

UNIVERSITY OF WATERLOO

Introduction to Scientific Computing with MATLAB

SAW Training Course

MATLAB R2009b

R. William Lewis

Computing Consultant

Client Services – Information Systems & Technology

rwlewis@uwaterloo.ca

February 2010

Table of Contents

1	MATLAB Basics	1
1.1	Obtaining software	1
1.2	Research License, Classroom license	1
1.3	University-owned Machines	1
1.4	Signing up for the Research License	1
1.5	Troubleshooting / Problems Running MATLAB	1
1.6	MATLAB Versions	1
1.7	Running MATLAB	1
2	Desktop Environment	2
2.1	Starting up MATLAB	2
2.1.1	Command Window	3
2.2	Command history	3
2.3	Current Folder	4
2.4	Workspace	4
2.5	Editor	5
2.6	Matrices, Arrays	5
2.6.1	Dimensions, contents	5
2.6.2	Ranges (vectors)	5
2.6.3	Manipulating Elements	6
2.6.4	Building up matrices	6
2.6.5	Matrix operations	6
2.6.6	Array operations	7
2.7	Assigning entries, rows, columns	7
2.8	Deleting entries, rows and columns	7
2.9	Functions	7
2.9.1	Function, operand, result	7
2.9.2	Dimensions of results	7
2.9.3	Example Statistical Summary Functions	8
2.10	Logical Indexing, Finding Elements	8
2.11	Exercises	9
2.11.1	Define the following matrices	9

2.11.2	Performing matrix and array operations	9
2.11.3	Apply functions to the matrices you have defined	9
2.11.4	Investigate the “find” command	9
3	Getting Help	10
3.1	From the Command Window	10
3.2	Function Browser	10
3.3	Help Menu	11
3.4	Help Window	11
3.5	Examples	12
4	MATLAB editor	13
4.1	Basics	13
4.1.1	Starting editor	13
4.1.2	Source navigation	13
4.1.3	Saving	13
4.1.4	Splitting the editor Window	13
4.2	Running your script	13
4.3	Function in an m-file	13
4.4	Debugging Commands	14
4.5	Development Methodology	14
4.5.1	Experiment at command line, put working commands into script	14
4.5.2	Identify parameters that will change and make then function arguments	14
5	Data Analysis Example	15
5.1	Importing data into MATLAB	15
5.2	Import Data interactively	15
5.3	Plot the Data	16
5.4	Import Data in a script	17
5.4.1	Exercise	18
5.4.2	Exercise: Write a MATLAB script main.m	18
6	Plotting	19
6.1	Creating Plots Interactively	19
6.2	Format the Plot	20
6.3	Create script file	22

6.4	Exercise: Use this formatting in your script	22
7	Fitting a Model to the Sample Data	23
7.1	Polynomial Fitting	23
7.2	Use polyval to evaluate the fitting polynomial	23
7.3	Plot the fit	23
7.4	Use a Different Model	24
7.5	Performing the Non-Linear Fit	24
7.6	Exercise	24
8	Three-Dimensional Surface Plot	26
8.1	Function of two variables.....	26
8.2	Data grid.....	26
8.3	Evaluate Function.....	26
8.4	Plot with mesh	26
8.5	A Better Plot.....	27
8.6	Explore the Graph	28
9	Programming	29
9.1	MATLAB editor	29
9.2	Methodology (test commands, add to script)	29
9.3	Testing, debugging, spying as it's running	32
9.4	Exercises.....	33
9.4.1	Fix the function so it can plot a parabola with complex roots.....	33
9.4.2	Handle parabolas with a double root	33

1 MATLAB Basics

1.1 Obtaining software

MATLAB is available on many student labs on campus, including NEXUS labs. For research use, faculty and staff can purchase a license for themselves and for graduate students. You can also purchase the student version directly from The MathWorks.

1.2 Research License, Classroom license

There are two site licenses for MATLAB on campus. The research license requires payment of a yearly fee and is licensed per person. The classroom license cannot be used for research purposes, and is used in instructional labs. If you are using MATLAB for a course, you should be using the classroom license. If you are doing academic research, you should be using the research license, and are prohibited from using the classroom license. Commercial use requires a separate license directly from The Mathworks.

1.3 University-owned Machines

Our site licenses are both restricted to university-owned machines. Faculty and staff (only) are allowed to use MATLAB on a home machine. Graduate students are limited to university-owned machines.

1.4 Signing up for the Research License

To buy an annual license for MATLAB, visit <https://strobe.uwaterloo.ca/ist/saw/webstore/>. Only University of Waterloo accounts (AFFs) are accepted for payment. The account holder will need to login with their UW user ID and password and provide the user name for each person who needs access.

1.5 Troubleshooting / Problems Running MATLAB

If you run into any problems with MATLAB, remember three things:

1. Your login name must match your UW user ID (entered when your license was purchased).
2. You must have network access to run MATLAB. On wireless, you must run MinUWet.
3. Purchases do not take effect immediately, and there are short periods of downtime on the license servers during the day while new licenses are loaded.

1.6 MATLAB Versions

There are two MATLAB releases each year. This course is based on R2009b released September 1, 2009. MATLAB and each of its toolboxes have individual version numbers as well. Release R2009b corresponds to MATLAB 7.9. Users of our site licenses are entitled to use new versions when they are released.

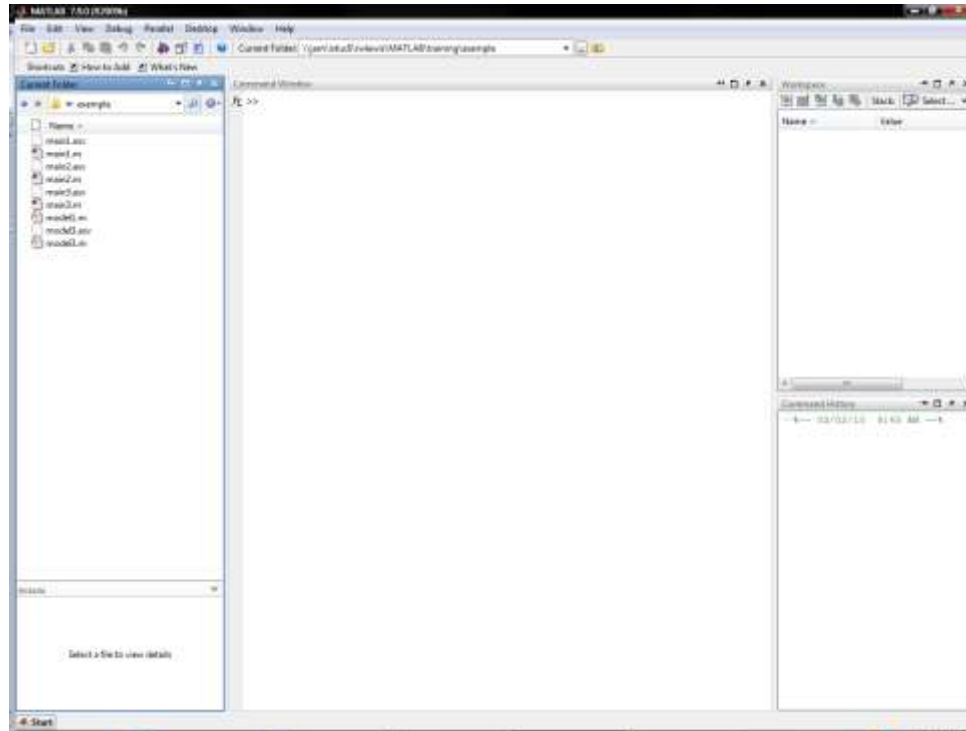
1.7 Running MATLAB

Installing MATLAB creates a program group in the Start Menu. In the IST training labs and in NEXUS labs, you will find MATLAB under Start | All Programs | Math & Statistical | MATLAB.

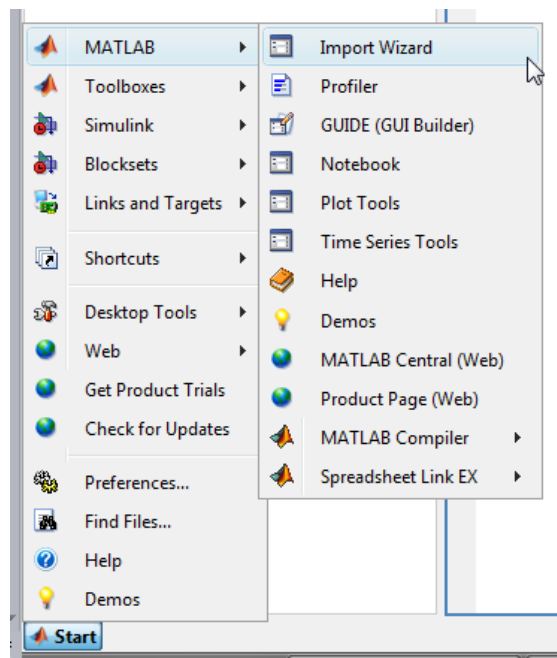
2 Desktop Environment

2.1 Starting up MATLAB

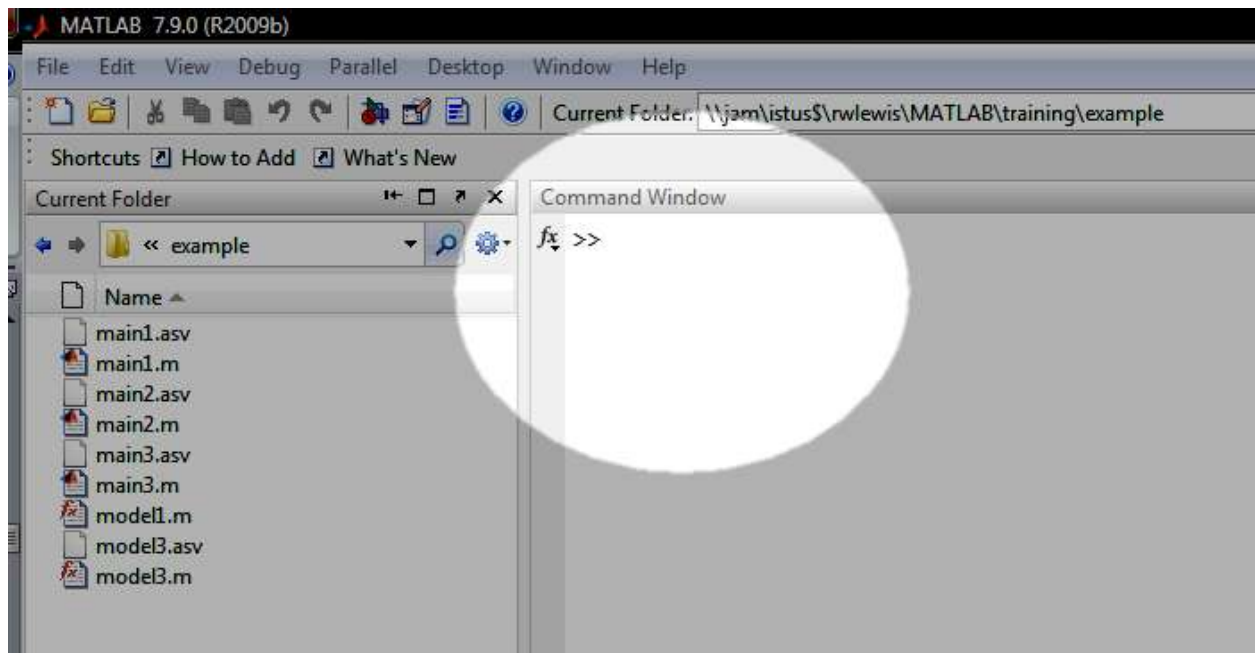
MATLAB uses a three-column layout as the default.



MATLAB has its own Start menu, which gives quick access to a number of graphical tools.



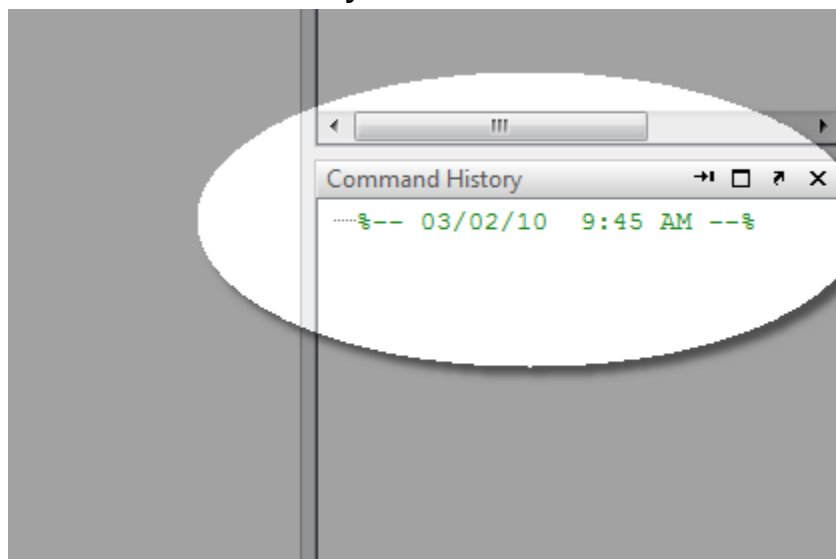
2.1.1 Command Window



Your first interaction with MATLAB will be through the Command Window, which shows up in the middle of the MATLAB window. This is where you can type MATLAB commands and view the output of these commands. In these notes, sample code will be in a box as shown below. For your first command try “ver” which tells you which MATLAB features are installed.

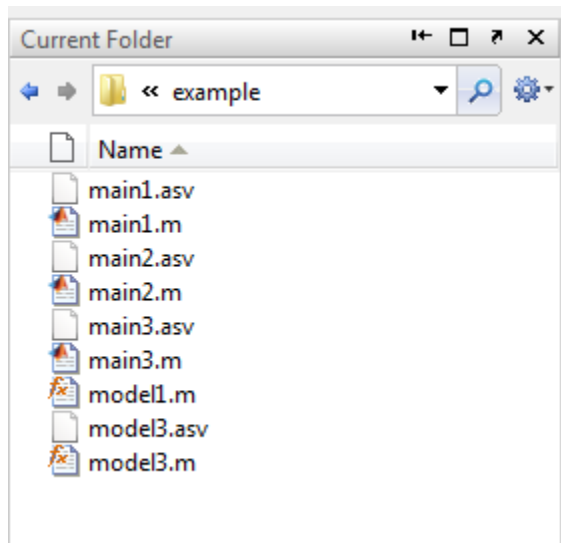
```
ver
```

2.2 Command history

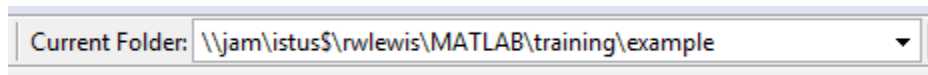


In the lower right corner of the MATLAB window is a section called “Command History” which will store a record of the commands you give to MATLAB. You can double click on a command to repeat it, or right-click on a command to copy it, create an M-file, and other options.

2.3 Current Folder



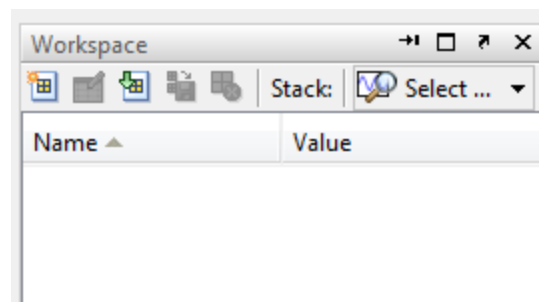
The left side of the MATLAB window shows the files in the current folder. In this interface you can look in different folders for files. To change the working directory, double-click on the folder you want. You can also use the double-arrow to the left of the folder name to move up the folder tree. The current working directory is also shown in the toolbar.



MATLAB uses a path setting to determine where it looks for code to execute. This path includes the toolboxes that come with MATLAB but might not necessarily include the places you save your files. MATLAB always sees the current directory, and this section lets you see the current directory and change it. Use the path command to see where MATLAB is looking for files.

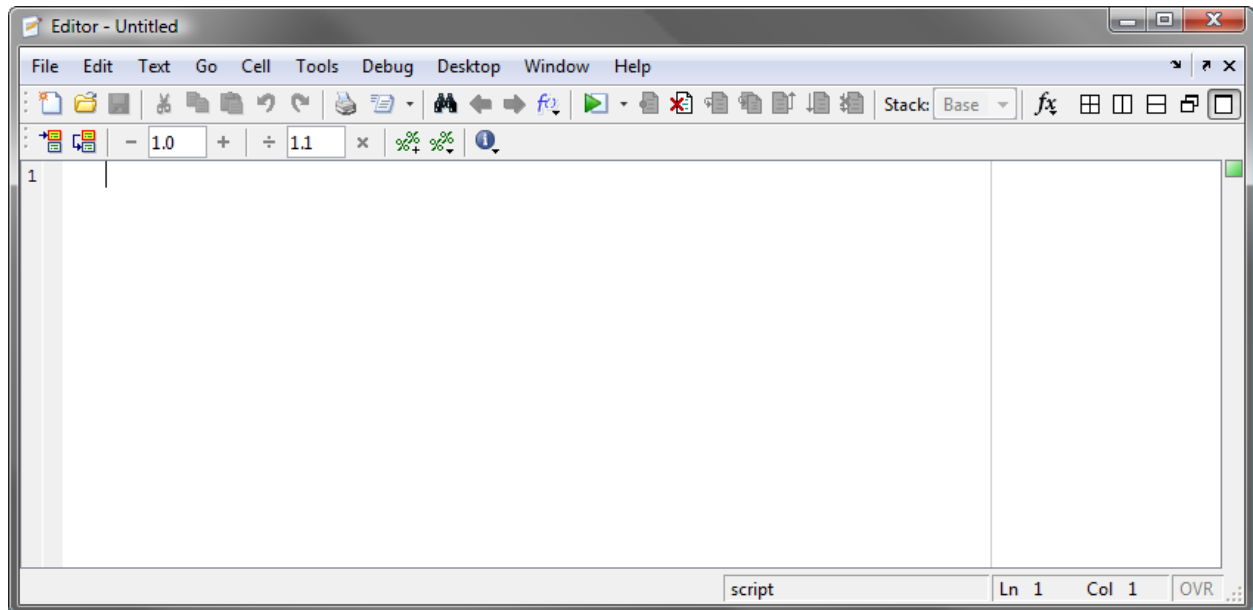
path

2.4 Workspace



The right hand side of the interface shows an area called the Workspace. The Workspace shows all of the variables that are currently defined in your MATLAB session. You can view and change these variables as well as access an array editor from the Workspace tab by double-clicking on an array.

2.5 Editor



Opening a MATLAB m-file gives you access to the MATLAB Editor, which is a text editor with some special features to make it useful for working with MATLAB code.

2.6 Matrices, Arrays

The fundamental data structure in MATLAB is an array. This will be the main data structure that you work with, and it will help to remember that even a variable with a single value is a 1x1 array in MATLAB.

2.6.1 Dimensions, contents

Each array has dimensions, which are specified with the standards of “n by m” or “n x m” meaning “n rows” by “m columns”. All elements of the array have the same data type. You can specify a matrix by typing it between square braces [] with elements separated by spaces and rows separated by semicolons.

```
A = [1 2 3; 4 5 6; 7 8 9]
```

2.6.2 Ranges (vectors)

To indicate a range in MATLAB, you use the colon. To specify the increment, use the syntax start:increment:end. The increment, start and end can all be decimal numbers.

```
1:4  
1:3:20  
.5:.2:1.4
```

Notice the final range will end at 1.3, the largest number in the pattern specified that is smaller than the request end point.

2.6.3 Manipulating Elements

In MATLAB you can address an individual element of a matrix by using round brackets. For example, to obtain the item in the 3rd row and 4th column of a matrix A, type A(3,4). The subscripts can be ranges. To specify the last element in a row or column use “end”.

```
A = magic(5)
A(1:3, 2:4)
A(3:end, 4)
```

MATLAB also allows linear indexing by starting with the first column then moving through each column.

```
A = magic(5)
A(2)
A(20)
```

2.6.4 Building up matrices

To type in a matrix at the command line, separate columns with spaces and rows with semicolons:

```
A = [1 2 3; 4 5 6; 7 8 9]
```

Built-in matrices can be generated in a number of ways:

```
zeros(3)
zeros(3,4)
diag([1 4 6 10])
```

You can use built in matrices and ranges to build up a bigger matrix by separating columns by a space and rows by a semicolon:

```
A = [zeros(3) ones(3)]
A = [magic(3); ones(3)]
A = [zeros(3) ones(3); magic(3) [1:3; 2:4; 3:5]]
```

2.6.5 Matrix operations

It is important to draw a difference between a matrix and an array. Arrays don't have a strict mathematical algebraic structure; they are simply numbers in a grid. Operations on arrays are done element-wise. Matrices are a special type of mathematical object that have their own algebra.

In MATLAB, standard mathematical symbols mean matrix operations.

```
A*B
B/A
```

2.6.6 Array operations

To make a MATLAB operation work element-wise, use the standard mathematical symbol with a period before it. Note that addition is the same for matrices and arrays.

```
A .* B  
B ./ A
```

When dimensions don't match, MATLAB tries to perform scalar expansion. You can add a scalar to an array to add the number to each element in the array.

```
A = magic(3)  
A + 1  
2*A
```

2.7 Assigning entries, rows, columns

You can assign a value to an individual entry of a matrix, or to an entire row or column using the colon syntax.

```
A = magic(3)  
A(1) = 3  
A(1, 2) = 3  
A(1, :) = 2  
A(:, 1) = 1
```

2.8 Deleting entries, rows and columns

The general syntax to delete parts of matrices is to assign the empty matrix to them.

```
A(1, :) = []  
A(:, 1) = []
```

2.9 Functions

The commands just used to produce built-in matrices are examples of MATLAB functions.

2.9.1 Function, operand, result

Functions take a list of parameters and provide a result. MATLAB functions can take one or more operands of various types, some functions can handle variable numbers of arguments (e.g. `zeros(3)` or `zeros(3,3)`), some require a fixed number of arguments. Functions can also return more than one value, and this is usually accomplished by assigning the result of the function to a vector.

2.9.2 Dimensions of results

Most MATLAB functions can take matrix arguments and return results as a matrix. In general it is much faster to operate on matrices in MATLAB than to operate on each element of the matrix in term. This is called "vectorizing" your code.

2.9.3 Example Statistical Summary Functions

The following statistical functions work on each column of a matrix.

sum	Sum (total) of each column
mean	Average (mean) of each column
max	Maximum of each column
min	Minimum of each column
median	Median
mode	Mode
std	Standard deviation
var	Variance

A useful trick to get one of these functions to operate on all elements of a matrix is to use linear indexing. If you use the colon as a linear index, MATLAB converts the matrix to a vector. Combined with a statistical function gives a compact way to operate on the entire matrix rather than on each column:

```
sum(A)
sum(A(:))
```

2.10 Logical Indexing, Finding Elements

MATLAB can easily provide you with all elements of a matrix that meet a specified condition.

```
A = magic(4)
A(A>10)
```

In this example, the result of `A>10` is a matrix, consisting of zeroes and ones, indicating which elements are greater than 10. The elements that meet the condition are identified by ones and those that do not meet the condition are identified by zeroes.

```
>> A>10

ans =

     1     0     0     1
     0     1     0     0
     0     0     0     1
     0     1     1     0
```

This matrix is then used as an index to the matrix `A`. Because it is the same size as `A`, it indicates whether or not to display the particular element of the matrix. If you need to know where these elements are located, use “find” which tells you the location of the ones. With one return value, find uses linear indexing. With two return values, find gives lists of rows and columns.

```
find(A>10)
[rows, cols] = find(A>10)
```

2.11 Exercises

2.11.1 Define the following matrices

$$A = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 2 & 3 \\ 0 & 1 & 2 \end{bmatrix}$$

$$B = \begin{bmatrix} -1 & 2 \\ 1 & 4 \\ 1 & 2 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \\ 3 & 0 & 0 & 4 & 4 & 4 \\ 0 & 2 & 0 & 4 & 4 & 4 \\ 0 & 0 & 1 & 4 & 4 & 4 \end{bmatrix}$$

Note that C is made up of four matrices with special structures. The commands ones, eye and diag will be helpful.

2.11.2 Performing matrix and array operations

Try commands like:

```
A*B  
C * [A; A]
```

To compute array operations you will need to produce same-sized matrices. Try:

```
B(3,3) = 10  
A .* B
```

2.11.3 Apply functions to the matrices you have defined

For example,

```
sum(A)  
max(C)
```

2.11.4 Investigate the “find” command

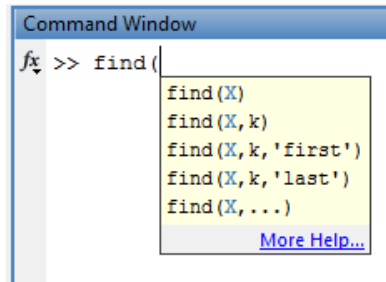
The find command can return row and column indices instead of linear indices by assigning its result to a matrix of two variables:

```
[i, j] = find(A>1);
```

3 Getting Help

3.1 From the Command Window

After you type in the name of a MATLAB command followed by an open bracket, MATLAB will provide syntax hints for the command.



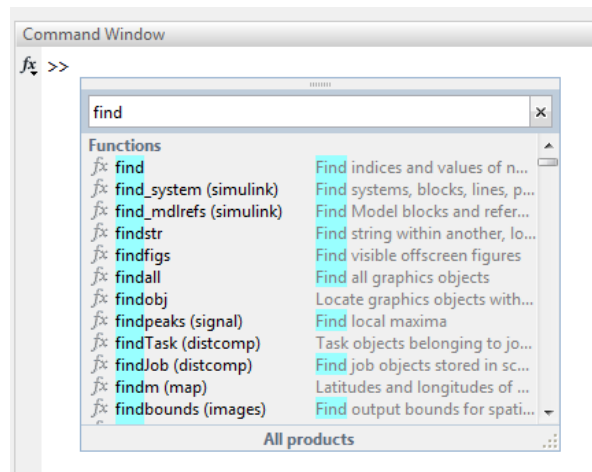
You can then click on “More Help...” to access the help topic for find, in a minimal interface.



There will now be a link to the main “Help Browser” which is MATLAB’s full help interface. You can get directly to the Help Browser through the Help menu as well.

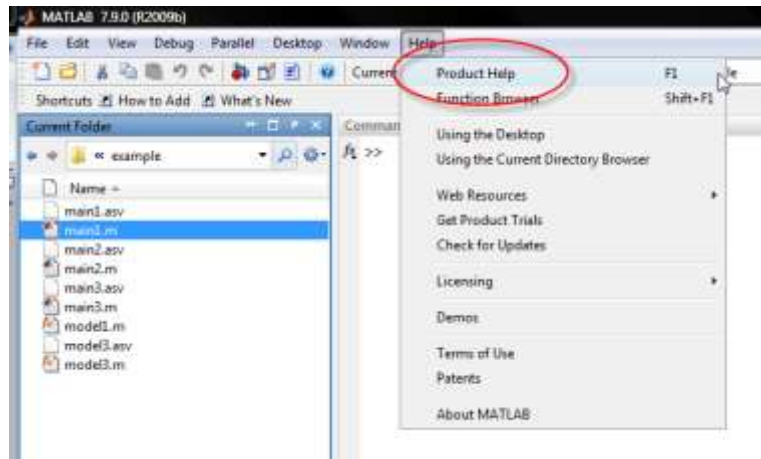
3.2 Function Browser

If you don’t know the name of a function, you can press SHIFT+F1 at the command window to open a simple keyword search interface. Type a word and MATLAB will give a list of matching commands.



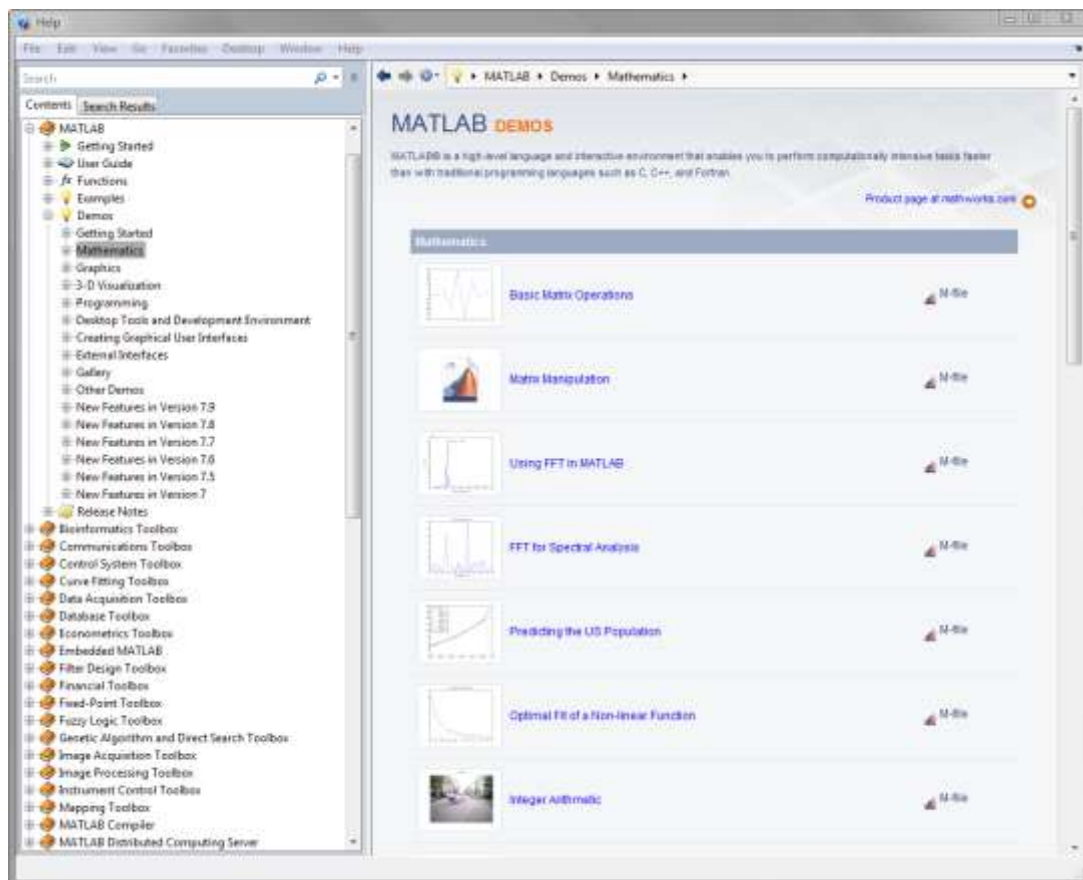
3.3 Help Menu

To access the MATLAB help system, use the Help menu. Most of the menu options take you to different places within the help browser.



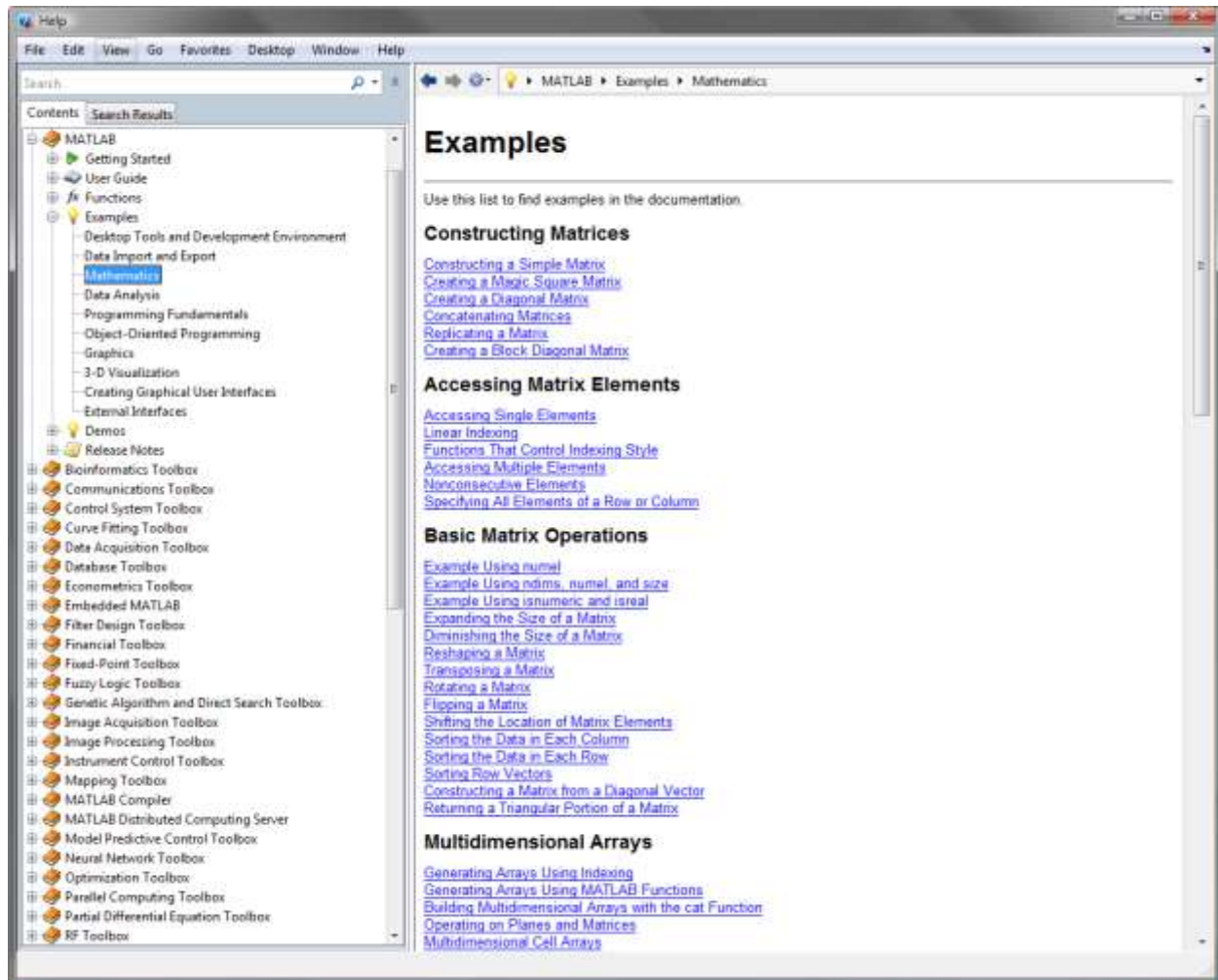
3.4 Help Window

MATLAB Help takes you to the help window. Browsing through the topics is a good way to learn about MATLAB and to learn about new functions and toolboxes. A good way to learn more about MATLAB is to use the included Demos. You will see a section for Demos within any of the books in the help system.



3.5 Examples

The help system also has lists of all of the examples used in the documentation. This can be another good way to find useful information about MATLAB.



4 MATLAB editor

4.1 Basics

As you use MATLAB, you may find yourself repeating sequences of commands. Or you may develop a way of analyzing certain data that you need to repeat as new data become available. In order to keep an accurate record of MATLAB commands, and to make your sessions repeatable, you can put MATLAB commands into M-files, which are simply text file with the extension “.m”. You can also use M-files to define your own functions. MATLAB includes an editor which is tailored to editing MATLAB code.

4.1.1 Starting editor

The editor is opened by default when you open an M-file. You can create a new M-file by choosing the option New | M-file from the Current Directory context menu (accessed by right-clicking) or from the File menu. You can also invoke the editor directly from the Desktop menu.

4.1.2 Source navigation

The MATLAB editor shows line numbers on the left to help navigate within your code. You can include comments in your M-file by starting a line with a % (percent) symbol.

4.1.3 Saving

Until you save your file, MATLAB will continue to run the old version on disk. The MATLAB editor tells you there are changes by adding a star * to the file name. Make sure you save before you run your code.

4.1.4 Splitting the editor Window

In the Window menu there are several options to Tile your Editor windows. Tiling lets you view several files at the same time, and is very useful when working on larger projects. Tiling is especially useful if you keep your line lengths short.

4.2 Running your script

You can run an M-file by typing its name at command line in the Command Window. Or you can choose Debug | Run (or click on the Run icon in the toolbar).

4.3 Function in an m-file

A script M-file executes a sequence of commands. Variables defined in your workspace are available to the script, and variables defined by the script become part of the workspace. While this can be useful, it can also cause problems if your scripts are used in different contexts. To formalize what information is passed to and from your M-file, use MATLAB functions. This will make your M-file readable and reusable. You can also write short special-purpose functions that are simpler to code and debug.

To define a function called myfunction which takes input a, b and c and returns output A:

```
function A=myfunction(a, b, c)
    ... Commands go here
end
```

The lines within the function should assign the value to be returned into A. I recommend that you terminate your function with the “end” command and indent the commands within your function for greater clarity. The function must have the same name as the M-file. Note that when you create a new M-file by right-clicking in the Current Directory tab, MATLAB provides you with a template function definition.

4.4 Debugging Commands

A simple way to check the commands in your M-file is to copy them to the Command Window and make sure they work as expected. Select a command, right-click and choose “Evaluate selection”. You can also use “Copy” and paste the command to the Command Window to edit it and try variations on it.

Since your commands will likely not work in isolation, you will often need to select all of the commands that set up the environment. Execute the entire selection by right-clicking and “Evaluate selection”.

For more sophisticated debugging, use the commands in the Debug menu. With a Breakpoint, you specify a place in your code where MATLAB will stop when the code runs. You can then view the Workspace and see what has been defined, and use Debug | Step to run your code one line at a time. It is also possible to set MATLAB to automatically launch the debugger when it encounters an error.

4.5 Development Methodology

In compiled languages like C, it is common to develop code by writing portions, then compiling and running them to make sure they work. In MATLAB, you can develop your code interactively at the command line to confirm syntax and proper operation before you put them in your M-file.

4.5.1 Experiment at command line, put working commands into script

Working at the command line lets you try variations on commands and experiment with the syntax of commands you have not used before. By checking the output interactively you will find errors and mistakes immediately and be able to put well-formed working commands into your M-file.

4.5.2 Identify parameters that will change and make them function arguments

To write good functions, you should identify which parameters may change and define them as arguments to your function. In this way you can reuse the function simply by providing different parameters. Your code will also be clearer and easier to maintain.

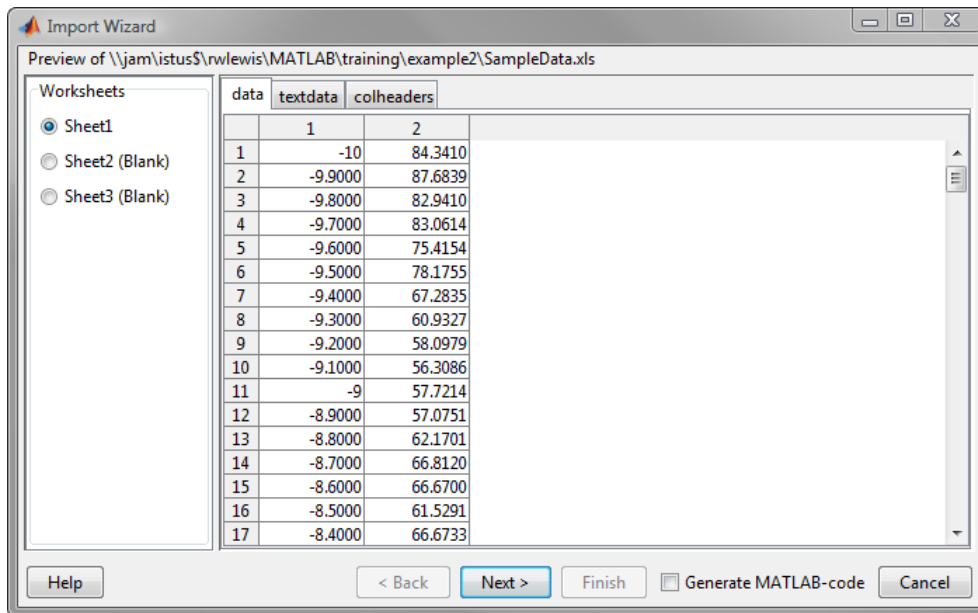
5 Data Analysis Example

5.1 Importing data into MATLAB

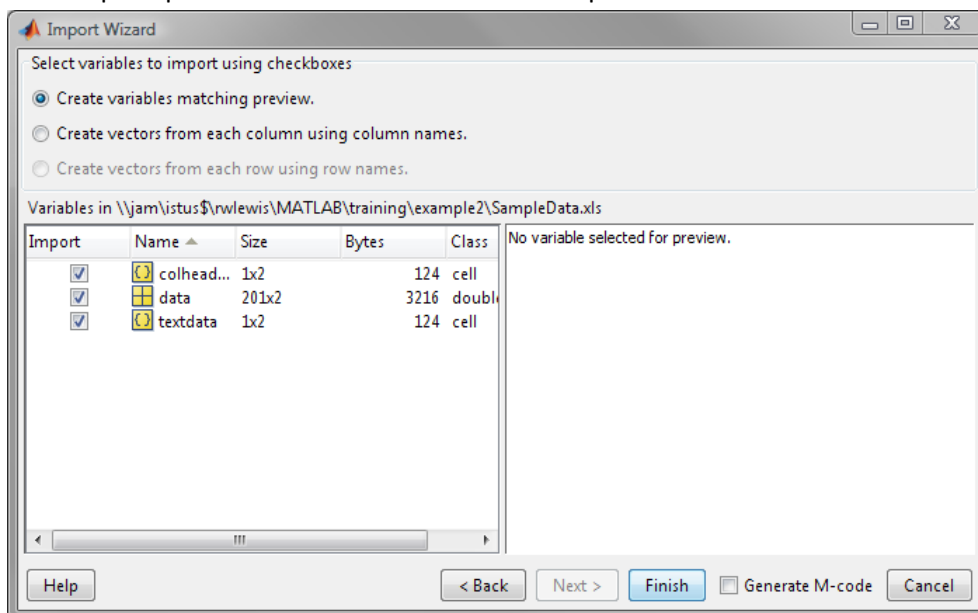
We will analyze some data to find parameters in specific models. The data are stored in an Excel spreadsheet called “SampleData.xls”. The simplest way to import an Excel spreadsheet into MATLAB is to double-click on the file in your Current Directory tab. We will then automate this process into a script.

5.2 Import Data interactively

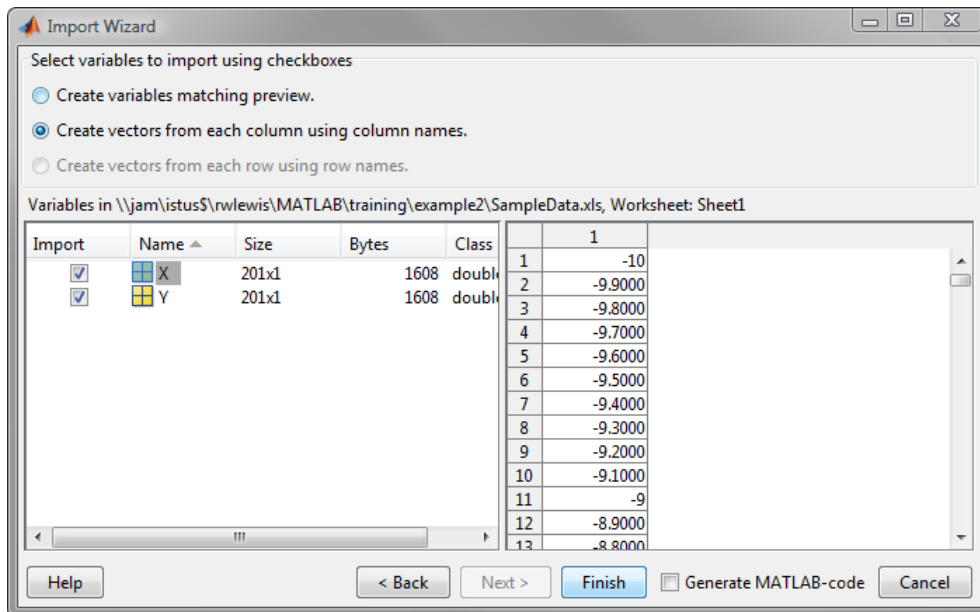
1. Double-click on the file “SampleData.xls”.
2. [MATLAB R2009b and later only] The Import Wizard opens with a preview of the spreadsheet:



3. Click Next.
4. You are prompted to select which variables to import:



5. The default settings will create a single two-column matrix with all of the data. This spreadsheet is formatted with column headings, so MATLAB is able to automatically determine its structure.
6. Choose the second option, “Create vectors from each column using column names”.
7. MATLAB will create variables called X and Y to match the column names.



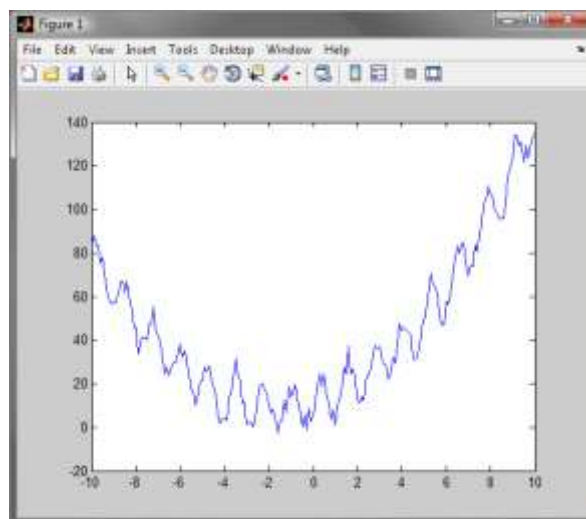
8. Check off “Generate MATLAB-code”.
9. Click on Finish.

Because we selected “Generate MATLAB-code” the editor opens with a new MATLAB M-file. We’ll ignore the generated code for now, and move to the command windows to see what has happened.

5.3 Plot the Data

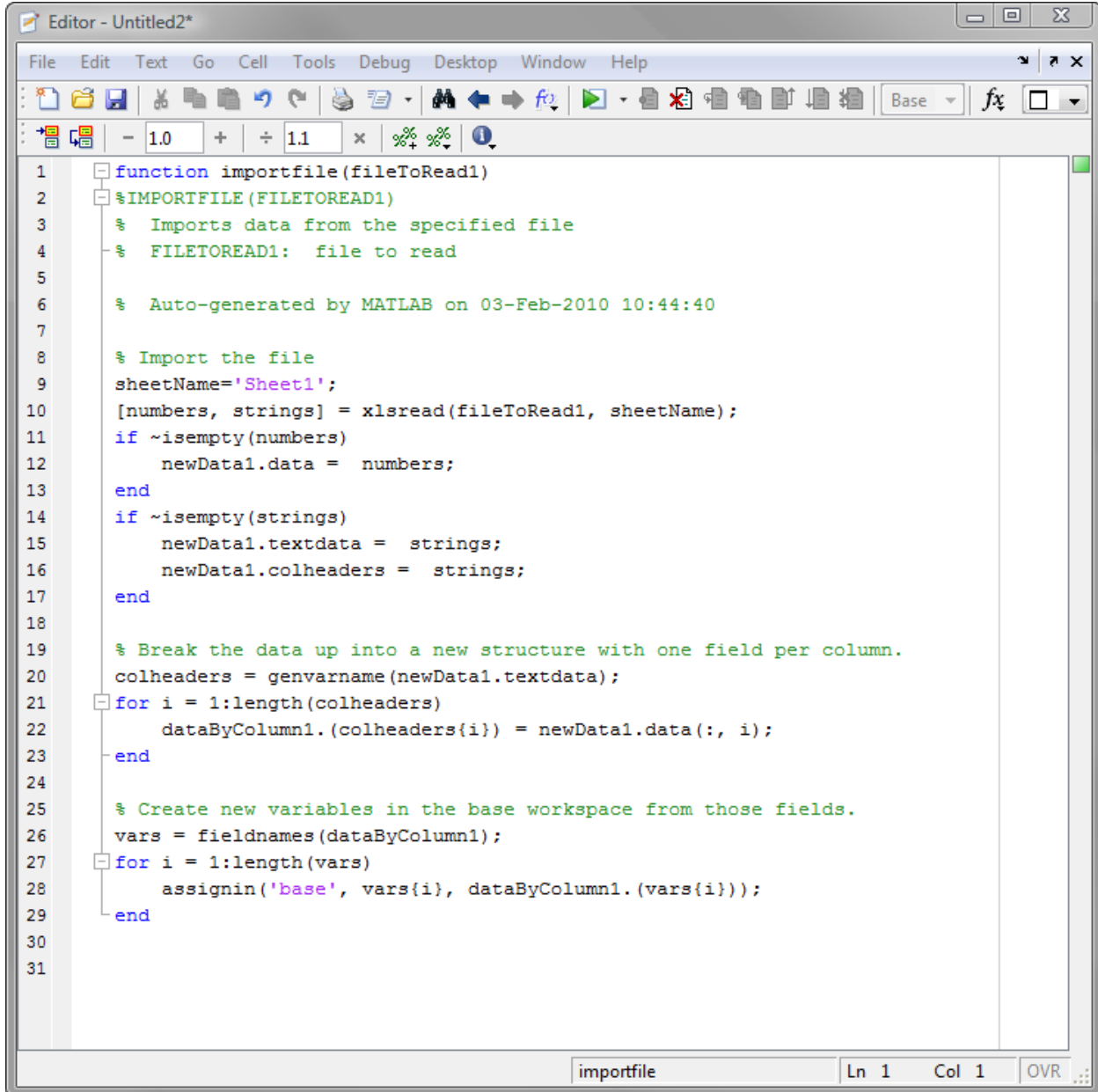
To take a quick look at the data, use the command:

```
plot(X, Y)
```



5.4 Import Data in a script

In many applications, it is necessary to repeat the same methodology with different sets of data. Even if the data are the same, it may be necessary to try different approaches. Consequently, it is beneficial to automate the process. Return to the MATLAB editor to see the code generated by the Import Wizard.

The image shows a screenshot of the MATLAB Editor window titled 'Editor - Untitled2*'. The window has a menu bar (File, Edit, Text, Go, Cell, Tools, Debug, Desktop, Window, Help) and a toolbar with various icons. Below the toolbar is a numeric keypad with values like 1.0, 1.1, and mathematical symbols. The main area contains a MATLAB function script for 'importfile'. The script starts with a function definition and a comment block. It then uses 'xlsread' to load data from a file, checks for empty cells, and processes the data into a structure. Finally, it creates variables in the base workspace from the structure's fields. The script is 31 lines long. The status bar at the bottom shows 'importfile', 'Ln 1', 'Col 1', and 'OVR'.

The details of this file are not particularly important to us at this point. The key is that running this function and providing it the file name for our data will reproduce what we did in the Import Data wizard.

```
importfile('SampleData.xls')
```

5.4.1 Exercise

1. Run the Import Wizard with the option to generate M-code.
2. Choose File | Save.
3. Call the file `importsampladata.m`.
4. MATLAB now underlines the function name in the first line to notify us of a discrepancy. MATLAB will ignore the name in line 1 and instead use the file name. It is recommended to keep these names consistent, change line 1 to use the function name **`importsampladata`**.
5. Save the file and close it.
6. In the command window type

```
importsampladata('SampleData.xls')
```

7. Notice that MATLAB overwrites (or creates) variables X and Y.

5.4.2 Exercise: Write a MATLAB script `main.m`

1. Create a script called `main.m` with the commands:

```
importsampladata('SampleData.xls')  
plot(X, Y)
```

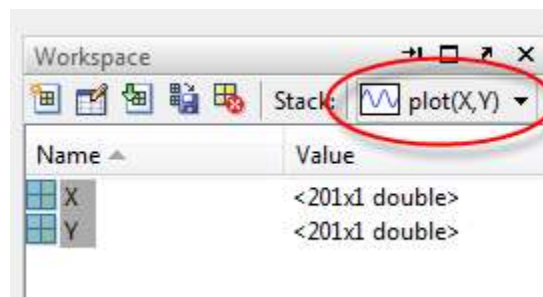
6 Plotting

6.1 Creating Plots Interactively

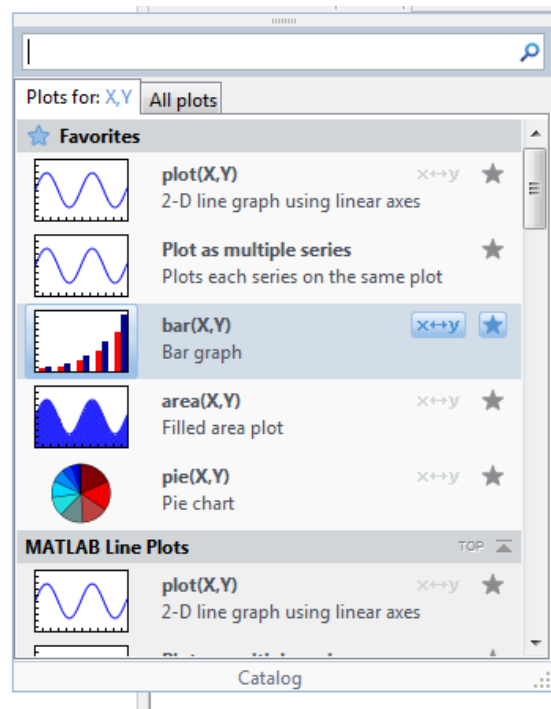
Once you have data in your workspace, you can use either the MATLAB GUI (graphical user interface) or MATLAB commands like “plot” to create plots. Run your main.m script from Chapter 4 to create the variables X and Y in the workspace and show the result of a plot command. For now we’ll close the Figure window and create a new one using other methods.

```
main
```

Click in the Workspace tab, select the X and Y variables and you’ll see that MATLAB offers “plot(X,Y)” as a suggested plotting command.



Click the arrow next to the graph icon to access MATLAB’s Plot Catalog.



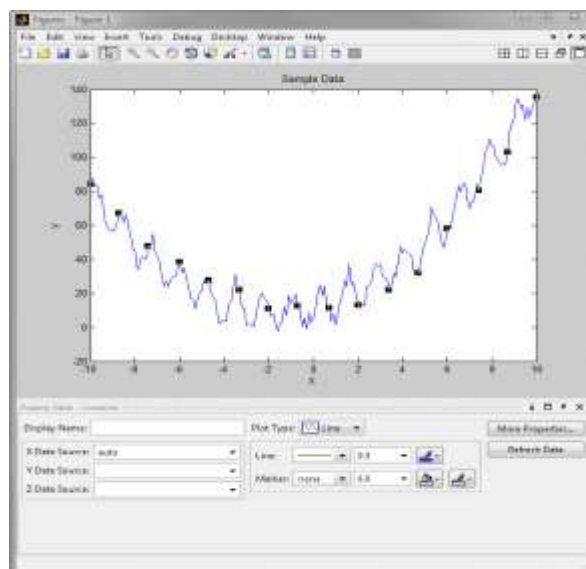
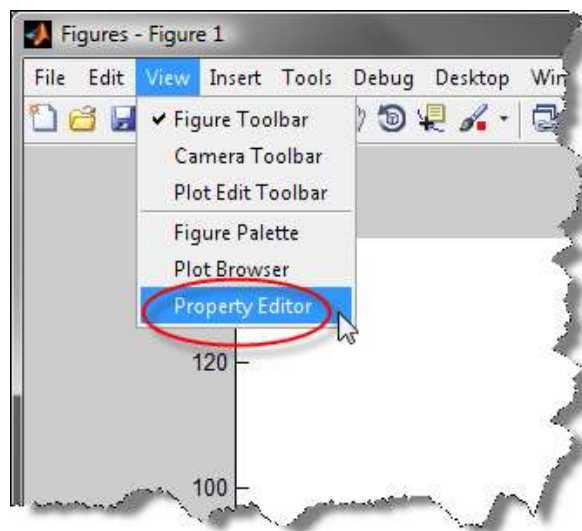
MATLAB opens a figure window with the requested plot.

We'll start with the simple plot:

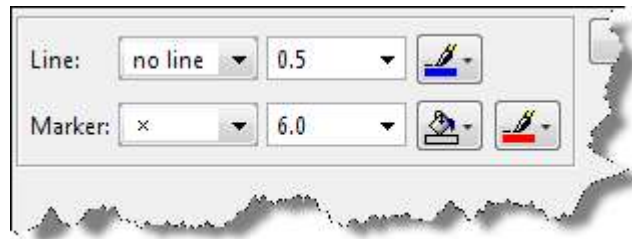
```
plot(X, Y)
```

1. On the Insert menu, choose X-label, type "X"
2. On the Insert menu, choose Y-label, type "Y"
3. On the Insert menu, choose Title, type "Sample Data"

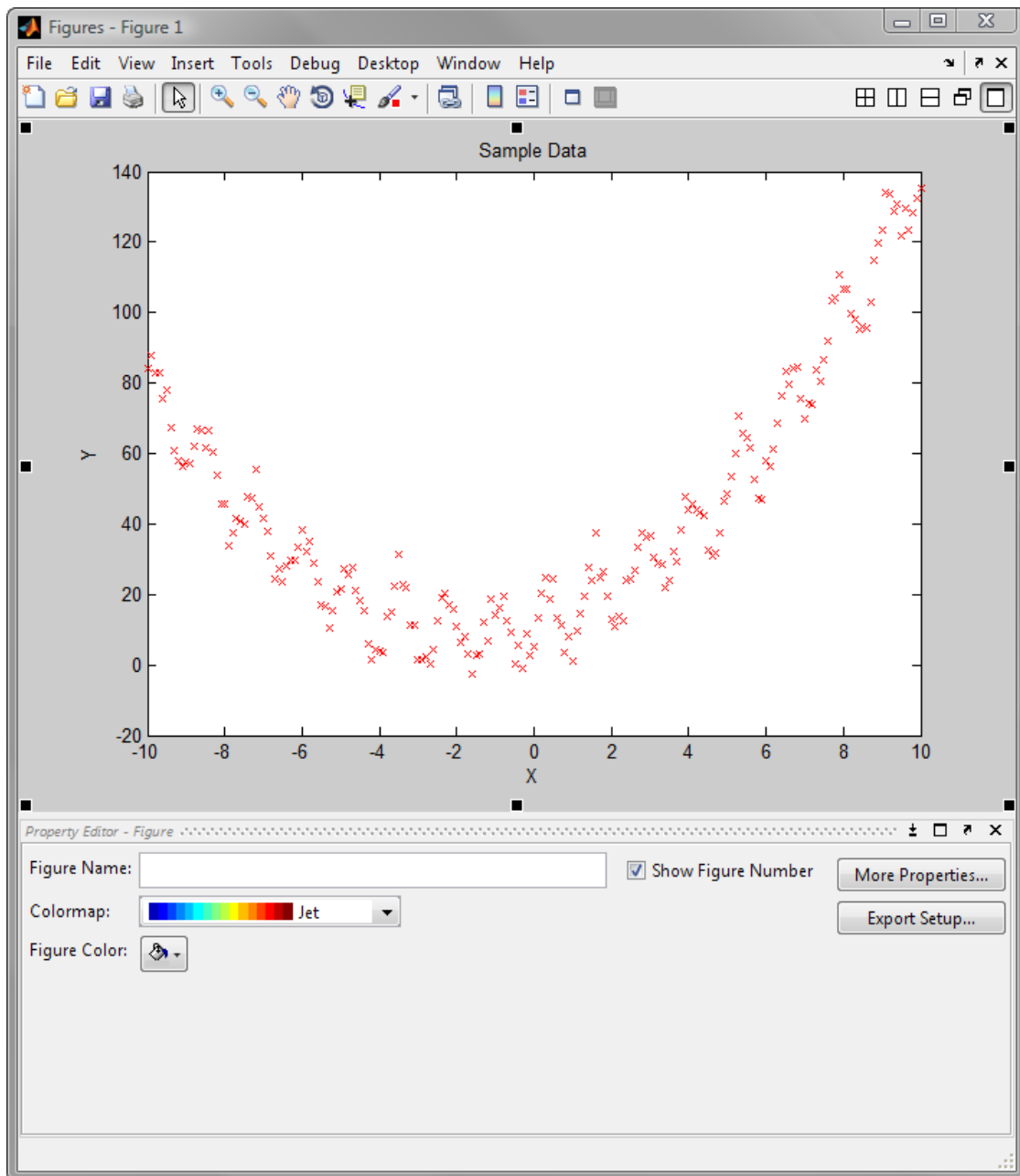
Downloaded from <http://ajphaphysocpharm.sagepub.com> at 11:01 11 November 2014



We'll remove the lines and change the markers to red X's.



The resulting plot looks like this:

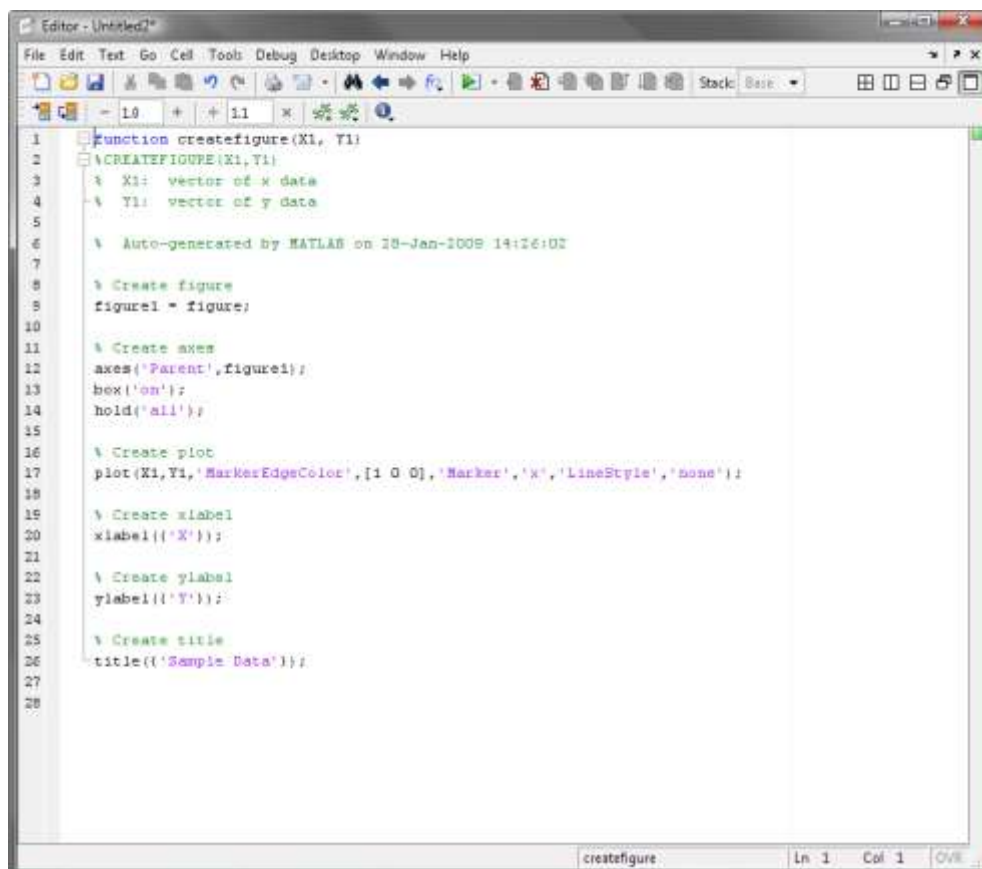


6.3 Create script file

Now we would like to reproduce this formatting every time the data are imported. Choose File | Generate M-file. MATLAB generates the code needed to reproduce your formatting.



MATLAB creates a new M-file which will reproduce your graph.

A screenshot of the MATLAB Editor window showing the code generated by MATLAB for the 'createfigure' function. The code is as follows:

```
1 function createfigure(X1, Y1)
2 %CREATEFIGURE(X1,Y1)
3 % X1: vector of x data
4 % Y1: vector of y data
5
6 % Auto-generated by MATLAB on 18-Jan-2009 14:16:02
7
8 % Create figure
9 figure1 = figure;
10
11 % Create axes
12 axes('Parent',figure1);
13 box('on');
14 hold('all');
15
16 % Create plot
17 plot(X1,Y1,'MarkerEdgeColor',[1 0 0],'Marker','x','LineStyle','none');
18
19 % Create xlabel
20 xlabel('X');
21
22 % Create ylabel
23 ylabel('Y');
24
25 % Create title
26 title('Sample Data');
```

6.4 Exercise: Use this formatting in your script

Copy the code from the generated m-file into your script in order to reproduce the formatting that we created interactively. You will need to make sure that the variable names are consistent.

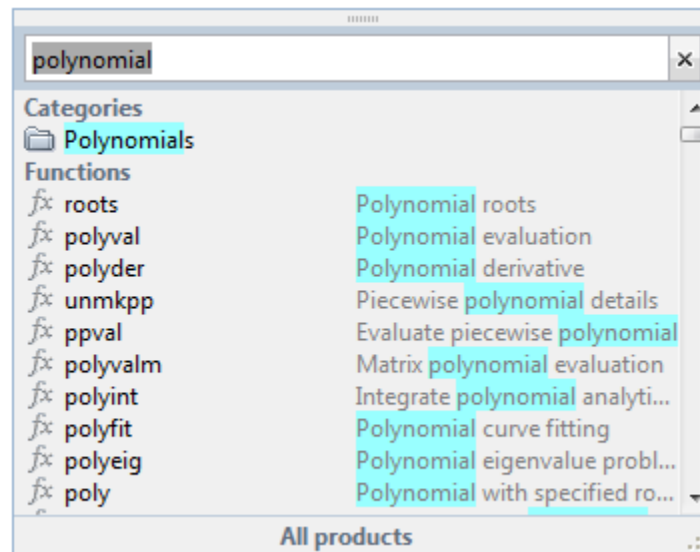
7 Fitting a Model to the Sample Data

7.1 Polynomial Fitting

MATLAB has built in routines for polynomial fitting that will work well for the sample data we have created. We will use the function `polyfit` which takes the `x` and `y` values as arguments. It also requires a third argument which is the degree of polynomial to use for the fit.

```
p = polyfit(X, Y, 2);
```

The result of the `polyfit` command is a vector of polynomial coefficients. MATLAB has a number of built-in functions for dealing with polynomials in this form. To learn more about functions for working with polynomials, push `SHIFT+F1` and type `polynomial`.



7.2 Use polyval to evaluate the fitting polynomial

In order to compare the fitted polynomial to the data, use the function `polyval` to evaluate the polynomial at the points in the vector `X`.

```
values = polyval(p, X);
```

7.3 Plot the fit

We can then superimpose the fitted polynomials onto the sample data plot. In order to have MATLAB reuse the figure window and keep the old graph, use the command `hold on`.

```
hold on  
plot(X, Y, 'color', 'b');  
hold off
```

7.4 Use a Different Model

For a more complicated model, it will be necessary to use our own function to tell MATLAB how to calculate the data values.

Let's consider a model with five coefficients and a sinusoidal term:

$$y = c_1x^2 + c_2x + c_3 + c_4\sin(c_5x)$$

We'd like to solve for a value of $c = [c_1 \ c_2 \ c_3 \ c_4 \ c_5]$. We will use MATLAB's non-linear fitting routines, which require a MATLAB function that computes the model. This function needs to be stored in its own m-file called model.m.

```
function y = model(c, x)
    y = c(1)*x.^2 + c(2)*x + c(3) + c(4)*sin(c(5)*x);
end
```

7.5 Performing the Non-Linear Fit

In order to have MATLAB do the fitting, we will use the lsqcurvefit command. This command does most of the work for us because it is intended for curve fitting in the least-squares sense.

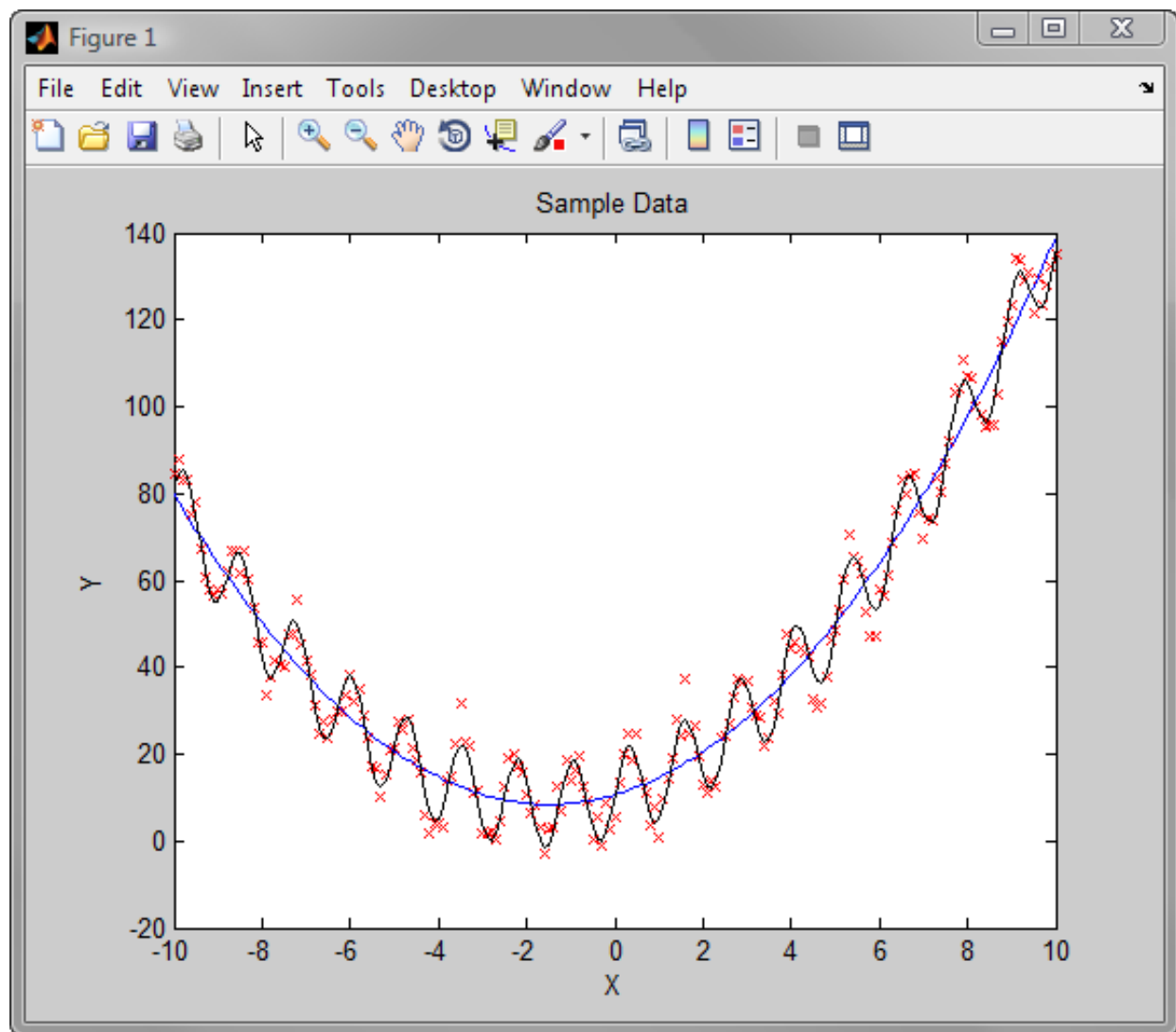
```
fit = lsqcurvefit(@model, [1 1 1 5 5], X, Y)
```

Note that we need to give an initial guess for the coefficients and use the @ sign to pass the name of our model function. The @ tells MATLAB to pass a function handle, rather than evaluate the function model. The output of the command is the optimal value for the coefficients. To evaluate the fit, we will plot it along with the sample data and the polynomial fit.

```
values = model(fit, X);
plot(X, values, 'color', 'black');
```

7.6 Exercise

1. Add the above commands to your main.m file so that running it produces the finished graph, with the data points and the two fits plotted.
2. Add a legend to your plot to explain which fit is which.



8 Three-Dimensional Surface Plot

8.1 Function of two variables

In this section we'll create a surface plot to visualize a function of two variables:

$$z = x \sin(y)$$

In order to set this up in MATLAB, we will need to create a matrix that is essentially a table of values for z over ranges of values for each of x and y .

8.2 Data grid

In order to set up these grids of values, we'll use `meshgrid`. We'll start with a very small range to illustrate the concepts.

```
[X, Y] = meshgrid(1:5);
```

This gives the following results for X and Y :

$$X = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{bmatrix} \quad Y = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 5 \end{bmatrix}$$

The idea is that by reading off corresponding entries in X and Y we can obtain all 25 points in the desired range.

8.3 Evaluate Function

Now we need to find the value of z at each of these points. By writing the function in element-wise notation, we can compute the value of z at each of the 25 points in a single MATLAB command.

```
Z = X .* sin(Y);
```

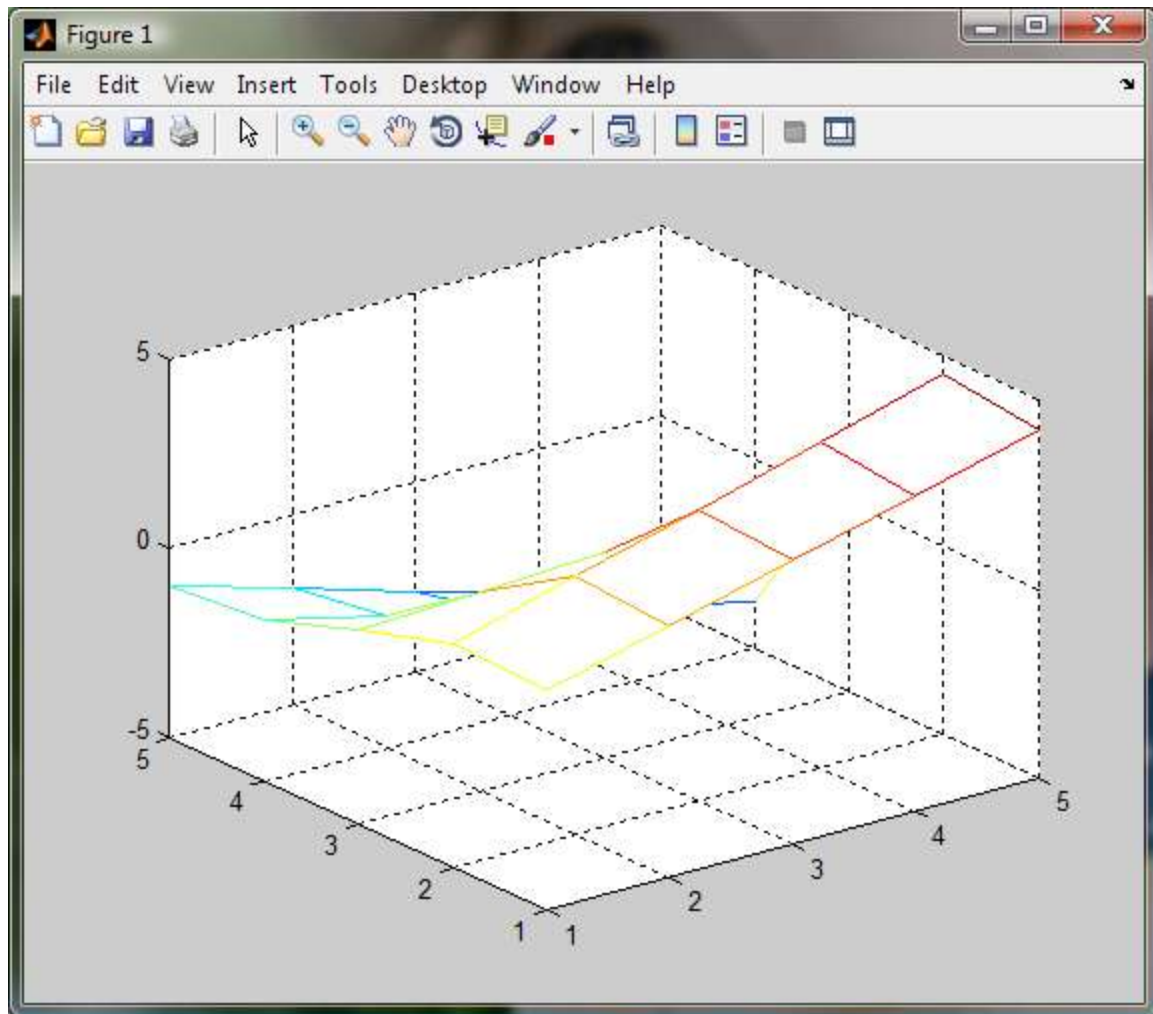
Note the use of the "dot" before the multiplication, and that the built-in `sin` function will operate element-wise on each entry in the matrix Y .

8.4 Plot with mesh

To draw the graph, we use the `mesh` command:

```
mesh(X, Y, Z);
```

This graph doesn't look like much since we only have 25 points.

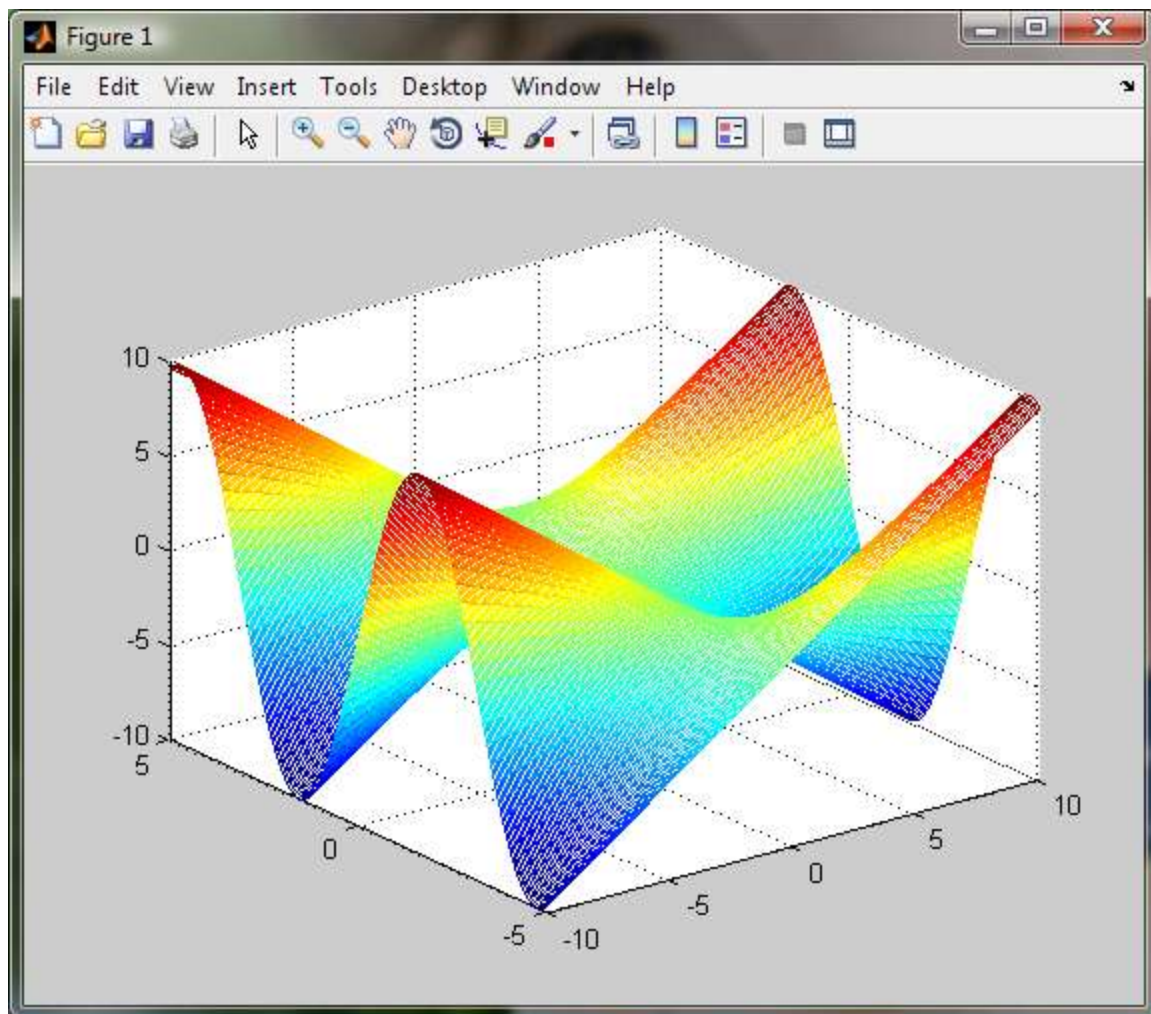


Now that we have the basic idea, we can expand the ranges of the variables and plot over more points.

8.5 A Better Plot

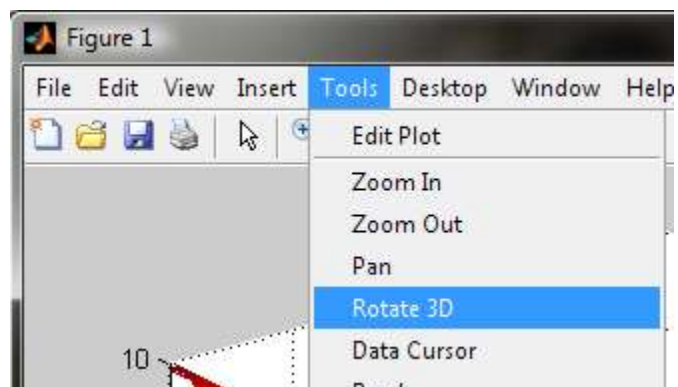
This code uses an expanded range, steps by 0.1 for more detail, and uses separate ranges for each of X and Y. This will create X and Y matrices that are rectangular instead of the square ones in the previous example.

```
rangeX = -10:.1:10;  
rangeY = -5:.1:5;  
[X, Y] = meshgrid(rangeX, rangeY);  
Z = X .* sin(Y);  
mesh(X, Y, Z);
```



8.6 Explore the Graph

In the Tools menu, choose Rotate 3D.

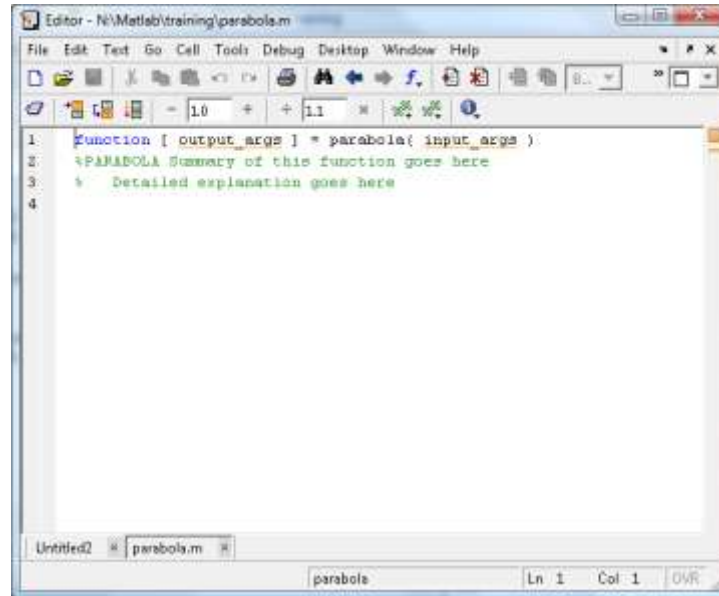


You can now click and drag on the graph to change the viewpoint.

9 Programming

9.1 MATLAB editor

To get into the MATLAB editor, you can double-click on an M-file in the Directory window, or choose File | Open and pick an m-file, or choose Window | Editor. Creating a new m-file from the Directory window conveniently creates a function prototype automatically and will save you a little bit of work. Right-click and choose New M-file. In this section we will work on an M-file called parabola.m. Here is what the editor looks like when you open a new m-file.



Notice that MATLAB automatically inserted a function definition. The MATLAB editor automatically applies colours to your code to make it more readable. Comments begin with a % and are green.

This function will take the parameters for a parabola and plot it. The parabola will be of the form:

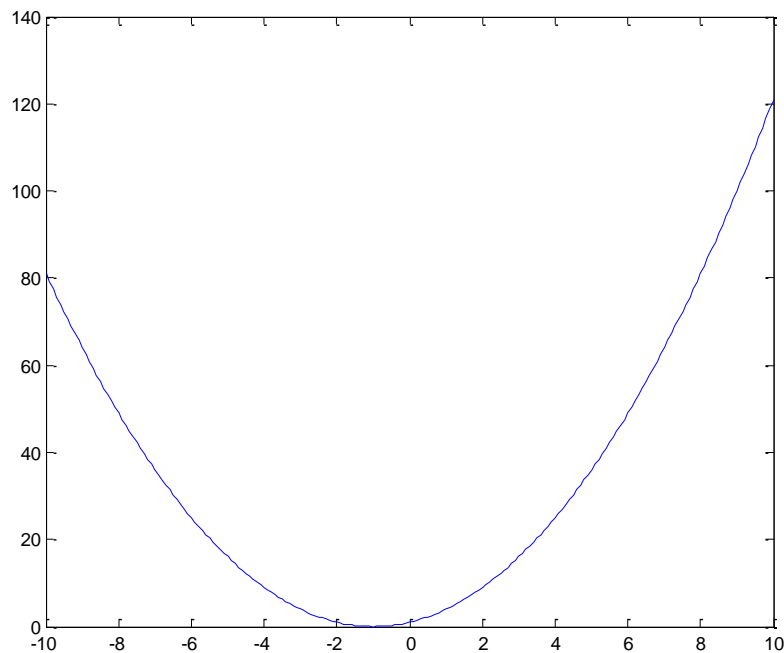
$$y = ax^2 + bx + c$$

9.2 Methodology (test commands, add to script)

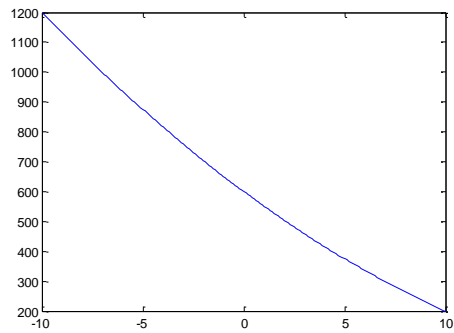
First, we delete the output_args and replace input_args with a,b,c the parameters of the parabola. Let's start with a simple plot.

```
Editor - N:\Matlab\training\parabola.m
File Edit Text Go Cell Tools Debug Desktop Window Help
[Icons] B...
- 1.0 + ÷ 1.1 x %>% %>%
1 function parabola( a, b, c)
2 %PARABOLA Summary of this function goes here
3 % Detailed explanation goes here
4
5 x = -10:1:10;
6 y = a*x.^2 + b*x + c;
7 plot(x, y);
8 end
Untitled2 x parabola.m x
parabola Ln 8 Col 4 OVR
```

To execute the code, type “parabola(1, 2, 1)” at a command window. A plot window will appear.



Unfortunately using a fixed range is not the best idea if we are going to handle a general parabola. There may be nothing of interest in the specified -10 to 10 domain. E.g. $y = x^2 - 50x + 600$ has roots $x=20$ and $x=30$ and a critical point at $x=25$. Running “parabola(1, -50, 600)” gives a graph:

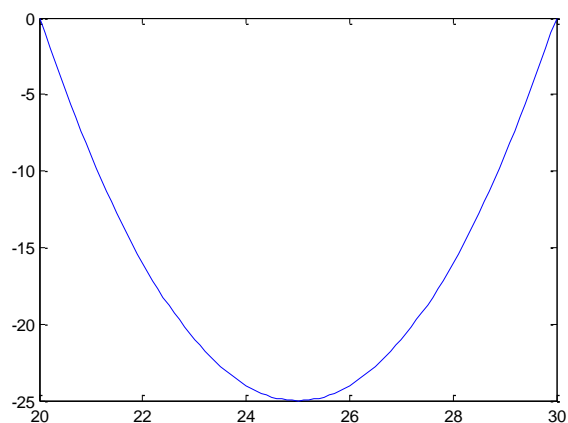


We need to find the roots of the polynomial and use these somehow in the plot range. Let's experiment at the command window:

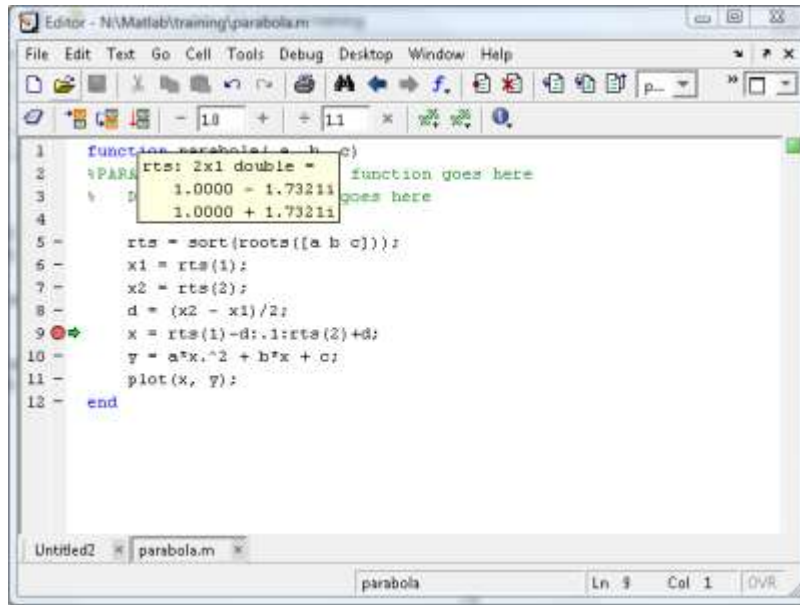
```
p = [1 -50 600]
roots(p)
sort(ans)
```

We need to sort the output of roots so that we have the smaller root first. Let's add the following code to the function:

```
rts = sort(roots([a b c]));
x1 = rts(1);
x2 = rts(2);
x = rts(1):.1:rts(2);
```



It would be even better to have the plot range extend on both sides of the roots, rather than just between the roots.



To see the value of variables, hover the mouse over them, or look in the workspace.

Current Directory		Workspace		
		Stack: parabola		
Name	Value	Min	Max	
a	1	1	1	
b	-2	-2	-2	
c	4	4	4	
d	0 + 1.7321i	0 + 1.7321i	0 + 1...	
rtss	[1 - 1.7321i; 1 + 1.7321i]	1 - 1.7321i	1 + 1...	
x1	1 - 1.7321i	1 - 1.7321i	1 - 1....	
x2	1 + 1.7321i	1 + 1.7321i	1 + 1...	

The problem is apparent: the parabola has complex roots!

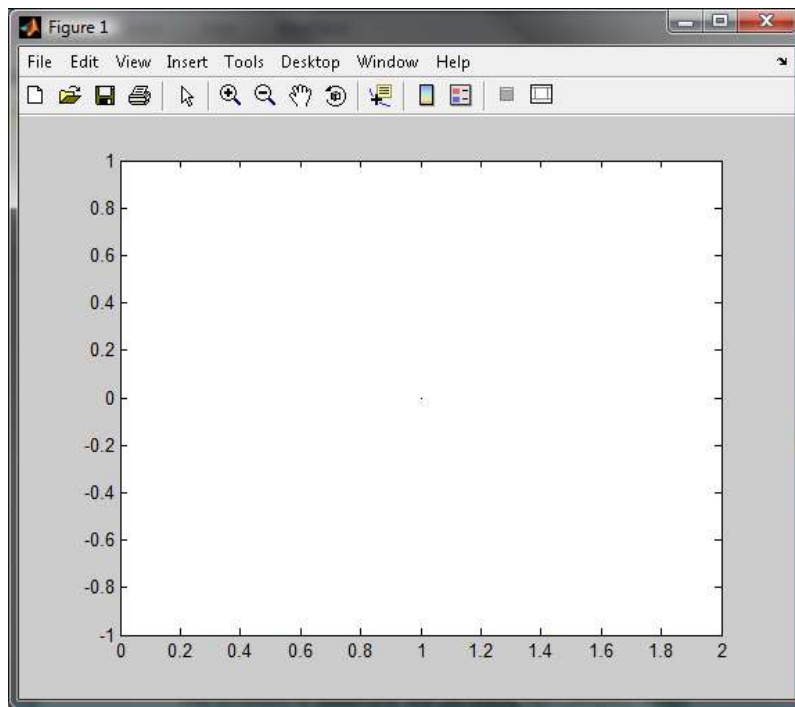
9.4 Exercises

9.4.1 Fix the function so it can plot a parabola with complex roots

Your function should choose a relevant range for the plot. Hint: There will always be a real-valued critical point. If $y = ax^2 + bx + c$, then $y' = 2ax + b$ so that $y' = 0$ when $x = -\frac{b}{2a}$.

9.4.2 Handle parabolas with a double root

Running `parabola(1, -2, 1)` gives an apparently empty plot (if you look really closely there is a dot at (1,0):



Investigate what is going wrong in the function and fix it.