PROGRAMMING IN MATLAB

What is Matlab?

Matlab is an interactive environment that allows us to easily enter data, numerically solve problems and it provides graphing capabilities so we can visualize the results. In this document, green text shows Matlab commands and blue text shows Matlab respond.

Starting Matlab

To start Matlab on a Windows machine, click *Start\Programs\Matlab\Matlab* or click on the Matlab icon on your desktop. To start Matlab from a UNIX machine, type 'matlab' at a command prompt. The Matlab command window and command line should appear similar to that shown in Figure 1

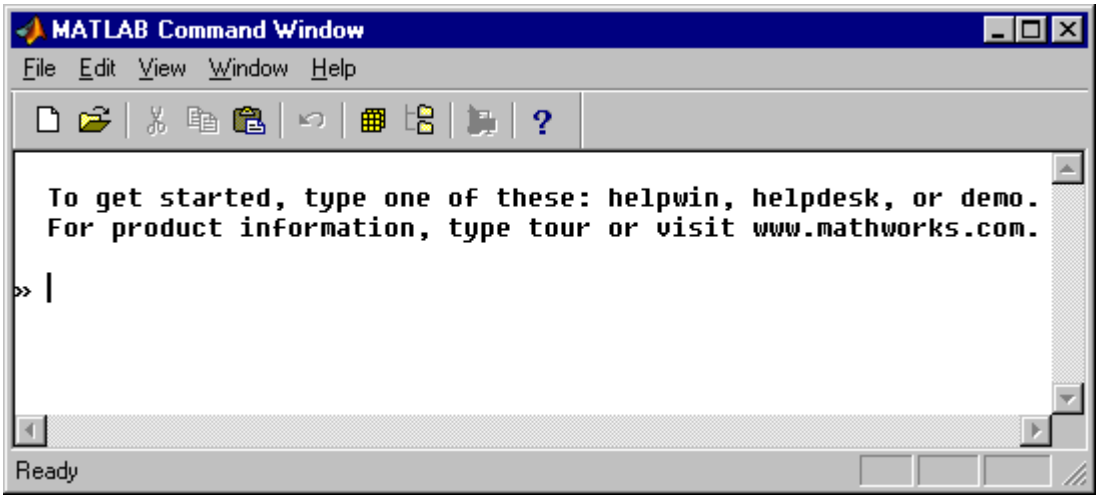

Figure 1 - The Matlab command line as viewed on a Windows machine.

Need help?

Getting help:

» ***help***

example: getting help for if command.

» ***help if***

Keyword search in the descriptions of the command plot:

» ***lookfor plot***

» ***help elfun***

Elementary math functions.

Trigonometric.

sin	- Sine.
sinh	- Hyperbolic sine.
asin	- Inverse sine.
asinh	- Inverse hyperbolic sine.
cos	- Cosine.
cosh	- Hyperbolic cosine.
acos	- Inverse cosine.
acosh	- Inverse hyperbolic cosine.
tan	- Tangent.
tanh	- Hyperbolic tangent.
atan	- Inverse tangent.
atan2	- Four quadrant inverse tangent.
atanh	- Inverse hyperbolic tangent.
sec	- Secant.
sech	- Hyperbolic secant.
asec	- Inverse secant.
asech	- Inverse hyperbolic secant.
csc	- Cosecant.
csch	- Hyperbolic cosecant.
acsc	- Inverse cosecant.
acsch	- Inverse hyperbolic cosecant.
cot	- Cotangent.
coth	- Hyperbolic cotangent.
acot	- Inverse cotangent.
acoth	- Inverse hyperbolic cotangent.

Exponential.

exp	- Exponential.
log	- Natural logarithm.
log10	- Common (base 10) logarithm.
log2	- Base 2 logarithm and dissect floating point number.
pow2	- Base 2 power and scale floating point number.
sqrt	- Square root.
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.

Getting started

Here is an example in Matlab.

```
» 3*4
ans =        12

» 2^3
ans =     8


» d=[11 12 13 ; 21 22 23 ; 31 32 33]
d =          11    12    13
             21    22    23
             31    32    33
```

Using Matlab built-in functions

```
» sqrt(64)
ans =        8

» e = ones(3,3)
e =          1     1     1
             1     1     1
             1     1     1
```

```
» f = d + e
f =      12   13   14
         22   23   24
         32   33   34

» f'
ans =    12   22   32
         13   23   33
         14   24   34
```

complex numbers:

```
» a=[ 1+i   2]
a =   1.0000 + 1.0000i   2.0000

» b=[4-2i   5+10i];
>> the_product = a * b

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

» the_product = a' * b

the_product =
   2.0000 - 6.0000i  15.0000 + 5.0000i
   8.0000 - 4.0000i  10.0000 +20.0000i
```

The "colon" (:) operator:

```
» numbers1 = 2:2:8
numbers1 =     2     4     6     8
» numbers2 = 2:8
numbers2 =     2     3     4     5     6     7     8
```

Element-by-element operations:

```
» elem_prod = a' .* b
??? Error using ==> .*
Matrix dimensions must agree.
» elem_prod = a .* b
elem_prod =   6.0000 + 2.0000i  10.0000 +20.0000i
```

Basic Operations

The following matrix operations are available in MATLAB:

+	addition
-	subtraction
*	multiplication
^	Power
'	transpose
\	left division
/	right division

x = A \ b is the solution of A * x = b and, resp.,
x = b / A is the solution of x * A = b.

Saving and Restoring Workspace

At this point you may be bored with our little tutorial and wish to spend a few hours with your friends in the village. If you leave your computer unattended you run the risk of someone (an evil roommate perhaps) closing Matlab in which case you might loose the data you tediously entered. Fortunately we may save the Matlab workspace to disk for retrieval at a later time. First let's obtain a summary of the variables we have so far, use the 'whos' command.

» ***whos***

```
  Name      Size            Bytes  Class

  A         3x4               96  double array
  B         3x3               72  double array
  ans       3x3               96  double array
  x         1x1               16  double array (complex)
```

To save the workspace we can type

» ***save practicedata***

which saves the data in the file 'practicedata.mat' in the current directory. To verify that the file is there, we can actually execute shell commands from Matlab. On a Windows machine we can type 'dir' to get a directory listing. On a UNIX machine we can type 'ls' to get a directory listing. We must precede the shell command with an exclamation point (actually, some commands work without it).

» ***!dir***

```
Volume in drive C is Applications1
Volume Serial Number is 40ED-0A50

Directory of C:\MATLABR11\work

01/07/00  09:58p       <DIR>          .
01/07/00  09:58p       <DIR>          ..
01/07/00  09:55p                 416 practicedata.mat
               3 File(s)          416 bytes
                        582,668,800 bytes free
```

Now our data is stored safely on disk. Let's clear out the variables in our workspace so it's as if we've closed Matlab and reopened it. We use the 'clear' command to remove all variables and then obtain another summary.

» ***clear***
» ***whos***

Matlab doesn't list any variables after the 'whos' command which verifies that all variables have been cleared. We may now restore the workspace using the 'load' command.

» load practicedata
» whos

```
  Name      Size         Bytes  Class

  A         3x4             96  double array
  B         3x3             72  double array
  ans       3x3             96  double array
  x         1x1             16  double array (complex)
               Grand total is 34 elements using 280 bytes
```

Suppose we only want to save a few variables instead of all of them, how do you think we'd accomplish that? Try typing 'help save' to obtain more information.

On a Windows machine we may alternately using the menu items File\Save Workspace and File\Load Workspace to save and restore the workspace respectively.

Visualization

Matlab has many features that allow us to visual data. Let's generate a vector of sine wave samples and plot the results. First we will create a length 100 vector represented by the variable *x* containing all zeros using the 'zeros' function. Then we will use a for-loop to fill the vector with sine wave samples. Note that in the following example, we place a semicolon at the end of the line 'x(k)=sin(k*2*pi/100);', this prevents the value of 'x(k)' from being displayed during each iteration of the loop.

```
» x=zeros(100,1);
» for k=1:100
x(k)=sin(k*2*pi/100);
end
```

To plot the results we simply use the 'plot' command. We follow this command with the 'xlabel', 'ylabel' and 'title' commands to label the plot, these commands are completely optional.

```
» plot(x)
» xlabel('sample index');
» ylabel('sample value');
» title('Sine wave');
```

A plot should appear in a separate window. It should be similar to that shown in Figure 2.

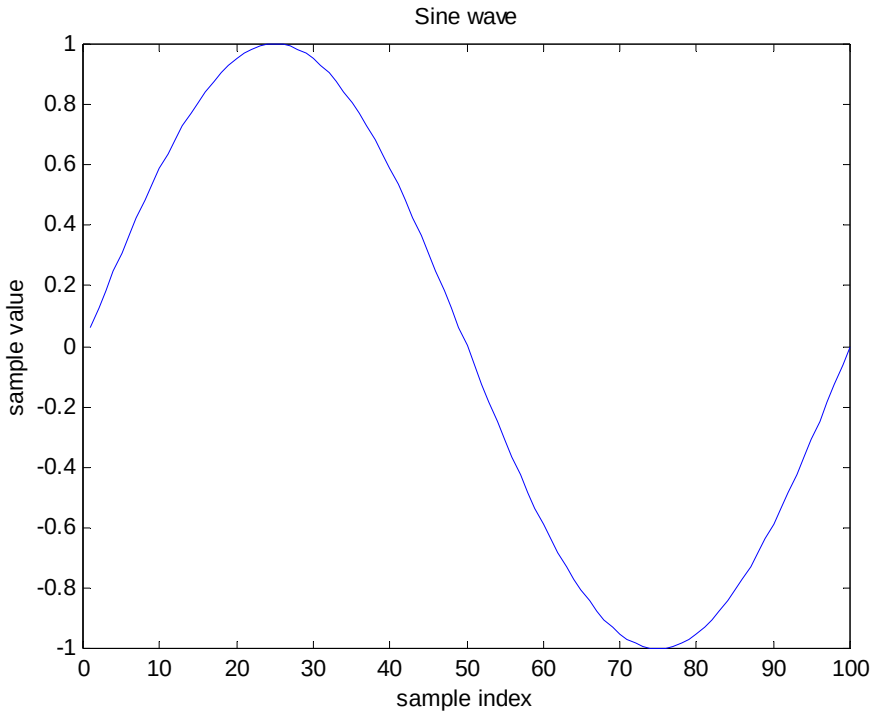

Figure 2 – Sine wave plot.

Next, let's superimpose a second plot. For the second plot we'll use a sine wave of twice the frequency. It is important to point out that for-loops are very inefficient in Matlab. In time critical applications we should find ways to eliminate the use of for-loops. For generating a sine wave it is simple and we will show how by example. We'll create a length 100 vector represented by the variable *z* containing the samples of our second sine wave.

» ***z=sin((1:2:200)*2*pi/100);***

The statement '(1:2:200)' generates a sequence of numbers from 1 to 200 in increments of 2 (actually, from 1 to 199). That is, it generates the vector *[1 3 5 … 199]*. Do not use 'plot(z)' at this juncture for it will overwrite the existing plot. Since we wish to superimpose the plots, we use the 'hold' command.

» ***hold on;***

Since we wish to differentiate *x* from *z*, we plot *z* in the color red and using a dotted line using a slightly different version of the 'plot' command.

» ***plot(z,'red.')***

The plot should appear similar to that shown in Figure 3. If we then use the command 'hold off', future plot commands will overwrite the existing plot.

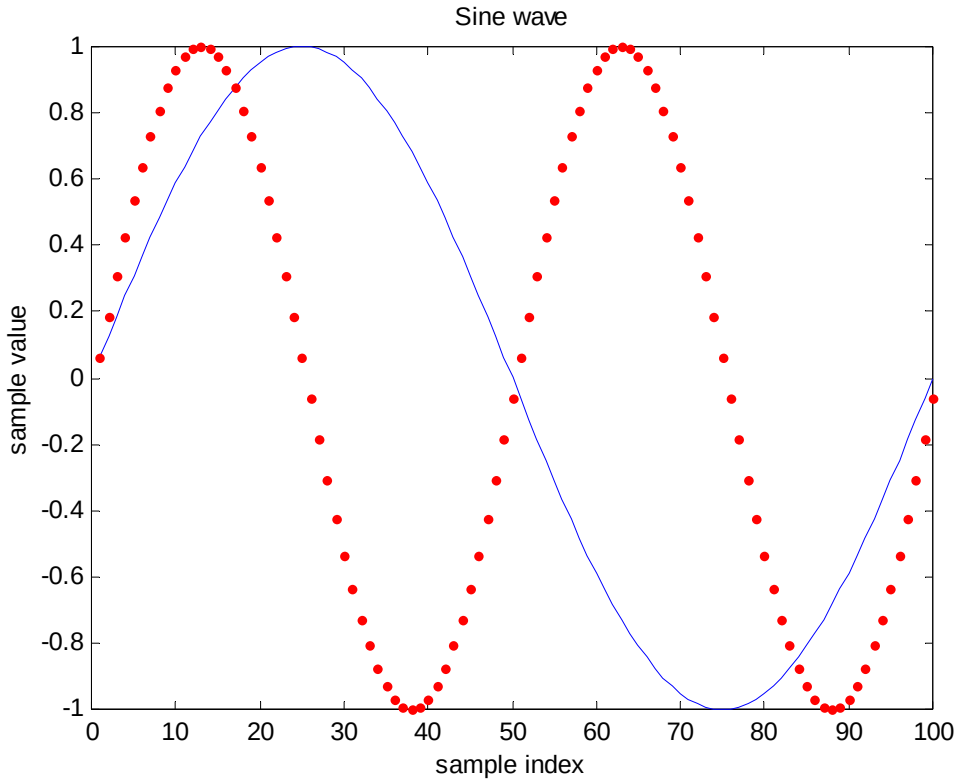

Figure 3 – Superimposed sine wave plots.

What if we wanted a legend to accompany our plot? Much more sophisticated plots are possible. Try using the help system to obtain more information about plots and related topics.

Programming and M-Files

So far we have done all of our work using the Matlab command line. We used functions like 'sin' for generating data. M-files provide a way for us to write our own functions. An m-file is a script that allows us to sequence commands. Any command that can be used from the command line can be used in an m-file.

Let's write an m-file to generate the two vectors *x* and *z* from the previous example. The function prototype should look like this:

[x,z]=***generatesinewaves(n);***

where *n* is the length of the vectors. The m-file should plot the vectors and return them. We use a text editor to create m-files.

On a UNIX machine for instance we can use a text editor such as emacs or vi. Create a text file with the filename 'generatesinewave.m' (for example, type 'vi generatesinewave.m' at a command line in the same directory that you used for starting Matlab).

On a Windows machine, we can also use any text editor such as notepad, but there's a built-in editor called 'MEdit' that has some useful debugging utilities. To start MEdit, click *File\New\M-File* from the Matlab menu.

The first line of the m-file must contain the keyword 'function' followed by the function prototype.

function [x,z]=***generatesinewaves(n);***

Enter this line now. With MEdit, click *File\Save* now and it will automatically generate the correct filename. The lines following the first will contain the help information that can be displayed from the Matlab command line by typing 'help generatesinewaves'. Each line of the help information must begin with the '%' character, this character indicates that the line is a comment. The first comment line should be the function prototype. Enter the help information now, the m-file so far should appear as follows:

```
function [x,z]=generatesinewaves( n );
% [x,z]=generatesinewaves(n);
% Input:
% n - the number of samples
% Output:
% x - a vector containing n samples of a single cycle of a sine wave
% z - a vector containing n samples of two cycles of a sine wave
%
% Plots of the sine waves are generated automatically.
```

Now we are ready to list the commands needed to generate the vectors and plot the results. The complete m-file is shown below, notice that we have used additional comments in the code to make it more readable.

```
function [x,z]=generatesinewaves( n );
% [x,z]=generatesinewaves(n);
% Input:
% n - the number of samples
% Output:
% x - a vector containing n samples of a single cycle of a sine wave
% z - a vector containing n samples of two cycles of a sine wave
%
% Plots of the sine waves are generated automatically.

% Generate the vectors x and z
x=sin( (1:1:n)*2*pi/n );
z=sin( (1:2:n)*2*pi/n );

% Plot the results
plot(x);
xlabel('sample index');
ylabel('sample value');
title('Sine waves');
hold on;
plot(z,'red.');
```

Save the file when you've finished entering it. Note: you should save the m-file in the current directory in which Matlab is running or else Matlab may not be able to find the m-file. To determine what directory Matlab is running in you can type 'pwd' from the Matlab command line.

Now we are ready to test the m-file. Type 'help generatesinewaves' at the Matlab command line and ensure that the help information is displayed properly.

» ***help generatesinewaves***

[x,z]=***generatesinewaves(n);***
Input:
n - the number of samples
Output:
x - a vector containing n samples of a single cycle of a sine wave
z - a vector containing n samples of two cycles of a sine wave

Plots of the sine waves are generated automatically.

Now test the by typing

[x,z]=***generatesinewaves(100);***

at the command line. If everything was entered correctly you should obtain a plot of the sine waves and the variables *x* and *z* should contain the samples. If there's a syntax error, you will get a detailed error message. For instance, suppose we misspelled the command 'sin' as 'sine':

x=sine((1:1:n)*2*pi/n);

The following error message is displayed in response to this syntax error

» [x,z]=***generatesinewaves(100);***
??? Undefined function or variable 'sine'.

Error in ==> C:\MATLABR11\work\generatesinewaves.m
On line 12 ==> x=sine((1:1:n)*2*pi/n);

which suggests that you should inspect line 12 and that you should check the validity of the call to 'sine'. The next section covers some debugging fundamentals.

What happens if we call 'generatesinewave' and forget to pass in the size *n*? What if *n* has a fractional part (say n=1.2) or is negative? Check the help information for the functions 'ceil' and 'floor' and see if you can add code to verify that *n* is a positive and has no fractional part. Also, check the 'error' function for a way of reporting errors detected in your m-files.

Debugging M-Files

As discussed in the last section, syntax errors will prevent a function from being invoked. A detailed error message will be generated. But logic errors are more difficult to find, they may allow a function to run until the logic error is encountered. There are several debug commands available to help you in debugging your m-file. Type 'help debug' at the command line now to get a listing of available debugging commands.

Perhaps the most useful debugging tool is the 'keyboard' command. This allows us to stop execution of an m-file at a particular location in a file and return control to the keyboard for inspecting the values of variables and for using the debug commands to step through the code.

As an example insert the 'keyboard' command in your m-file as shown below.

```
function [x,z]=generatesinewaves( n );
% [x,z]=generatesinewaves(n);
% Input:
% n - the number of samples
% Output:
% x - a vector containing n samples of a single cycle of a sine wave
% z - a vector containing n samples of two cycles of a sine wave
%
% Plots of the sine waves are generated automatically.

% Generate the vectors x and z
x=sin( (1:1:n)*2*pi/n );
z=sin( (1:2:n)*2*pi/n );

% Plot the results
keyboard;
plot(x);
xlabel('sample index');
ylabel('sample value');
title('Sine waves');
hold on;
plot(z,'red');
```

Remember to save the m-file after modifying it or your changes will not be instantiated. Now execute the function by typing '[x,z]=generatesinewaves(100)' at the command line. When execution gets to the 'keyboard' command, the Matlab command line should return the prompt

K»

Execution is suspended before the 'plot' function. You can use the command line to inspect the values of *x* and *z*. You can use the 'dbstep' command to execute the next line in the m-file. Experiment with the debug commands. The 'dbquit' will stop the debugger.

If you are using MEdit then debugging is even easier. You don't need to use 'keyboard' or the debug commands at all. Simply set your cursor next to the 'plot' function. Then click the *Debug\Set/Clear Breakpoint* menu item from the MEdit menu. Then run the m-file by typing '[x,z]=generatesinewaves(100)' at the Matlab command line. Execution will

stop at the ‘plot’ function. You can inspect the values of variables by placing the cursor over the variables in the m-file. If you return to the Matlab command line, you will see that you can also use the command line to inspect the values of variables. Use the “Debug” menu of MEdit to step through lines of code. See Figure 4.

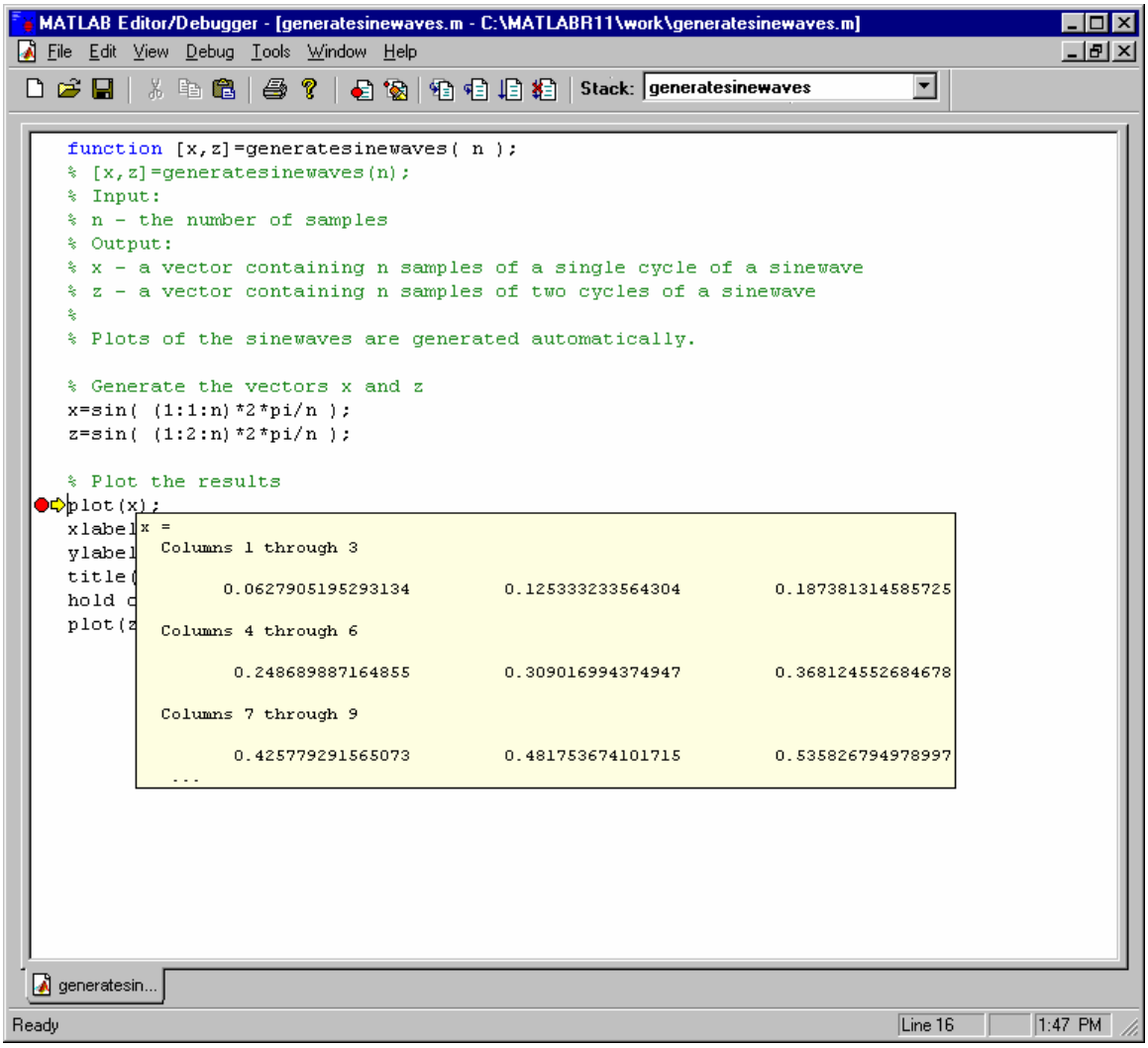

Figure 4 – An m-edit debugging session. Execution has halted at the breakpoint before the ‘plot’ command. The cursor is placed over the variable *x* which causes its contents to be displayed.

Reading/Writing text files

The syntax is almost the same than in the C Language. The basic functions in Matlab are:

Open a file:
file_id = fopen(file_name,'mode')

Read a line from a file:
data_line = fgetl(file_id);

Get the variables from the line of data:
variable = sscanf(data_line,'%f')

Read formatted data from a file:
array=fscanf(file_id,'%e%c',[4,inf])

Write formatted data to a file:
fprintf(file_id,' %s \n', text_variable)
fprintf(file_id,'%8.2f\t', data_variable)

Close a file:
fclose(file_id);

1. To show how you can save your data in a text file

```
x = 0:.1:1; y = [x; exp(x)];
        fid = fopen('exp.txt','w');
        fprintf(fid,'%6.2f  %12.8f\n',y);
% The special formats \n,\r,\t,\b,\f can be used to produce linefeed,
%  carriage return, tab, backspace, and form feed characters respectively.
%    Use \\ to produce a backslash character and %% to produce the percent
 %   character.

        fclose(fid);
```

2. Demonstrating how to read data from file and plotting data as figures:

```
filename=input('Input the filename:','s');
fid=fopen(filename,'r');
% 'r'     read
% 'w'     write (create if necessary)

array=fscanf(fid,'%e%c',[4,inf]);
% Use %c to read space characters

plot(array([1],:),array([3],:));
fclose(fid);
xlabel('Time(us)')
ylabel('Voltage(v)')
title('Waveform demonstration')
grid
```

Exercise:

1) Try creating a file(result.txt) with the cosgen function's result in it. Then open the file (result.txt) and plot and print it.