inverse secant.
asecd         - Inverse secant, result in degrees.
asech         - Inverse hyperbolic secant.
csc           - Cosecant.
cscd          - Cosecant of argument in degrees.
csch          - Hyperbolic cosecant.
acsc          - Inverse cosecant.
acscd         - Inverse cosecant, result in degrees.
acsch         - Inverse hyperbolic cosecant.
cot           - Cotangent.
cotd          - Cotangent of argument in degrees.
coth          - Hyperbolic cotangent.
acot          - Inverse cotangent.
acotd         - Inverse cotangent, result in degrees.
acoth         - Inverse hyperbolic cotangent.
```

## Exponential.

exp	- Exponential.
expm1	- Compute $\exp(x)-1$ accurately.
log	- Natural logarithm.
loglp	- Compute $\log(1+x)$ accurately.
log10	- Common (base 10) logarithm.
log2	- Base 2 logarithm and dissect floating point number.
pow2	- Base 2 power and scale floating point number.
realpow	- Power that will error out on complex result.
reallog	- Natural logarithm of real number.
realsqrt	- Square root of number greater than or equal to zero.
sqrt	- Square root.
nthroot	- Real n-th root of real numbers.
nextpow2	- Next higher power of 2.

## Complex.

abs	- Absolute value.
angle	- Phase angle.
complex	- Construct complex data from real and imaginary parts.
conj	- Complex conjugate.
imag	- Complex imaginary part.
real	- Complex real part.
unwrap	- Unwrap phase angle.
isreal	- True for real array.
cplxpair	- Sort numbers into complex conjugate pairs.

## Rounding and remainder.

fix	- Round towards zero.
floor	- Round towards minus infinity.
ceil	- Round towards plus infinity.
round	- Round towards nearest integer.
mod	- Modulus (signed remainder after division).
rem	- Remainder after division.
sign	- Signum.

To obtain a listing of the specialized math functions, type the command:

```
>> help specfun
```

For a listing of elementary matrices and matrix manipulation functions, type:

```
>> help elmat
```

For a listing of the graphics functions, give the command:

```
>> help graphics
```

For a listing of all the operators and special characters, give the command:

```
>> help ops
```

For a help with any of the Toolboxes give the command `help` followed by the name of the Toolbox:

```
>> help symbolic math
```

or

```
>> help simulink
```

## A.5 Graphics

MATLAB has a variety of visualization tools: 2-D graphics, 3-D graphics, pie charts, bar charts, histograms, and contour plots that may be used to represent data and functions.

### 2-D graphs

Functions of one independent variable can easily be visualized in MATLAB (see Fig. A.8):

```
>> x = linspace(0,2,30);      % create a vector of 30 values
                                % equally spaced between 0 and 2
>> y = x.*exp(-x);           % calculate the y values
                                % corresponding to the vector x
>> plot(x,y)                  % plot y versus x
>> grid                       % add grid lines to the current axes
>> xlabel('Distance')         % label the x-axis
>> ylabel('Concentration')     % label the y-axis
>> title('Figure 1: y = xe^-x') % add the title of the graph
>> gtext('anywhere')          % place text with mouse
>> text(1,0.2,'(1,0.2)')      % place text at the specific point
```

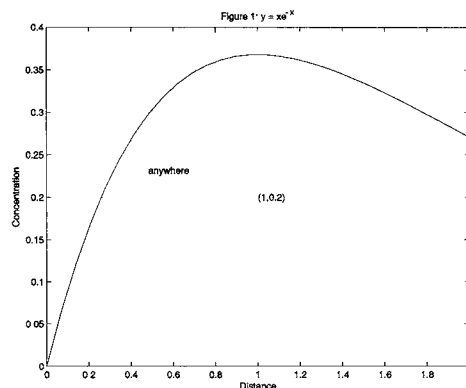
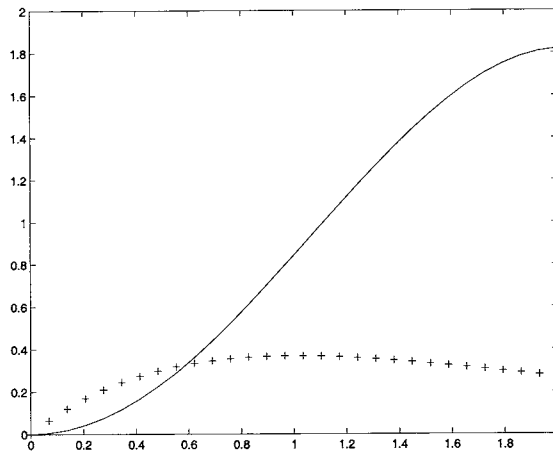


Figure A.8 Sample of 2-D graph.

You can use symbols instead of lines. You can also plot more than one function on the same figure (see Fig. A.9):

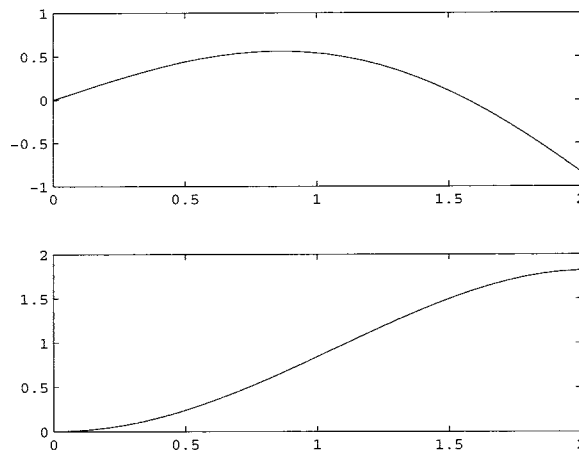
```
>> plot(x, y, '+', x, x.*sin(x))
```



**Figure A.9** Multiple functions plotted on the same figure.

If desired, more than one graph may be shown in different frames of the same figure. The following commands create a  $(2 \times 1)$  array of frames and place the first plot in frame 1 and the second in frame 2 (see Fig. A.10):

```
>> subplot(2,1,1), plot(x,x.*cos(x))  
>> subplot(2,1,2), plot(x,x.*sin(x))
```



**Figure A.10** Multiple graphs on the same figure.

Axis limits can be seen and modified using the `axis` command:

```
>> axis
ans =
     0     2     0     2
```

Try the following command and observe what happens to the plot:

```
>> axis([0, 1.5, 0, 1.5])
```

Before continuing, clear the graphics window:

```
>> clf
```

Now let us see the comet-like trajectory of the function:

```
>> shg, comet(x,y)
```

The `shg` command shows the current graphics window. It is possible to use more than one graphics window by using the `figure(n)` command, where `n` is a positive integer.

Another easy way to plot a function is to use `fplot(FUN, LIMS)` that plots the function `FUN` between the x-axis limits `LIMS = [XMIN XMAX]`:

```
>> figure(2) % creates a second figure window
>> fplot('x*exp(-x)', [0, 2])
```

The function to be plotted may also be a user-defined function (see Section A.6).

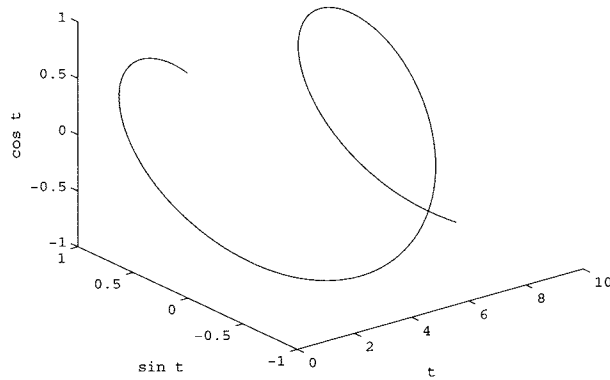
Other useful two-dimensional plotting facilities are (the student is encouraged to try all of these):

```
>> plotyy(x1,y1,x2,y2) % plots x1 versus y1 with y-axis labeling
                        % on the left and plots x2 versus y2 with
                        % y-axis labeling on the right
>> semilogx(x,y) % semilogarithmic plot, x-axis is logarithmic
>> semilogy(x,y) % semilogarithmic plot, y-axis is logarithmic
>> loglog(x,y) % full logarithmic plot, both axes logarithmic
>> area(x,y) % filled area plot
>> polar(x,y) % polar coordinate plot
>> bar(x,y) % bar graph of x on the horizontal axis
              % and y on the vertical axis shown as columns
```

### 3-D graphs

MATLAB has several commands for visualizing three-dimensional functions. A 3-D curve can be shown by the `plot3` command:

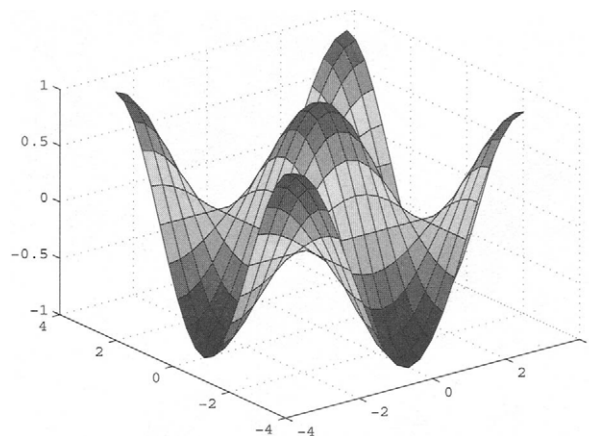
```
>> t = 0:0.01:3*pi;  
>> plot3(t, sin(t), cos(t))  
>> xlabel('t'), ylabel('sin t'), zlabel('cos t')
```



**Figure A.11** Example of a 3-D graph.

Surfaces can be shown in many ways. Here are two that may be of interest:

```
>> [x, y] = meshgrid(-pi:pi/10:pi, -pi:pi/10:pi);  
>> z = cos(x) .* cos(y);  
>> surf(x, y, z)
```



**Figure A.12** Example of a 3-D surface graph.

Also try the following commands:

```
>> mesh(x,y,z)
>> view(30,60)
```

You may make your graph look better by using the `shading` command

```
>> shading interp      %controls the color shading of surface
```

To see the color scale, use `colorbar`

```
>> colorbar
```

## 2½-D Graphs

The so-called 2½-D graph is used for visualizing a 3-D graph on a 2-D system of coordinates. Let us use  $x$ ,  $y$ , and  $z$  variables from the previous section. We can show different  $z$ -levels on an  $x$ - $y$  system of coordinates by its contour lines (Fig. A.13):

```
>> contour(x,y,z)
```

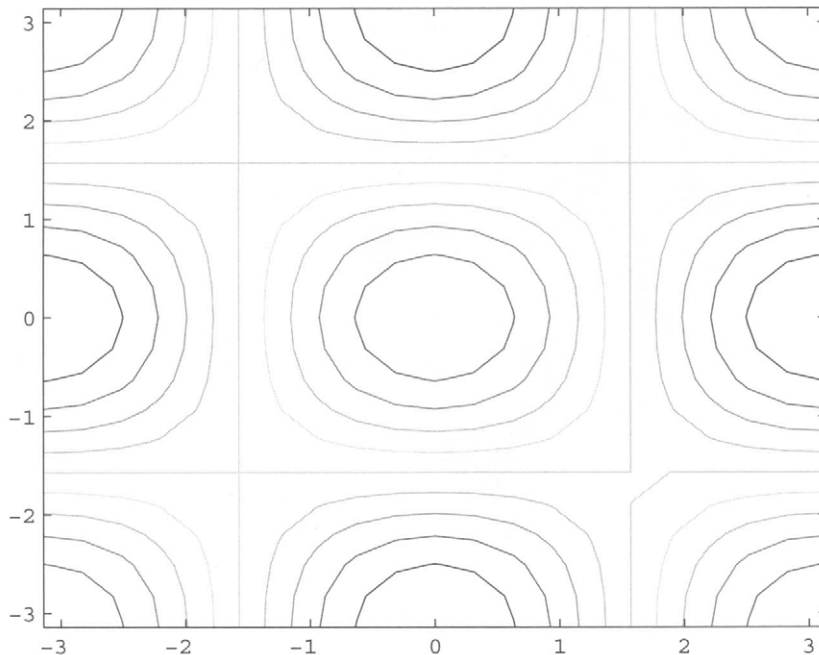


Figure A.13 Contour plots.



It is possible to show only specific contour lines, as required:

```
>> contour(x,y,z,[-0.9:0.3:0.9]) % shows contours between -0.9  
                                % and 0.9 in intervals of 0.3
```

and to print the level values on the contour lines:

```
>> [c,h] = contour(x,y,z,[-0.9:0.2:0.9]);  
>> clabel(c,h)
```

Another method is to look at the graph as a pseudocolor plot (Fig. A.14) that assigns different colors to different z-values:

```
>> pcolor(x,y,z)  
>> colorbar  
>> shading interp
```

The `quiver` command can also be useful in visualizing vector fields such as velocity profiles.

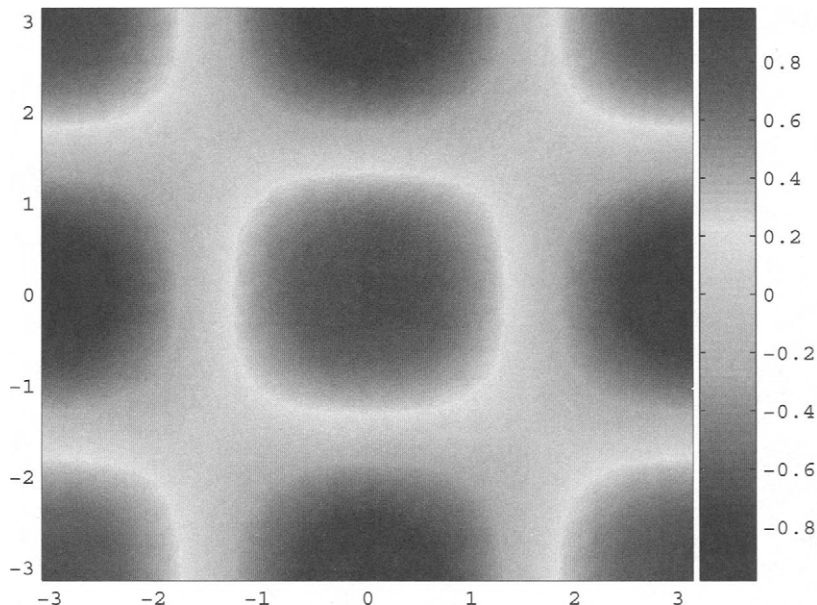


Figure A.14 Pseudocolor plot.

The figures in MATLAB may be edited, rotated, zoomed, and modified using the menus in the graphics window, as described in the next section.

## Interactive Plot Creation

MATLAB 7 introduces a new set of tools to let you interactively create and edit plots without typing any MATLAB code. You can create plots from scratch and visualize your data quickly. You can also automatically generate the code to create the plot again with new data. Interactive plot creation and editing is performed by using the plot tools, which are launched from the figure window toolbar, as shown in Fig. A.15. You can launch the plot tools after opening a blank figure window. However, it is more likely that you will turn them on after creating an initial plot by selecting and plotting from the workspace browser or array editor, from a plotting function entered at the command window, or even from an existing .fig file. The plot tools provide point-and-click functionality that enables you to:

- Drag and drop new data sets onto the figure
- Add new subplot axes
- Change object properties
- Add annotations and draw shapes

In MATLAB 7, you can generate an M-code function from the graphic you created with plot tools (or with plotting functions entered at the command line). You do this by choosing Generate M-file... from the figure's File menu.

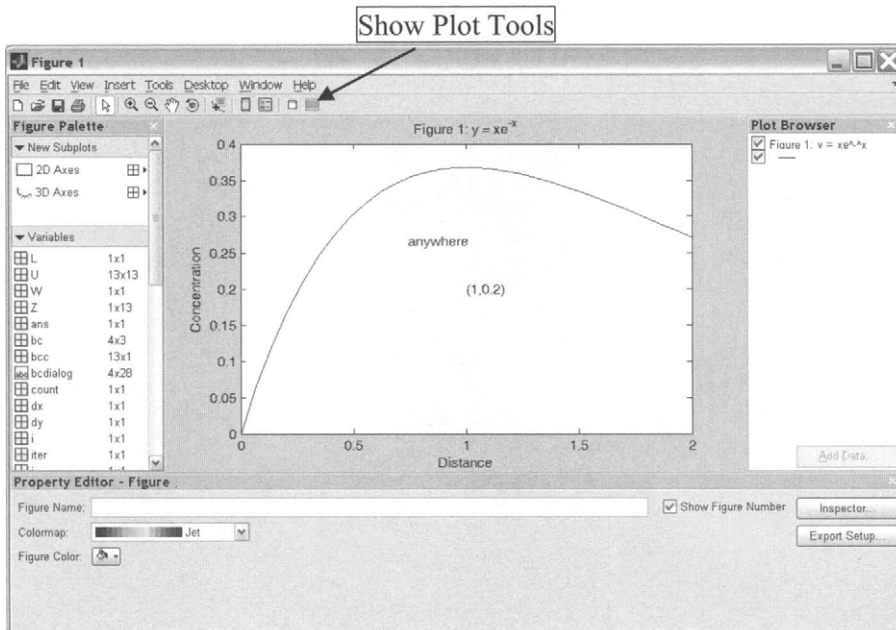
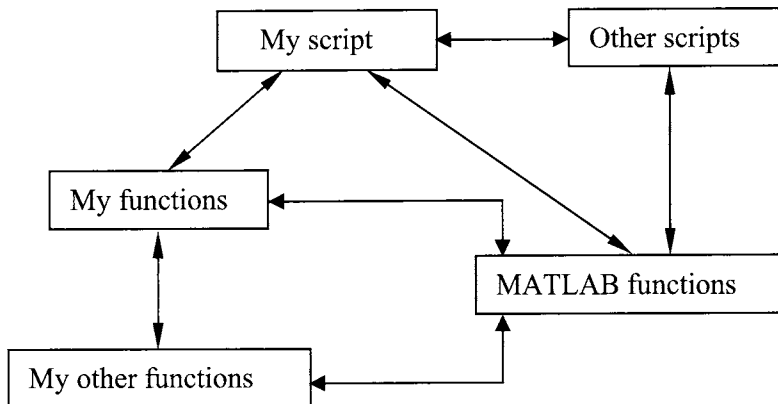


Figure A.15 Turn on the plot tools by clicking the button shown above.

## A.6 Scripts and Functions

Use the MATLAB Editor to write programs in the MATLAB language and save them for future use. The editor saves your files with the `m` extension, which is where the name of *M-file* originates. The editor may also be used to execute and debug the programs. M-files can be in the form of *scripts*, such as a sequence of commands that perform a certain task and serve the purpose of a main program, or *functions*, which perform specific tasks, such as repetitive calculations. The M-files, once debugged, may be executed in the MATLAB workspace. Scripts may call functions written by the user, or built-in MATLAB functions, to perform certain tasks and return results. Scripts may also call on other scripts, as shown on Fig. A.16.



**Figure A.16 Scripts, functions, and MATLAB functions.**

A script is a series of MATLAB commands that are assembled together into a program using the editor. When typing the name of the script, the commands will be executed sequentially as if they were typed in the same order in the MATLAB Command Window. For example, let us calculate the volume of an ideal gas as a function of pressure and temperature. Type the following commands in the editor:

```

% ideal_gas.m - A sample script file
% Calculates the volume of an ideal gas
%
clear all; clc; %clear all previous variables and command window
disp(' Calculating the volume of an ideal gas.')
R = 8314;          % Gas constant
T = input(' Vector of temperatures (K) = ');
P = input(' Pressure (bar) = ')*1e5;
V = R*T/P;        % Ideal gas law
  
```

```
% Plotting the results
plot(T,V)
xlabel('T (K)')
ylabel('V (m^3/kmol)')
title('Ideal Gas Volume vs Temperature')
```

Save the above script as `ideal_gas.m`. Return to the MATLAB command window and type its name:

```
>> ideal_gas
```

Input the required data and observe the results.

A practical method for beginners to create a script is using `diary`. You can start creating a diary by typing:

```
>> diary mydiary
```

Then you start typing your statements in the Command Window, one by one. For example, you can type the statements of the script developed above, see the results at each step, and make corrections if necessary. When you get your desired results, close the diary:

```
>> diary off
```

Now you can develop a script by editing the file `mydiary` (no extension is added by MATLAB). Delete the unnecessary commands and output, and save it as an M-file.

You can develop your own functions and execute them just like any other function in MATLAB. A function takes in some data as input, performs the required calculations, and returns the results of calculations back to the command that invoked the function. As an example, let us write a function to do the ideal gas volume calculations that we have already done in a script. We can make this function more general so that it will be able to calculate the volume at multiple pressures and multiple temperatures:

```
function V = ideal_gas_func(T, P)
% This function calculates the specific volume of an ideal gas
R = 8314; % Gas constant
for k = 1:length(P)
    V(k,:) = R*T/P(k); % Ideal gas law
end
```

The first line of a function is the function declaration line and begins with the word `function` followed by the output argument(s), equality sign, name of the function, and input argument(s), as illustrated in the example above. The first set of continuous

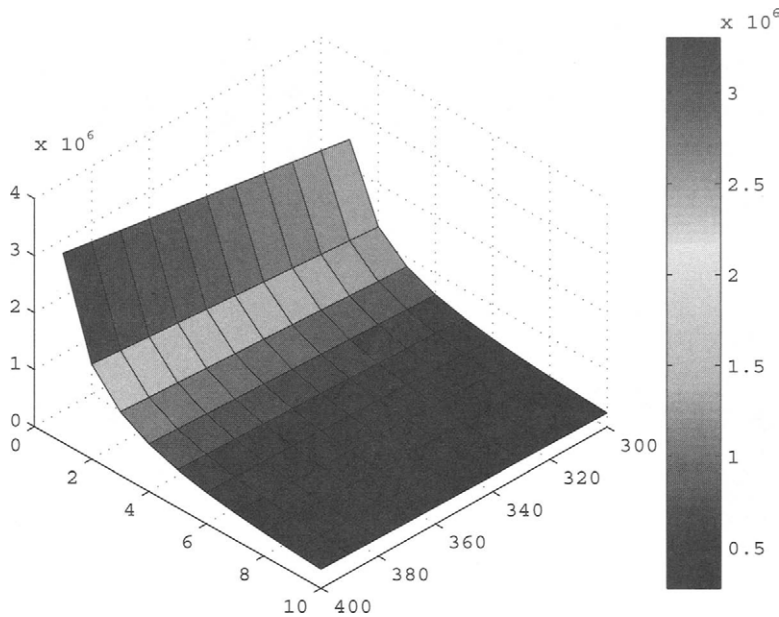
comment lines immediately after the function declaration line is the help for the function and can be reviewed separately:

```
>> help ideal_gas_func
```

This function must be saved as `ideal_gas_func.m`. You can now use this function in the workspace, in a script, or in another function. For example:

```
>> P = [1:10]; T = [300:10:400];  
>> vol = ideal_gas_func(T, P);  
>> surf(T, P, vol)  
>> view(135,45), colorbar
```

The results of these commands are shown on Fig. A.17.



**Figure A.17** Plot of ideal gas volume as function of temperature and pressure

Functions do not necessarily require input, and may not return the results of calculations back to the command. For example, the script `ideal_gas.m` listed above may also be saved in the form of a function:

```
function ideal_gas  
% Calculates the volume of an ideal gas  
%  
clear all; clc; %clear all previous variables and command window
```

```

disp(' Calculating the volume of an ideal gas.')
R = 8314;           % Gas constant
T = input(' Vector of temperatures (K) = ');
P = input(' Pressure (bar) = ')*1e5;
V = R*T/P;         % Ideal gas law
% Plotting the results
plot(T,V)
xlabel('T (K)')
ylabel('V (m^3/kmol)')
title('Ideal Gas Volume vs Temperature')

```

In this case, there is little difference between the script and the function. Either one may be executed from the Command Window (or from another script) by invoking it by name.

## A.7 Flow Control

MATLAB has several flow control structures that allow the program to execute different sequences of statements depending on the value of data. These structures are `if`, `switch`, `for`, and `while`, which we describe briefly below. We do not show the results of each operation. It is left as an exercise to the student to use these commands in sample programs and observe the results:

`if . . . (else . . .) end` –The `if` control structure enables the program to make decision about what commands to execute. In the following sequence, the line `y = x^2` executes only if `x` input is greater than or equal to zero:

```

x = input(' x = ');
if x >= 0
    y = x^2
end

```

You can also add an `else` clause with statements that are executed if the condition in the `if` statement is not true:

```

x = input(' x = ');
if x >= 0
    y = x^2
else
    y = -x^2
end

```

`switch . . . case . . . end` –Another form of conditional execution is to execute a different block of statements for different value of a variable. The `switch . . . case . . . end` structure is easier to use than a nested `if` structure. For example:

```
a = input('Give a value of a (1, 2, or 3) = ');
switch a
case 1
    disp('One')
case 2
    disp('Two')
case 3
    disp('Three')
end
```

**for ... end** –This control structure is used to repeat the execution of a block of statements a fixed number of times:

```
k = 0;
for x = 0:0.2:1
    k = k+1
    y(k) = exp(-x)
end
```

**while ... end** –This control structure is used to repeat the execution of a block of statements until a condition of the data exists:

```
x = 0;
while x<1
    y = sin(x)
    x = x+0.1;
end
```

There are three other flow control commands: **break**, **pause**, and **return**. The **break** command is used to stop executing an iteration before it has completed. The **pause** command will cause the program to wait for a key to be pressed (on the keyboard) before continuing:

```
k = 0;
for x = 0:0.2:1
    if k>3
        break
    end
    k = k+1
    y(k) = exp(-x)
    pause
end
```

The **return** command terminates a function invocation, returning to either the calling script or to the command line.

## A.8 Display, Export, and Import of Data

### A.8.1 Displaying data and results

There are many different ways that data and other information may be displayed on the screen, sent to the printer, saved as files, or exported to other formats. We begin with the most elementary ones: `disp` and `fprintf`.

```
>> X=[1 2 3; 4 5 6];  
>> disp(X)  
     1     2     3  
     4     5     6
```

The `disp` command displays the array without printing the array name. Only one variable may be displayed at a time. In all other ways the `disp` command is the same as leaving the semicolon off an expression, except that empty arrays do not display. The `disp` command may also be used to display text:

```
>> disp('This is the X array:'); disp(X)  
This is the X array:  
     1     2     3  
     4     5     6
```

The `fprintf` command writes formatted data to the screen or to a file. For example, to write text and data to the screen:

```
>> fprintf('\n The value of position (2,2) of X = %g',X(2,2))  
  
The value of position (2,2) of X = 5
```

The special format control characters `\n`, `\r`, `\t`, `\b`, and `\f` can be used to produce new line, carriage return, tab, backspace, and formfeed characters, respectively. Use `\\` to produce a backslash character and `%%` to produce the percent character.

The `%g` above is an example of a format specification string that controls how the data will be exhibited. This string contains C language conversion specifications that involve the character `%`, optional flags, optional width and precision fields, optional subtype specifier, and conversion characters `c`, `d`, `e`, `E`, `f`, `g`, `G`, `i`, `o`, `s`, `u`, `x`, and `x`. The uses of these characters are described in Table A.1.

Several examples of using the `fprintf` command are given below. The students should format the commands in a way that presents their data and results in an informative and easy-to-read display, appropriate for the problem at hand.

```
>> fprintf('This is a test of the printing commands')  
This is a test of the printing commands  
>> a=-5.23 ; b=3; c=0.000000015; d = 'Test';
```



```

>> fprintf('Value of a = %7.4f   b = %2i   c = %10g   d = %s \n', a,
b, c, d)
Value of a = -5.2300   b =   3   c =   1.5e-008   d = Test

>> fprintf('%12.2e', pi)
   3.14e+000

>> fprintf('%0.5e', pi)
3.14159e+000

>> fprintf('%0.5f', pi)
3.14159

>> fprintf('%10.5f', pi)
   3.14159

>> fprintf('%15.5g', eps)
   2.2204e-016

>> y=[1 2 3 4];
>> fprintf('%5.2f \n' ,y)
   1.00
   2.00
   3.00
   4.00

```

**Table A.1 List of format symbols for printing in MATLAB**

Specifier	Description
%c	Single character
%d	Decimal notation (signed)
%e	Exponential notation (using a lowercase e as in 3.1415e+00)
%E	Exponential notation (using an uppercase E as in 3.1415E+00)
%f	Fixed-point notation
%g	The more compact of %e or %f. Insignificant zeros do not print.
%G	Same as %g, but using an uppercase E
%i	Decimal notation (signed)
%o	Octal notation (unsigned)
%s	String of characters
%u	Decimal notation (unsigned)
%x	Hexadecimal notation (using lowercase letters a-f)
%X	Hexadecimal notation (using uppercase letters A-F)

### A.8.2 Saving and loading data

There are several ways in which you can save your data in MATLAB. Let us first clear all other variables from the workspace and generate some new variables:

```
>> clear
>> a = magic(3), b = magic(4) %magic is a special MATLAB matrix
a =
     8     1     6
     3     5     7
     4     9     2
b =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

The following command saves all the variables in the MATLAB workspace in the file `f1.mat`:

```
>> save f1
```

If you need to save only specific variables from the workspace, list their names after the file name. The following saves only the variable `a` in the file `f2.mat`:

```
>> save f2 a
```

The files generated above have the extension “`.mat`” and can be retrieved only by MATLAB. To use the data elsewhere you may want to save them in text format:

```
>> save f3 b -ascii
```

Here, the file `f3` is a text file with no extension in its name. To see all the different ways that `save` may be used, give the command `help save`.

You can load data into the MATLAB workspace using the `load` command. If the file to be loaded was generated by MATLAB (carrying the `.mat` extension), the variables will appear in the workspace with the same names they had at the time they were saved:

```
>> clear
>> load f1
>> whos      % verifies the presence of these data
```

Name	Size	Bytes	Class
a	3x3	72	double array

```
b          4x4          128 double array

Grand total is 25 elements using 200 bytes
```

However, if the file is a text file, the variables will appear in the workspace under the name of the file:

```
>> clear
>> load f3
>> whos
  Name      Size      Bytes  Class

  f3        4x4        128    double array

Grand total is 16 elements using 128 bytes
```

Data may also be created using the MATLAB editor. For example, open the Editor and enter the values of the vector variable *y* as a column of numbers:

```
0.8345
0.0381
0.0163
0.0287
0.0220
0.0434
```

Save the file as *y.dat*.

To load the data from the file just saved, use the `load` command from the MATLAB Command Window (or from within a script):

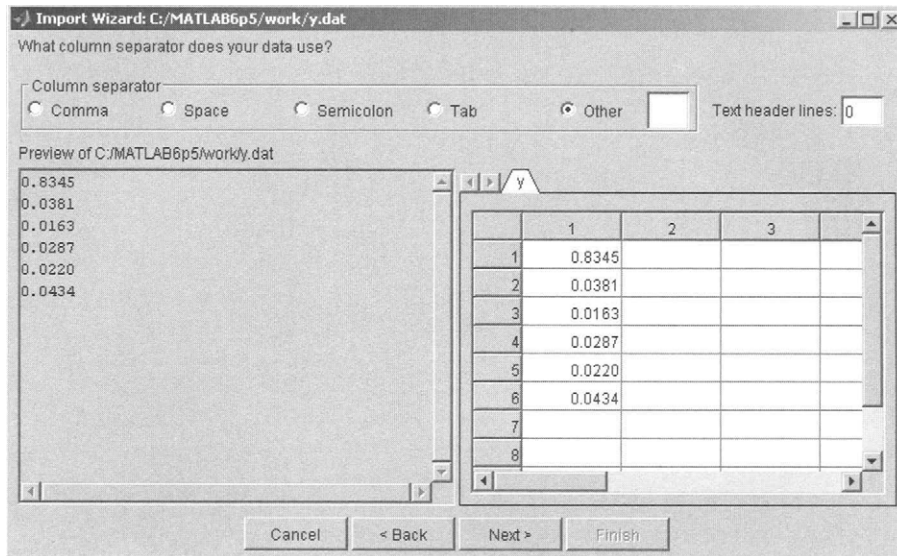
```
load y.dat
```

The workspace now contains the vector variable *y*.

Another way of reading the data is by using the `open` command:

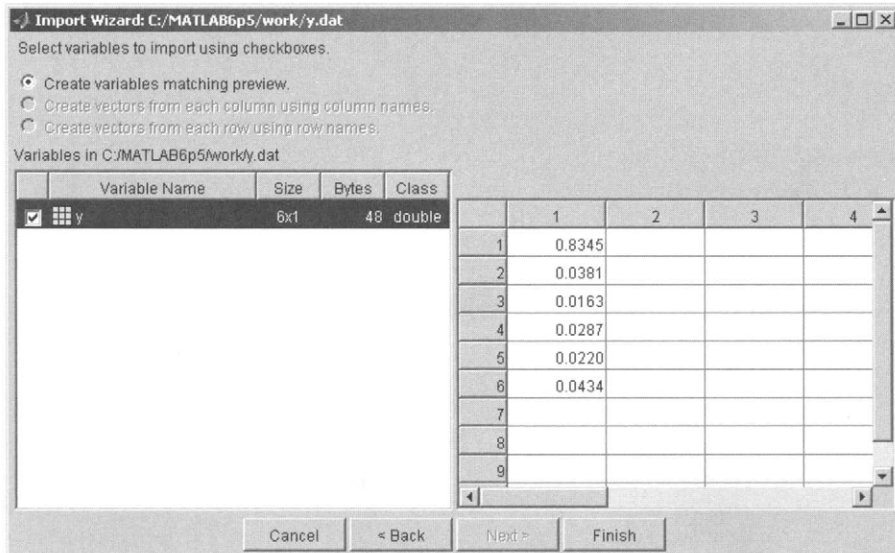
```
open y.dat
```

The Import Wizard (Fig. A.18) opens up automatically when the `open` command is used. It gives you the opportunity to specify how the data are formatted before they are read into the workspace.



**Figure A.18 Import Wizard.**

Press the **Next** button, to see the following window (Fig. A.19), which gives options for formatting multidimensional data:



**Figure A.19 Import Wizard: options for formatting multidimensional data.**

Pressing the **Finish** button will complete the task of loading `y`. Spreadsheet files, such as Excel files, may also be read this way, including column and row headings.

Alternatively, from the MATLAB Command Window (or from within a script), open the file and read it:

```
fidy = fopen('y.dat');
% fidy is the identification number for this file
y=fscanf(fidy, '%f')
y =
    0.8345
    0.0381
    0.0163
    0.0287
    0.0220
    0.0434
```

Always close the file when finished reading it:

```
fclose(fidy)
```

Similarly, you can create a file with the values of  $x$  and  $y$  and name it `xy.dat`:

```
0.0    0.8345
0.1    0.0381
0.2    0.0163
0.3    0.0287
0.4    0.0220
0.5    0.0434
```

Since the first column is  $x$  and the second column is  $y$ , we need to read each data in the sequence  $x(1)$ ,  $y(1)$ ,  $x(2)$ ,  $y(2)$ ,  $x(3)$ ,  $y(3)$ ,... etc. To do so let us write a script using the MATLAB Editor:

```
% Opening and reading xy.dat file with two columns of data
%
clear
fidxy=fopen('xy.dat');
for i=1:6
    x(i) = fscanf(fidxy,'%f',1); % reads one value of x at a time
    y(i) = fscanf(fidxy,'%f',1); % reads one value of y at a time
end
fclose(fidxy);
x, y
```

The `fscanf` command reads formatted data from a file. Save this script as `read_xy.m` and use it from the Command window:

```
>> read_xy
x =
```

```

Columns 1 through 5
    0    0.1000    0.2000    0.3000    0.4000
Column 6
    0.5000
y =
Columns 1 through 5
    0.8345    0.0381    0.0163    0.0287    0.0220
Column 6
    0.0434

```

Please note that `x` and `y` are read as row vectors (rather than column vectors).

Also, try using the commands `open` and `load` and notice the difference in how these commands read the file:

```
open xy.dat
```

or

```
load xy.dat
```

### A.8.3 Generating data in a program and saving into a file

The program below generates some data, creates and opens a text file named `expon.dat`, uses the `fprintf` command to write the data into a file using the desired format, and closes the file `expon.dat` that contains a short table of the exponential function:

```

x = 0:0.1:1;
y = [x; exp(x)];
fid = fopen('expon.dat','w');
fprintf(fid,'%6.2f %12.8f\n',y);
fclose(fid);

```

Use the command `load` to read the file and place all the data into one variable (matrix):

```

clear
load expon.dat
expon
expon =
    0    1.0000
    0.1000    1.1052
    0.2000    1.2214
    0.3000    1.3499
    0.4000    1.4918
    0.5000    1.6487
    0.6000    1.8221
    0.7000    2.0138

```

0.8000	2.2255
0.9000	2.4596
1.0000	2.7183

If necessary, use the method described in Section A.8.2 to open the `expon.dat` file and read it as two variables.

## A.9 Symbolic Computation

The Symbolic Math Toolbox, an integral part of the MATLAB Student Version, incorporates analytic computation into MATLAB's numeric environment. This toolbox enables the student to perform computations using symbolic mathematics and variable-precision arithmetic. The student is strongly encouraged to walk through the Symbolic Math demos that are provided (see Help/Demos/Toolboxes/Symbolic Math). Only a few examples of the symbolic math commands will be given in this Appendix.

To create a symbolic variable use the command `sym( )`:

```
>> y= sym('y')
y =
y
>> om= sym('omega')
om =
omega
```

The above commands created a symbolic variable `y` that prints as `y`, and a symbolic variable `om` that prints as `omega`. Variables may also be declared as symbolic by the command `syms`:

```
>>syms e f g
```

### A.9.1 Symbolic solution of algebraic equations

Now, let us use the symbolic function `solve` to find the solution of the quadratic equation:

$$ax^2 + bx + c = 0$$

Several ways of using the `solve` function will be demonstrated below:

(a) Find the zeroes of the function, assuming that the unknown is `x`.

```
>>clear
>> x = solve('a*x^2+b*x+c')
x =
[ 1/2/a*(-b+(b^2-4*a*c)^(1/2))]
[ 1/2/a*(-b-(b^2-4*a*c)^(1/2))]
```

(b) The user specifies that the function = 0, and the unknown is x.

```
>> x = solve('a*x^2 + b*x + c=0','x')
x =
[ 1/2/a*(-b+(b^2-4*a*c)^(1/2))]
[ 1/2/a*(-b-(b^2-4*a*c)^(1/2))]
```

(c) The function is defined symbolically as s.

```
>> syms x a b c S
S = a*x^2 + b*x + c;
x = solve(S)
x =
[ 1/2/a*(-b+(b^2-4*a*c)^(1/2))]
[ 1/2/a*(-b-(b^2-4*a*c)^(1/2))]
```

(d) The function is = 5, and the unknown is x.

```
>> x = solve('a*x^2 + b*x + c=5','x')
x =
[ 1/2/a*(-b+(b^2-4*a*c+20*a)^(1/2))]
[ 1/2/a*(-b-(b^2-4*a*c+20*a)^(1/2))]
```

(e) The function is solved for a instead of x.

```
>> a = solve('a*x^2 + b*x + c=5','a')
a =
-(b*x+c-5)/x^2
```

Solve two simultaneous algebraic equations:

$$\begin{aligned}\alpha N_1 - \beta N_1 N_2 &= 0 \\ -\gamma N_2 + \delta N_1 N_2 &= 0\end{aligned}$$

```
>> [N1,N2] = solve('alpha*N1 - beta*N1*N2 = 0',...
'-gamma*N2 + delta*N1*N2 = 0','N1,N2')
N1 =
[ 0]
[ 1/delta*gamma]
N2 =
[ 0]
[ 1/beta*alpha]
```

Note: Three or more points at the end of a line (...) indicate continuation of a MATLAB statement into the next line.



### A.9.2 Symbolic solution of differential equations

Solve the differential equation:

$$\frac{dy}{dt} + 4y = e^{-t}$$

```
>> y = dsolve('Dy + 4*y = exp(-t)')
y =
(1/3*exp(3*t)+C1)*exp(-4*t)
```

Solve the same differential equation with the initial condition  $y(0)=2$ :

```
>> y = dsolve('Dy + 4*y = exp(-t)', 'y(0)=2')
y =
(1/3*exp(3*t)+5/3)*exp(-4*t)
```

Set the range of the independent variable  $t$ , evaluate the dependent variable  $y$ , that was obtained above, for this range, and plot the profile (Fig. A.20):

```
>> tt = [0:.1:5];
>> for i=1:length(tt)
t=tt(i);
yy(i)=eval(y);
end
```

Note that the symbolic solution of  $y$  contains  $t$ , as a scalar, therefore  $t$  had to be created as such.

```
>> plot(tt,yy)
```

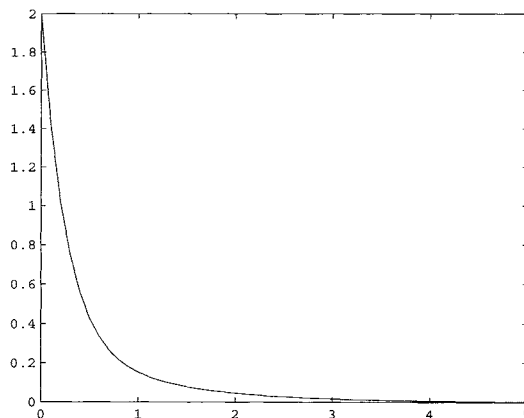


Figure A.20

Solve two simultaneous differential equations:

$$\frac{dy_1}{dt} + 4y_2 = e^{-t} \quad \text{with initial condition } y_1(0) = 1$$

and

$$\frac{dy_2}{dt} = -y_1 \quad \text{with initial condition } y_2(0) = 2$$

```
>> [y1,y2]=dsolve('Dy1 + 4*y2 = exp(-t)', 'Dy2=-y1', 'y1(0)=1, y2(0)=2')
y1 =
-4/3*exp(2*t)+2*exp(-2*t)+1/3*exp(-t)
y2 =
2/3*exp(2*t)+exp(-2*t)+1/3*exp(-t)
```

Verify the answers by substituting them into the differential equations. The function `diff` differentiates the symbolic expression (See Section A.9.3)

```
>> diff(y1) + 4*y2
ans =
exp(-t)
>> diff(y2)
ans =
4/3*exp(2*t)-2*exp(-2*t)-1/3*exp(-t)
```

Set the range of the independent variable, evaluate the dependent variable for this range, and plot the profile:

```
>> t=[0:.01:1]; yy1=eval(y1); yy2=eval(y2);
>> plot(t,yy1,':', t, yy2)
```

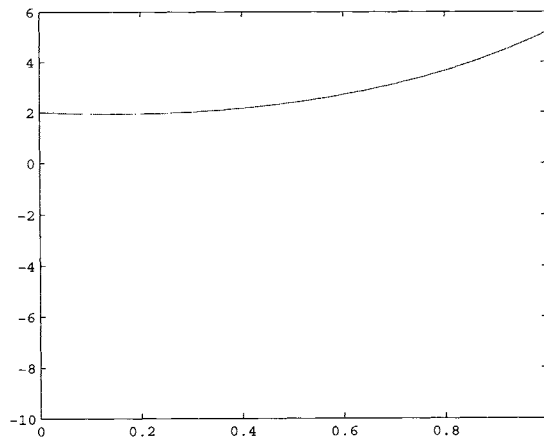


Figure A.21

### A.9.3 Symbolic differentiation

Clear all previous variables and define the new symbolic variables:

```
>> clear
>> syms x a f Re e
```

Differentiate:

```
>> diff(x^2)
ans =
2*x
>> diff(sin(x))
ans =
cos(x)

>> diff(exp(x^2))
ans =
2*x*exp(x^2)

>> diff(log(x))
ans =
1/x

>> diff(acos(x))
ans =
-1/(1-x^2)^(1/2)

>> diff(acos(x))
ans =
-1/(1-x^2)^(1/2)

>> diff(1/sqrt(f)+.86*log(e/3.7+2.51/Re/sqrt(f)), f)
ans =
-1/2/f^(3/2)-10793/10000/Re/f^(3/2)/(10/37*e+251/100/Re/f^(1/2))
```

### A.9.4 Symbolic integration

In the following vector, the first two elements involve integration with respect to  $x$ , while the second two are with respect to  $a$ .

```
>> [int(x^a), int(a^x), int(x^a, a), int(a^x, a)]
ans =
[ x^(a+1)/(a+1), 1/log(a)*a^x, 1/log(x)*x^a, a^(x+1)/(x+1)]
```

Another example of integration:

```
>> int('a*T+b*T^2+c*T^4','T')
ans =
1/2*a*T^2+1/3*b*T^3+1/5*c*T^5
```

Limits of integration may be used:

```
>> int('a*T+b*T^2+c*T^4','T',298,500)
ans =
80598*a+98536408/3*b+28899927176032/5*c
```

## A.10 MATLAB Toolboxes

Toolboxes are specialized collections of M-files built specifically for solving particular classes of problems. These toolboxes are optional, and may be licensed separately from MathWorks, Inc. Toolboxes related to the topics of this textbook are listed in Table A.2.

**Table A.2 Selected MATLAB Toolboxes**

Toolbox	Description
Curve Fitting	Perform model fitting and analysis
Image Processing	Perform image processing, analysis, and algorithm development
Partial Differential Equation	Solve and analyze partial differential equations
Signal Processing	Perform signal processing, analysis, and algorithm development
Simulink	Model, simulate, and analyze dynamic systems
Statistics	Apply statistical algorithms and probability models
Symbolic Math	Perform computations using symbolic mathematics and variable-precision arithmetic

The MATLAB Student Version contains Simulink and Symbolic Math toolboxes in addition to the basic MATLAB.

## A.11 References

- Hanselman, D., and Littlefield, B. 2005. *Mastering MATLAB 7*. Upper Saddle River, NJ: Prentice Hall.
- Hahn, B. D. 2002. *Essential MATLAB for Scientists and Engineers*. Oxford, UK: Butterworth-Heinemann Publications.
- Lyshevski, S. E. 2003. *Engineering and Scientific Computations Using MATLAB*. Hoboken, NJ: John Wiley & Sons, Inc..