

UNIVERSITY OF WATERLOO

Introduction to Scientific Computing with Matlab

SAW Training Course

R. William Lewis
Computing Consultant
Client Services – Information Systems & Technology

2007

Table of Contents

1	Matlab Basics	1
1.1	Obtaining software	1
1.2	Research License, Classroom license	1
1.3	Signing up for the Research License	1
1.4	Troubleshooting / Problems Running Matlab.....	1
1.5	Matlab Versions	1
1.6	Running Matlab.....	1
2	Desktop Environment	2
2.1	Command window (input, output)	2
2.1.1	Command Window	2
2.2	Command history.....	2
2.3	Current directory	2
2.4	Workspace	3
2.5	Editor.....	3
2.6	Matrices, Arrays	3
2.6.1	Dimensions, contents.....	3
2.6.2	Built-in matrices	3
2.6.3	Ranges (vectors).....	4
2.6.4	Manipulating Elements	4
2.6.5	Building up matrices.....	5
2.6.6	Matrix operations	6
2.6.7	Array operations	7
2.7	Functions.....	8
2.7.1	Function, operand, result.....	8
2.7.2	Dimensions of results.....	9
2.7.3	Example Statistical Summary Functions	9
2.8	Logical Indexing, Finding Elements	11
2.9	Exercises.....	12
2.9.1	Define the following matrices.....	12
2.9.2	Performing matrix and array operations	12

2.9.3	Restore your matrix definitions using Command History.....	12
2.9.4	Apply functions to the matrices you have defined.....	12
2.9.5	Investigate the “find” command.....	12
3	Getting Help.....	13
3.1	Help Menu.....	13
3.2	Help Window.....	13
3.3	Demos.....	14
3.4	Examples.....	14
4	Matlab editor.....	15
4.1	Basics.....	15
4.1.1	Starting editor.....	15
4.1.2	Source navigation.....	15
4.1.3	Saving.....	15
4.1.4	Splitting the editor Window.....	15
4.2	Running your script.....	15
4.3	Function in an m-file.....	15
4.4	Debugging Commands.....	16
4.5	Development Methodology.....	16
4.5.1	Experiment at command line, put working commands into script.....	16
4.5.2	Identify parameters that will change and make then function arguments.....	16
4.6	Exercises.....	16
4.6.1	Write a Matlab script to define test data.....	16
4.6.2	Write a Matlab function that adds the sine and cosine of each entry in a matrix.....	16
5	Plotting.....	17
5.1	Demonstration.....	17
5.1.1	Aside: Cell Mode M-Files.....	17
5.2	Creating Plots Interactively.....	17
5.3	Formatting plots interactively.....	19
5.4	Create script file.....	20
6	Importing Data for Analysis.....	21
6.1	Import from Excel.....	21
6.2	Demonstration M file.....	22

6.3	Organize/arrange data	22
6.4	Exercises.....	24
6.4.1	Problems with Tail data	24
6.4.2	Investigate Correlations	24
7	Programming	25
7.1	Matlab editor	25
7.2	Methodology (test commands, add to script)	25
7.3	Testing, debugging, spying as it's running	28
7.4	Exercises.....	29
7.4.1	Fix the function so it can plot a parabola with complex roots.....	29
7.4.2	Handle parabolas with a double root	29

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. 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 Signing up for the Research License

To buy an annual license for Matlab, visit <http://ist.uwaterloo.ca/ew/saw/webstore/>. UW accounts are accepted for payment. The account holder will need to login with their UW user ID and password.

1.4 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 as 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.

1.5 Matlab Versions

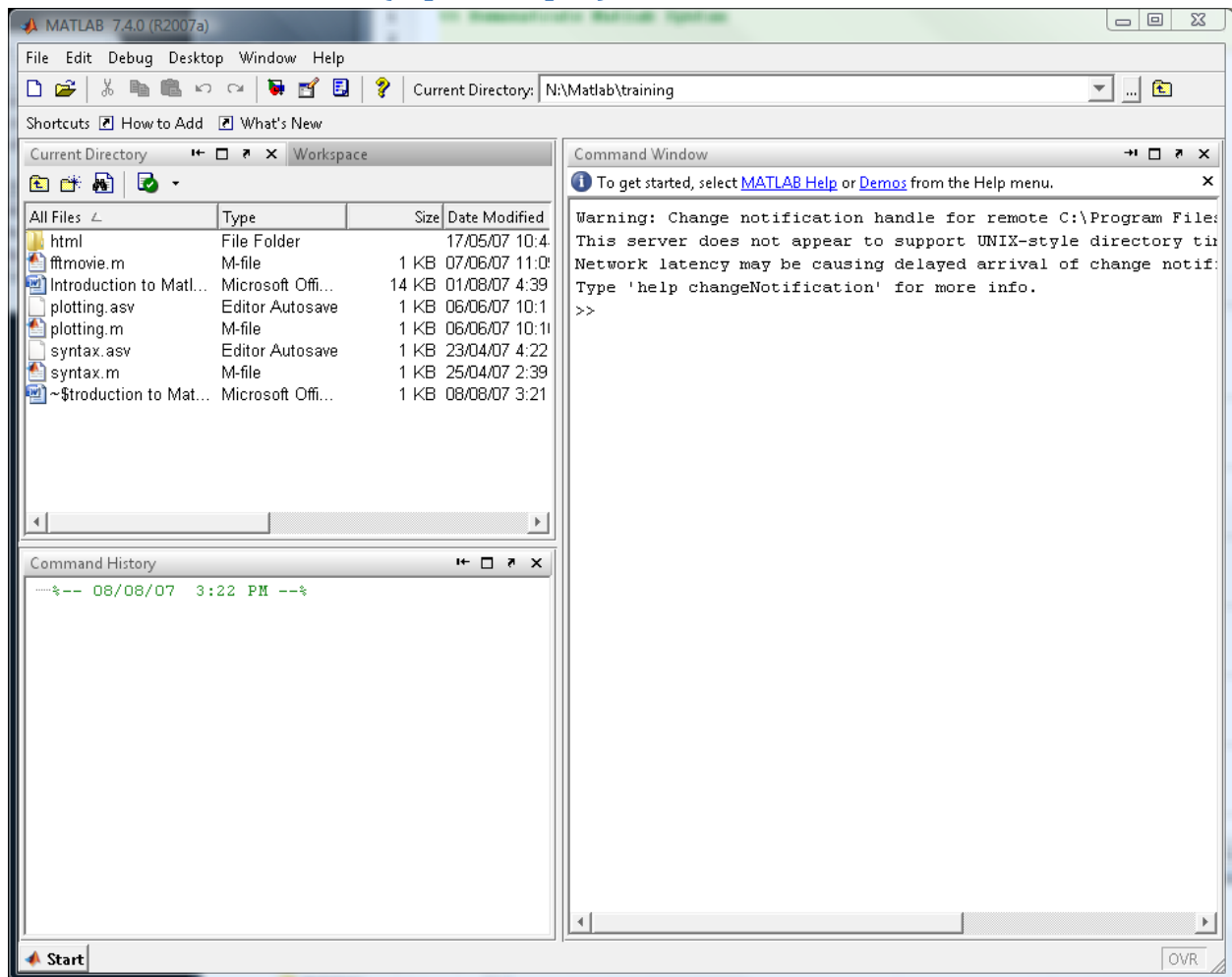
There are two Matlab releases each year. This course is based on R2007a and a newer version R2007b was released September 1, 2007. Matlab and each of its toolboxes have individual version numbers as well. Release R2007a corresponds to Matlab 7.4 and version 17.

1.6 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. In the training lab you can choose to run under either the research or classroom license.

2 Desktop Environment

2.1 Command window (input, output)



2.1.1 Command Window

Your first interaction with Matlab will be through the Command Window, which shows up as the right hand half of the Matlab window. This where you can type Matlab commands and view the output of these commands.

2.2 Command history

In the lower left 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 some other options.

2.3 Current directory

The top left corner of the Matlab window shows the files in the current directory. 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.

2.4 Workspace

The Current Directory tab shares space with the Workspace tab. The Workspace tab 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.

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]

A =

     1     2     3
     4     5     6
     7     8     9
```

2.6.2 Built-in matrices

To get started building arrays and matrices quickly, you can use these built-in commands. I’ve shown 3x4 matrices for the sake of example and to keep the notation clear:

Command	Array
<code>zeros(3,4)</code>	3x4 array full of zeros
<code>ones(3,4)</code>	3x4 array full of ones
<code>eye(3,4)</code>	3x4 matrix with ones on the main diagonal (the identity in matrix algebra)
<code>rand(3,4)</code>	3x4 array of random numbers from 0 to 1
<code>diag([3 4 2])</code>	3x3 matrix with specified entries on the main diagonal
<code>magic(3)</code>	3x3 magic square
<code>pascal(3)</code>	3x3 matrix with entries from Pascal’s triangle

2.6.3 Ranges (vectors)

To indicate a range in Matlab, you use the colon. E.g.

```
>> 1:4

ans =

     1     2     3     4
```

To specify the increment, use the syntax start:increment:end.

```
>> 1:3:20

ans =

     1     4     7    10    13    16    19
```

The increment, start and end can all be decimal numbers:

```
>> .5:.2:1.4

ans =

 0.5000  0.7000  0.9000  1.1000  1.3000
```

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

2.6.4 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.

```
>> A = magic(5)

A =

    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9

>> A(1:3,2:4)

ans =

    24     1     8
     5     7    14
     6    13    20
```


To specify the last element in a row or column use “end”:

```
>> A(3:end,4)

ans =

    20
    21
     2
```

Matlab also allows linear indexing by starting with the first column then moving through each column in turn:

```
>> A = magic(5)

A =

    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9

>> A(2)

ans =

    23

>> A(20)

ans =

     2
```

2.6.5 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]

A =

     1     2     3
     4     5     6
     7     8     9
```

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 =
```

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

```
>> A = [magic(3); ones(3)]
```

```
A =
```

```
8 1 6
3 5 7
4 9 2
1 1 1
1 1 1
1 1 1
```

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

```
A =
```

```
0 0 0 1 1 1
0 0 0 1 1 1
0 0 0 1 1 1
8 1 6 1 2 3
3 5 7 2 3 4
4 9 2 3 4 5
```

2.6.6 Matrix operations

It is now 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 = magic(3)
```

```
A =
```

```
     8     1     6
     3     5     7
     4     9     2
```

```
>> B = eye(3)
```

```
B =
```

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

```
>> A * B
```

```
ans =
```

```
     8     1     6
     3     5     7
     4     9     2
```

```
>> B / A
```

```
ans =
```

```
    0.1472   -0.1444    0.0639
   -0.0611    0.0222    0.1056
   -0.0194    0.1889   -0.1028
```

2.6.7 Array operations

To make a Matlab operation work element-wise, or in a array manner, you use the standard mathematical symbol with a period before it. Note however that addition is the same for matrices and arrays.

```
>> A .* B
```

```
ans =
```

```
     8     0     0
     0     5     0
     0     0     2
```

```

>> B ./ A

ans =

    0.1250         0         0
         0    0.2000         0
         0         0    0.5000

```

When dimensions don't match, Matlab tries to do the right thing. You can multiply an array by a single number (scalar) to multiply each element by that number or add a scalar to an array to add the number to each element in the array.

```

>> A = magic(3)

A =

     8     1     6
     3     5     7
     4     9     2

>> A + 1

ans =

     9     2     7
     4     6     8
     5    10     3

>> 2*A

ans =

    16     2    12
     6    10    14
     8    18     4

```

2.7 Functions

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

2.7.1 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.

2.7.2 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.

2.7.3 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.

```
>> A = magic(5)

A =

    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9

>> A(1), A(6), A(11), A(16)

ans =

    17

ans =

    24

ans =

     1

ans =

     8
```

If you use the colon as a linear index, Matlab converts the matrix to a vector:

```
>> A(:)
```

```
ans =
```

```
17
23
4
10
11
24
5
6
12
18
1
7
13
19
25
8
14
20
21
2
15
16
22
3
9
```

Combined with a statistical function gives a compact way to operate on the entire matrix rather than on each column:

```
>> sum(A)
```

```
ans =
```

```
65 65 65 65 65
```

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

```
ans =
```

```
325
```

2.8 Logical Indexing, Finding Elements

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

```
>> A = magic(4)

A =

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1

>> A(A>10)

ans =

    16
    11
    14
    15
    13
    12
```

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, using linear indexing:

```
>> find(A>10)
```

```
ans =
```

```
1  
6  
8  
12  
13  
15
```

2.9 Exercises

2.9.1 Define the following matrices

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

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

$$C = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \\ 3 & 0 & 0 & 5 & 4 & 4 \\ 0 & 2 & 0 & 4 & 5 & 4 \\ 0 & 0 & 1 & 4 & 4 & 5 \end{bmatrix}$$

Note that C is made up of four matrices with special structures!

2.9.2 Performing matrix and array operations

Compute matrix operations on A, B, and C. Add the columns, average all elements.

2.9.3 Restore your matrix definitions using Command History

2.9.4 Apply functions to the matrices you have defined

2.9.5 Investigate the “find” command

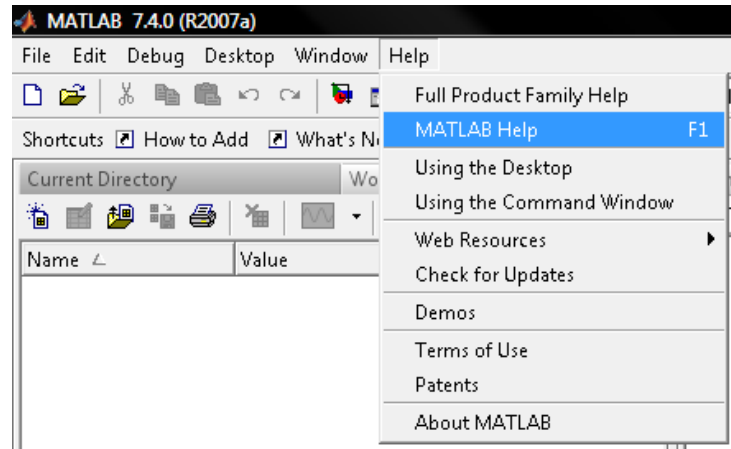
The find command can return row and column indices instead of linear indices by assigning its result to a matrix of two variables:

```
[i, j] = find(A>10);
```


3 Getting Help

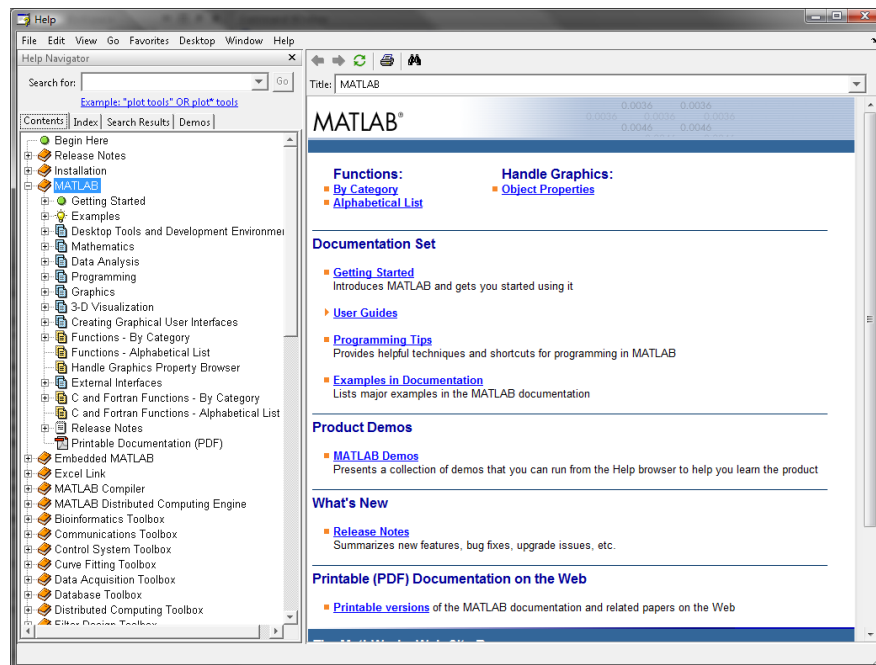
3.1 Help Menu

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



3.2 Help Window

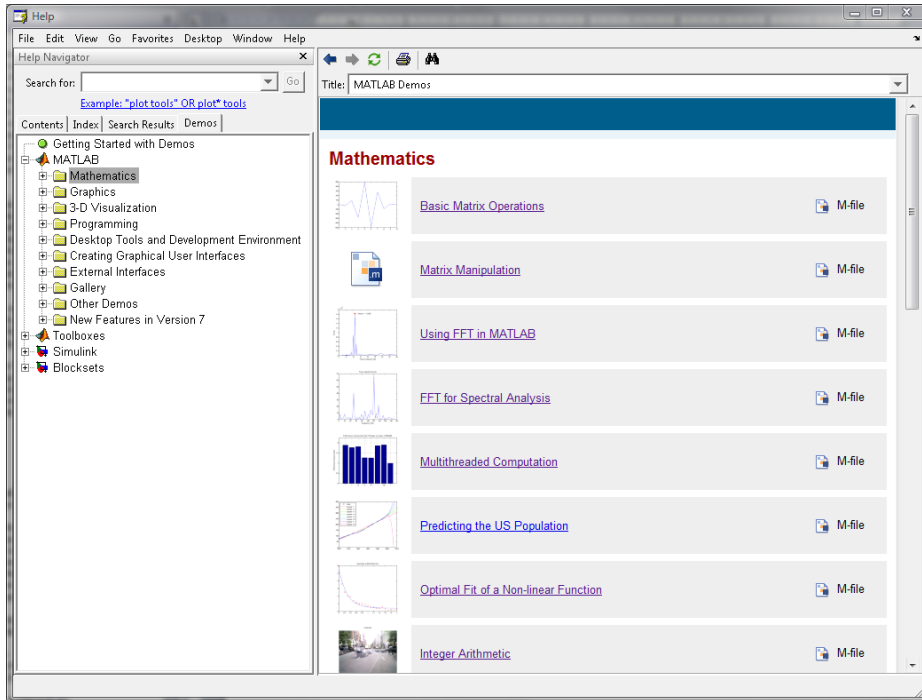
Matlab Help takes you to the help window.



Browsing through the topics is a good way to learn about Matlab and to learn about new functions and toolboxes. The search is good for getting documentation on a particular function (if you know its name already).

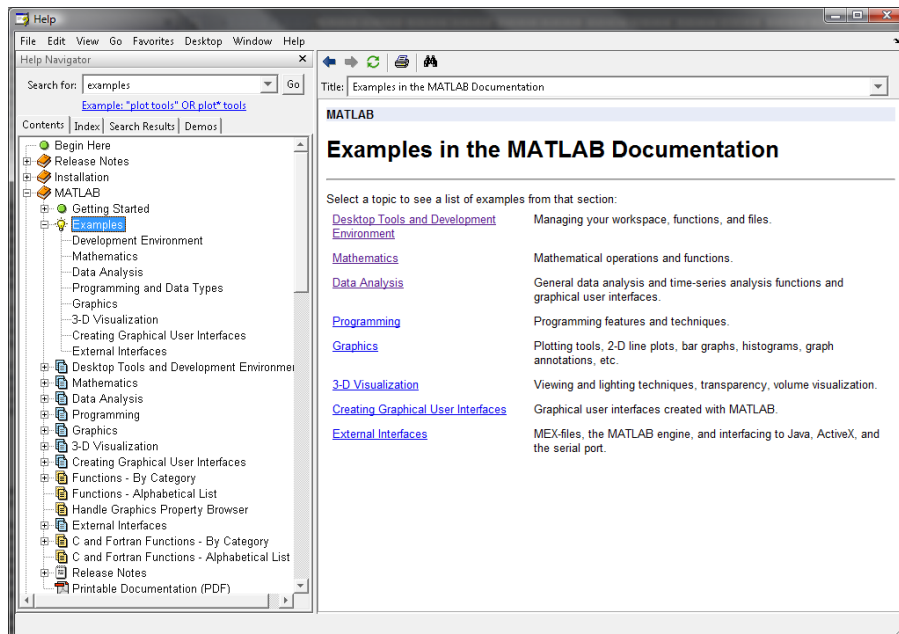
3.3 Demos

One of the best ways to learn more about Matlab is to use the included Demos. Click on the demos tab in the help browser or use Help | Demos. You can browse through them by category. A search of the help system will also give you results for demos.



3.4 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 so easier to debug.

To define a function called myfunction which takes input a, b and c and returns output A, write “function A = myfunction(a, b, c)”. The next lines define the function which should assign the value to be returned into A. Terminate your function with the “end” command. 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.

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.

4.6 Exercises

4.6.1 Write a Matlab script to define test data

$$testdata = \begin{pmatrix} 1 & .5 \\ 1.2 & .8 \\ 1.4 & 1.1 \\ 1.8 & 1.5 \\ 2.3 & 2 \\ 3 & 4 \end{pmatrix}$$

Now you can manipulate the “testdata” matrix and restore it at any time by running your script. Try manipulating the data at the command line, then add your commands to the script.

4.6.2 Write a Matlab function that adds the sine and cosine of each entry in a matrix

5 Plotting

5.1 Demonstration

A demonstration of plotting is in the file “plotting.m”. During the Matlab course, this file is available in R:\Matlab\plotting.m. After the course, these files are also available on the web at:

<http://ist.uwaterloo.ca/ew/saw/matlab/course>

There is a corresponding .html file which shows both the Matlab commands and their results.

5.1.1 Aside: Cell Mode M-Files

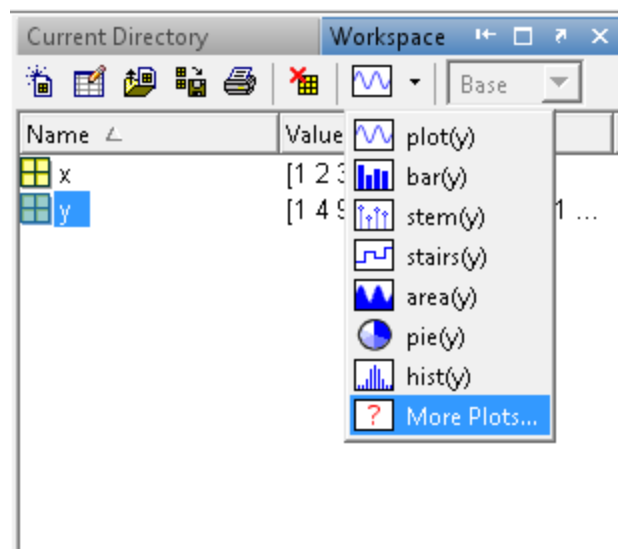
These M-files use cell mode in the editor, which gives a convenient way of publishing to html. It also allows for a different way of running your Matlab code. You can select a cell in the editor, press CTRL+Enter and the code is executed immediately in the workspace. This training does not cover cell mode but you will see it used in these demonstration files. There is a lot of information about cell mode in the help system, and it is used extensively in the matlab help system and demos.

5.2 Creating Plots Interactively

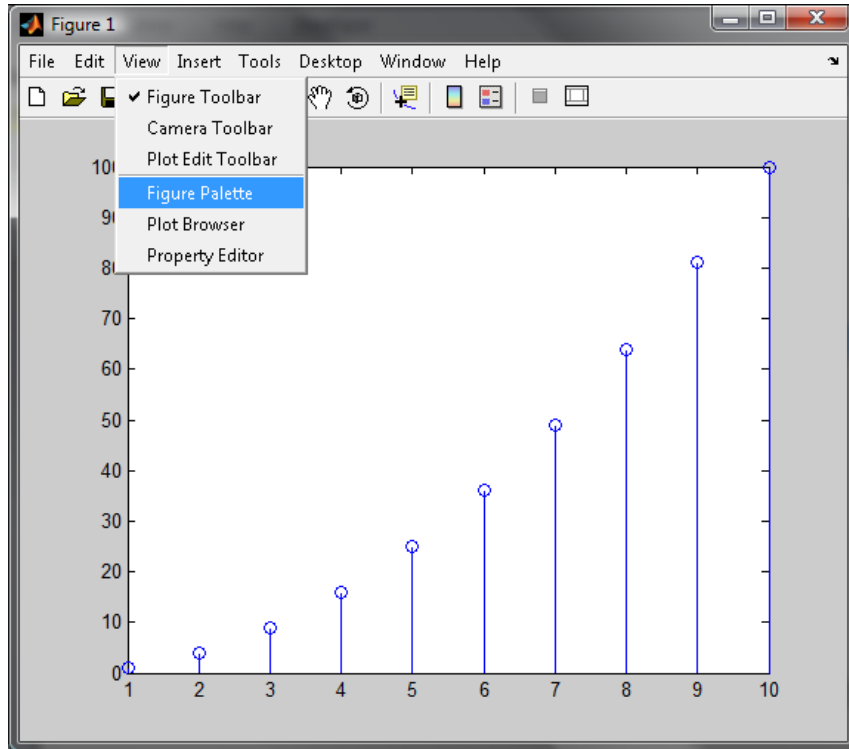
Once you have data in your workspace, you can use the Matlab GUI (graphical user interface) to create plots. The following commands will create some very simple data:

```
x = 1:10;  
y = x.^2;
```

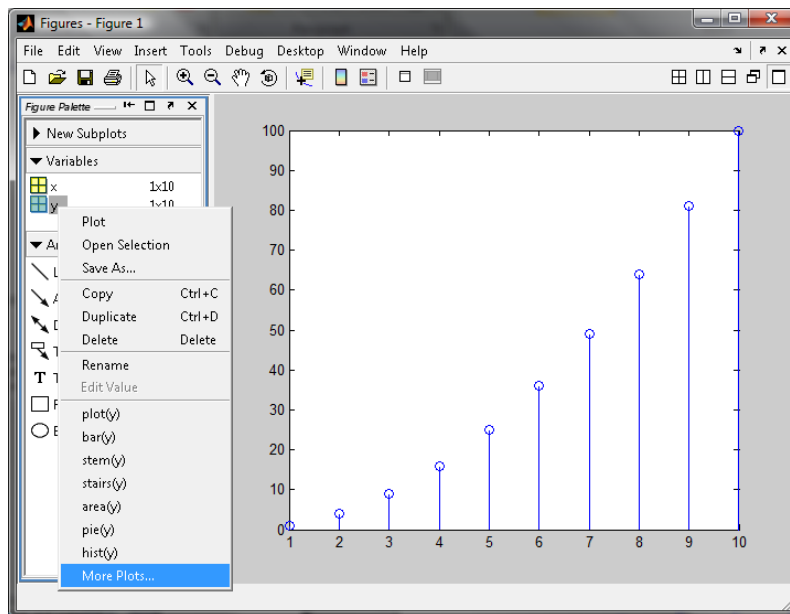
Select the y variable and click the graph icon to access a menu of plots. These notes will continue with a stem plot. (You can also right-click on a variable to access plots.)

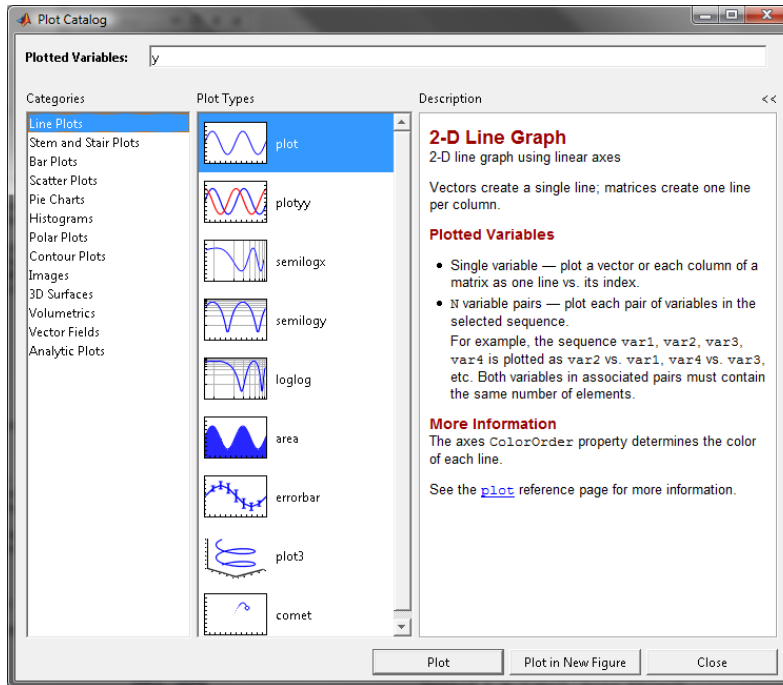


Matlab opens a figure window with the requested plot. You can also access the list of available plots from the Figure window by opening the Figure Browser and right clicking on a variable name.



Choosing "More Plots" (here or in the Workspace) will open the Plot Catalog.





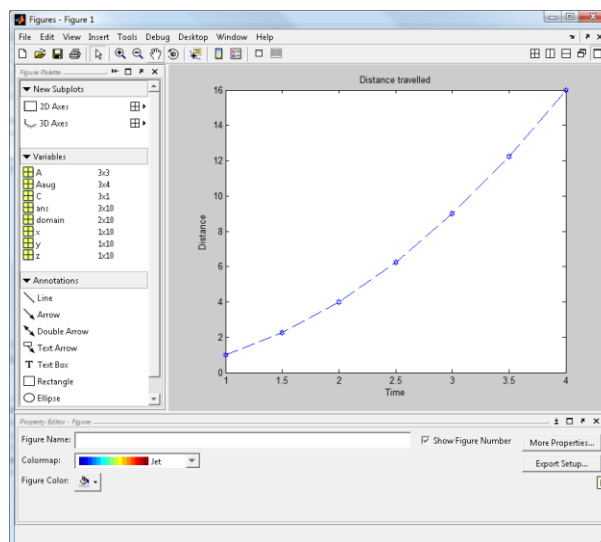
The plot catalog organizes all the different types of Matlab plots and explains how to use them.

5.3 Formatting plots interactively

In an active Figure window, there are menu and toolbar options available. You can see additional options by choosing “View | Figure Palette” (if it’s not still open) and “View | Property Editor”.

- Insert | X-label, type “Time”
- Insert | Y-label, type “Distance”
- Insert | Title, type “Distance travelled”

Click on data series, change line to dotted, add a marker. Here’s what this looks like, starting from the basic 2-D line plot.



5.4 Create script file

Choose File | Generate M-file. Matlab generates the code needed to reproduce your formatting.

```
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 21-Aug-2007 11:12:59
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,'Marker','hexagram','LineStyle','--');
18
19 % Create xlabel
20 xlabel({'Time'});
21
22 % Create ylabel
23 ylabel({'Distance'});
24
25 % Create title
26 title({'Distance travelled'});
27
28
```

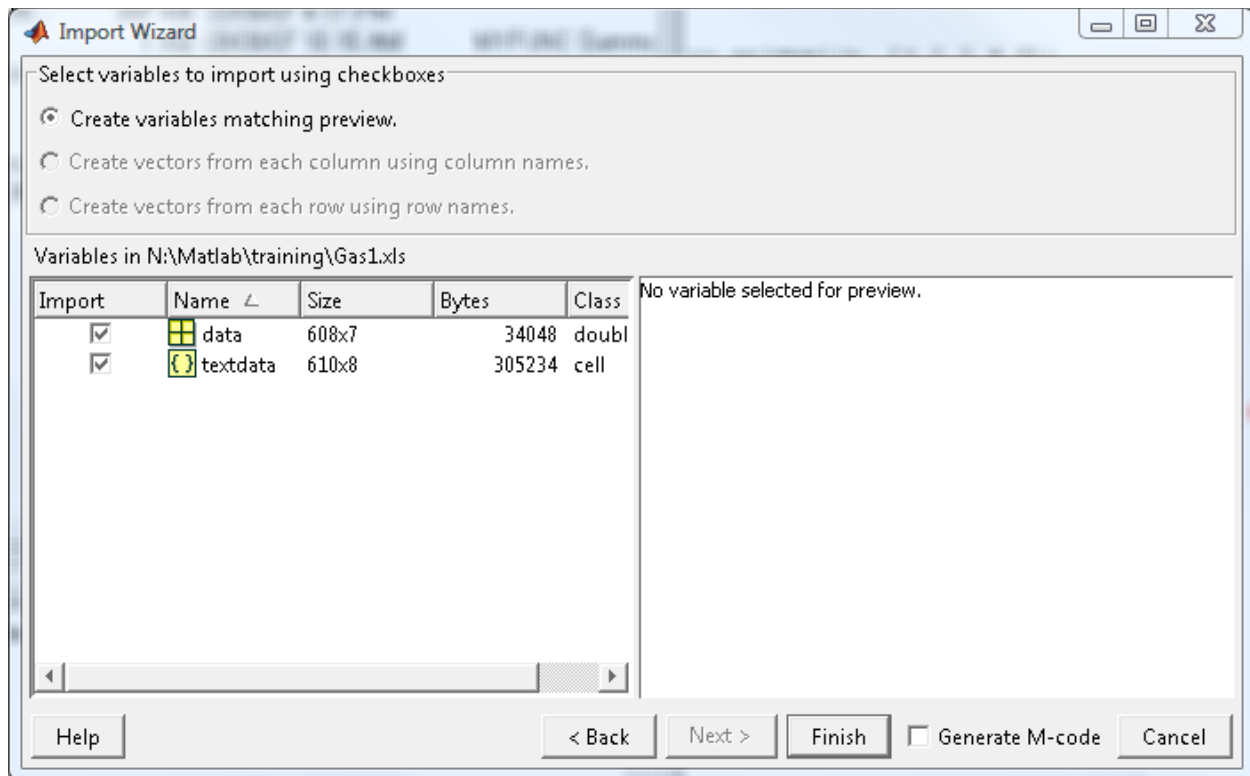

6 Importing Data for Analysis

6.1 Import from Excel

The file “gas1.xls” contains price information for gasoline (source <http://www.eia.doe.gov/emeu/international/prices.html>) . We will import this data into Matlab for analysis.

	A	B	C	D	E	F	G	H
1	Date	Belgium	France	Germany	Italy	Netherlands	UK	US
2	01/01/1996	3.95	3.93	4.07	3.89	4.32	3.20	1.27
3	08/01/1996	3.93	3.92	4.03	3.94	4.32	3.24	1.28
4	15/01/1996	3.92	3.90	4.00	3.95	4.29	3.25	1.29
5	22/01/1996	3.82	3.83	3.95	3.90	4.20	3.05	1.28
6	29/01/1996	3.80	3.77	3.92	3.87	4.18	3.00	1.27
7	05/02/1996	3.84	3.82	3.98	3.95	4.23	3.06	1.27
8	12/02/1996	3.84	3.80	3.97	3.94	4.21	3.03	1.27
9	19/02/1996	3.94	3.89	3.97	3.94	4.27	3.03	1.27
10	26/02/1996	4.01	3.97	4.07	3.94	4.38	3.07	1.29
11	04/03/1996	3.95	3.93	3.92	4.03	4.32	3.02	1.31
12	11/03/1996	3.92	3.92	3.96	4.02	4.30	3.02	1.31
13	18/03/1996	3.99	3.96	4.03	4.03	4.35	3.02	1.32
14	25/03/1996	3.98	3.97	4.00	4.06	4.41	3.00	1.35
15	01/04/1996	3.98	3.97	4.00	4.06	4.41	3.00	1.36
16	08/04/1996	3.98	3.97	4.00	4.06	4.41	3.00	1.38
17	15/04/1996	4.06	3.99	4.03	4.09	4.43	3.03	1.42
18	22/04/1996	4.04	4.00	3.99	4.11	4.41	3.04	1.44

To get started, simply open the data file from within Matlab. This will open the Import Wizard.



These defaults will work for this example. Click Finish.

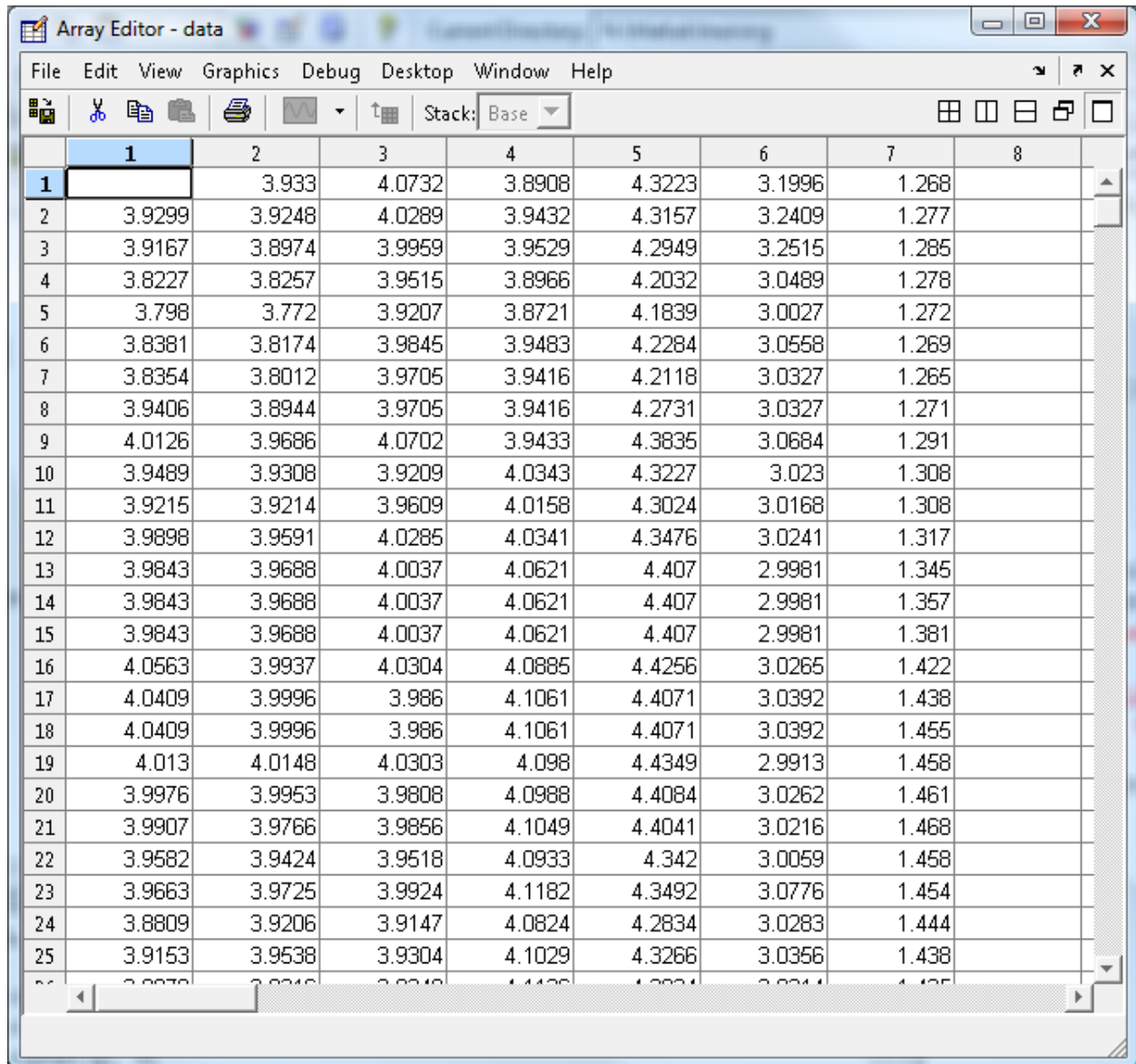
6.2 Demonstration M file

This example is also available in your course folder in R:\Matlab\gasprices.m. The M-file approach uses the command “xlsread” to bring in Excel data. The remaining commands are those demonstrated below.

6.3 Organize/arrange data

The wizard creates two variables: data and textdata. Double-click on these in the Workspace to view their contents.

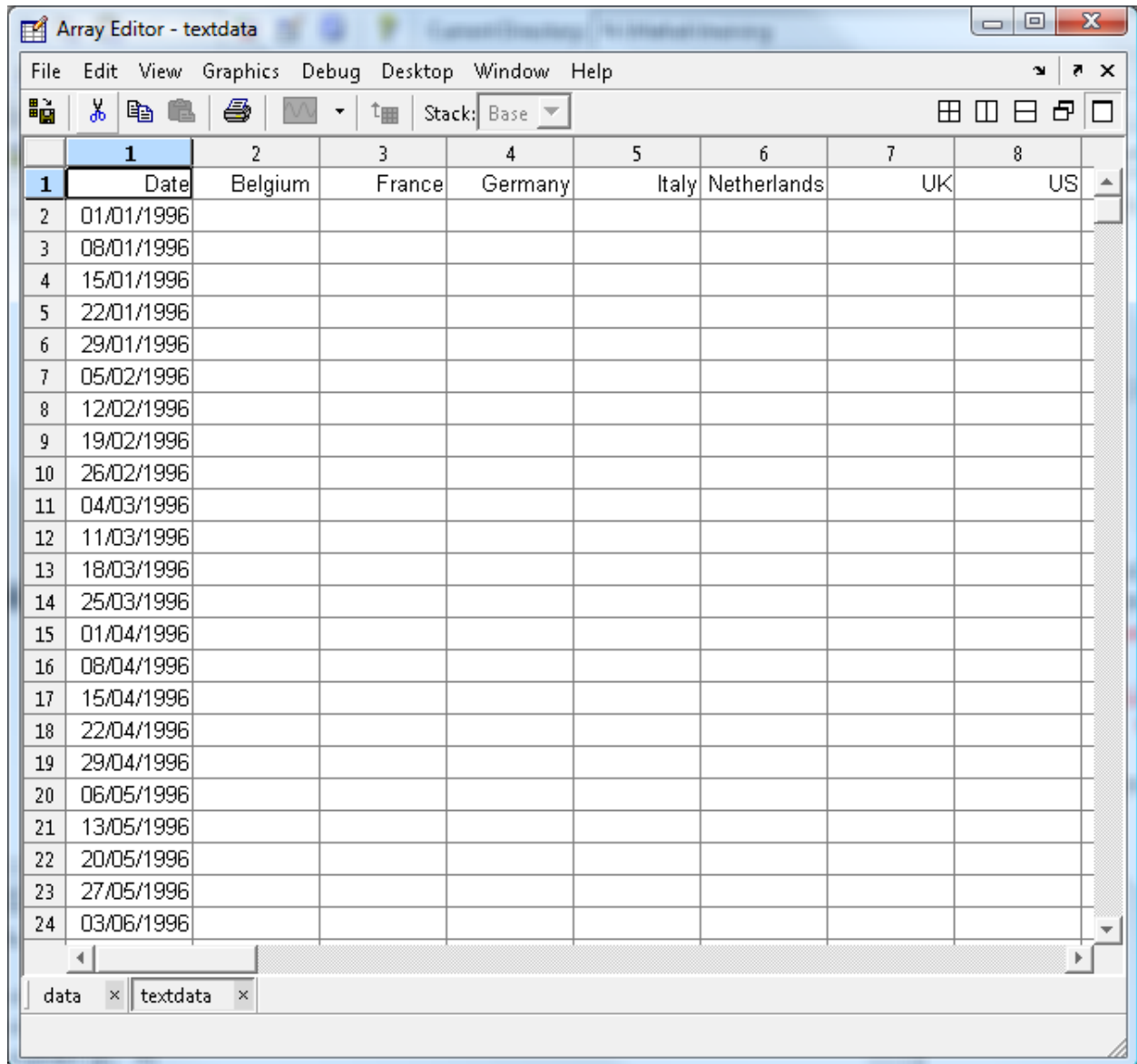
The variable “data” is a 608x7 matrix of floating point numbers (Matlab type double).



The screenshot shows the MATLAB Array Editor window for a variable named 'data'. The window title is 'Array Editor - data'. The menu bar includes File, Edit, View, Graphics, Debug, Desktop, Window, and Help. The toolbar contains icons for copy, paste, delete, and other editing functions. The main area displays a grid of data with 25 rows and 8 columns. The first column is labeled '1' and the other columns are labeled '2' through '8'. The data values are floating point numbers ranging from approximately 3.7 to 4.4. The window also shows a 'Stack' dropdown menu set to 'Base' and various window control buttons.

	1	2	3	4	5	6	7	8
1		3.933	4.0732	3.8908	4.3223	3.1996	1.268	
2	3.9299	3.9248	4.0289	3.9432	4.3157	3.2409	1.277	
3	3.9167	3.8974	3.9959	3.9529	4.2949	3.2515	1.285	
4	3.8227	3.8257	3.9515	3.8966	4.2032	3.0489	1.278	
5	3.798	3.772	3.9207	3.8721	4.1839	3.0027	1.272	
6	3.8381	3.8174	3.9845	3.9483	4.2284	3.0558	1.269	
7	3.8354	3.8012	3.9705	3.9416	4.2118	3.0327	1.265	
8	3.9406	3.8944	3.9705	3.9416	4.2731	3.0327	1.271	
9	4.0126	3.9686	4.0702	3.9433	4.3835	3.0684	1.291	
10	3.9489	3.9308	3.9209	4.0343	4.3227	3.023	1.308	
11	3.9215	3.9214	3.9609	4.0158	4.3024	3.0168	1.308	
12	3.9898	3.9591	4.0285	4.0341	4.3476	3.0241	1.317	
13	3.9843	3.9688	4.0037	4.0621	4.407	2.9981	1.345	
14	3.9843	3.9688	4.0037	4.0621	4.407	2.9981	1.357	
15	3.9843	3.9688	4.0037	4.0621	4.407	2.9981	1.381	
16	4.0563	3.9937	4.0304	4.0885	4.4256	3.0265	1.422	
17	4.0409	3.9996	3.986	4.1061	4.4071	3.0392	1.438	
18	4.0409	3.9996	3.986	4.1061	4.4071	3.0392	1.455	
19	4.013	4.0148	4.0303	4.098	4.4349	2.9913	1.458	
20	3.9976	3.9953	3.9808	4.0988	4.4084	3.0262	1.461	
21	3.9907	3.9766	3.9856	4.1049	4.4041	3.0216	1.468	
22	3.9582	3.9424	3.9518	4.0933	4.342	3.0059	1.458	
23	3.9663	3.9725	3.9924	4.1182	4.3492	3.0776	1.454	
24	3.8809	3.9206	3.9147	4.0824	4.2834	3.0283	1.444	
25	3.9153	3.9538	3.9304	4.1029	4.3266	3.0356	1.438	

The matrix “textdata” is a 610x8 cell array. It contains the values which could not be converted to numbers.



	1	2	3	4	5	6	7	8
1	Date	Belgium	France	Germany	Italy	Netherlands	UK	US
2	01/01/1996							
3	08/01/1996							
4	15/01/1996							
5	22/01/1996							
6	29/01/1996							
7	05/02/1996							
8	12/02/1996							
9	19/02/1996							
10	26/02/1996							
11	04/03/1996							
12	11/03/1996							
13	18/03/1996							
14	25/03/1996							
15	01/04/1996							
16	08/04/1996							
17	15/04/1996							
18	22/04/1996							
19	29/04/1996							
20	06/05/1996							
21	13/05/1996							
22	20/05/1996							
23	27/05/1996							
24	03/06/1996							

We need to get the dates and names of the columns out of textdata so that they are available in plots. Here are the commands to extract these portions:

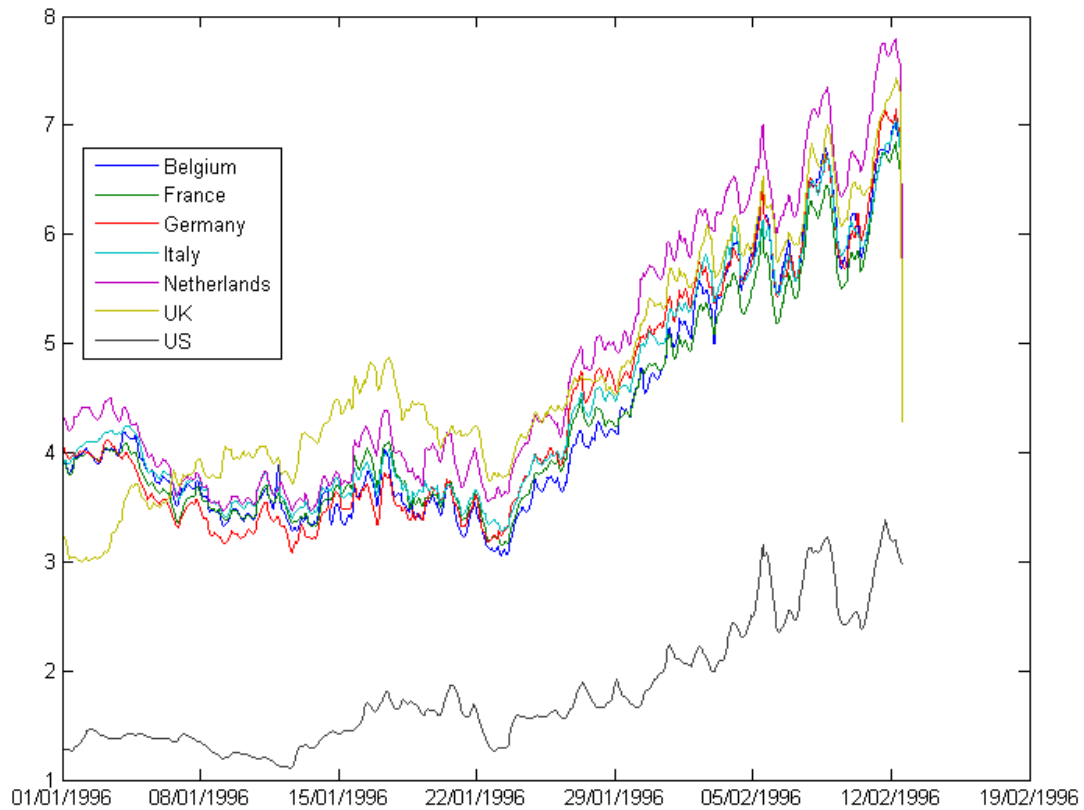
```
dates = textdata(2:end, 1)
cats = textdata(1, 2:end)
```

There are missing data, which Matlab shows as “NaN” (not a number). These values will make analysis and plotting difficult. In this example, we will smooth the data using a moving average filter. This fills in the missing data and also makes trends easier to see in the plots.

```
data = smooth(data)
```

To plot the data:

```
plot(data)
set(gca, 'XTickLabel', 'dates');
legend(cats);
```



To analyze the data, we can compute the pair-wise correlation coefficients:

```
corr(data)
```

6.4 Exercises

6.4.1 Problems with Tail data

The graph shows a steep drop off in some of the data series. Investigate why this is happening and fix it. (Hint: it is related to the way we “smoothed” the data.)

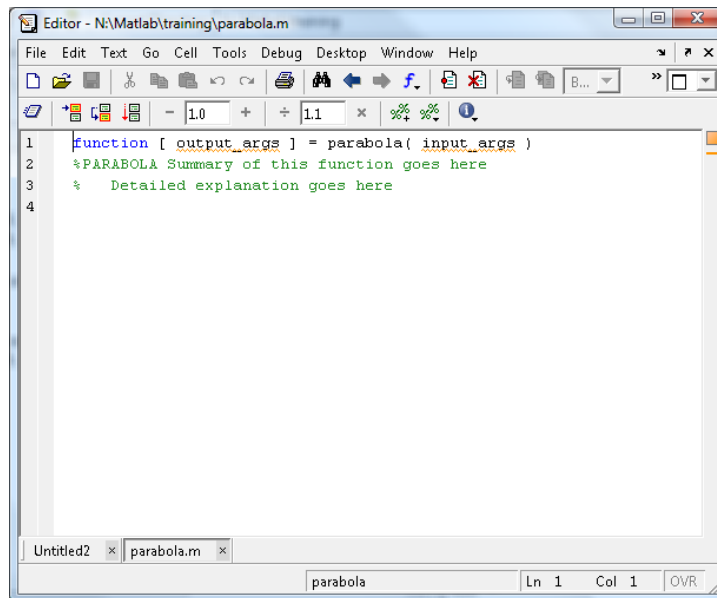
6.4.2 Investigate Correlations

Some of the data are highly correlated and some of the correlations are not so strong. Try plotting the highly correlated data together. Conversely, compare some of the series that do not have such a strong correlation.

7 Programming

7.1 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:

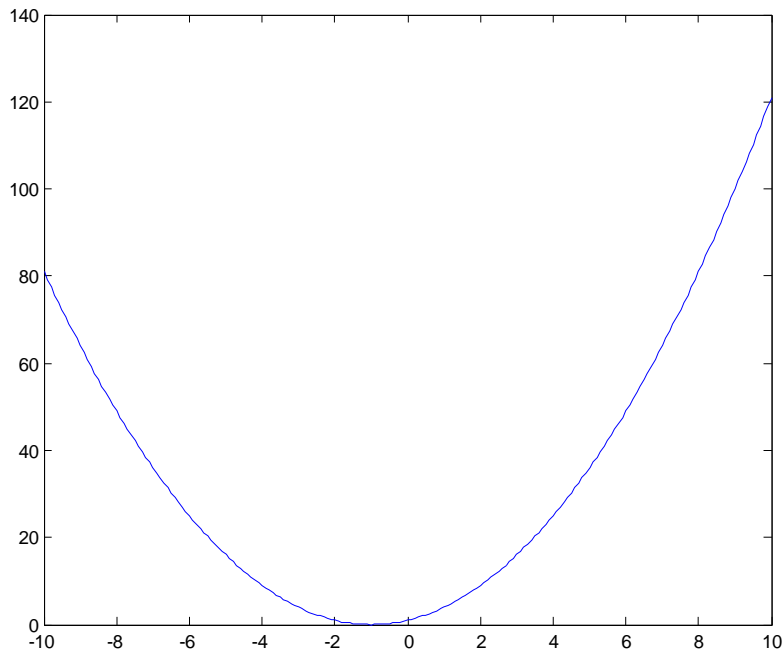
$$y = ax^2 + bx + c$$

7.2 Methodology (test commands, add to script)

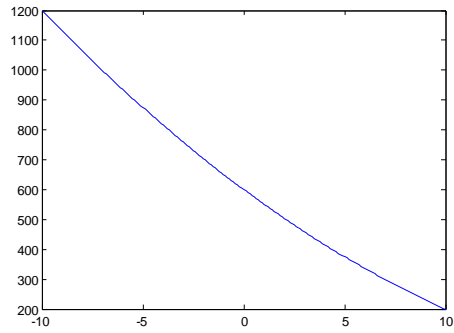
First, we delete the output_args and replace input_args with a,b,c the parameters of the parabola. Let's start with a simple plot.

```
Editor - N:\Matlab\training\parabola.m
File Edit Text Go Cell Tools Debug Desktop Window Help
[Icons]
- 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
[Icons]
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:

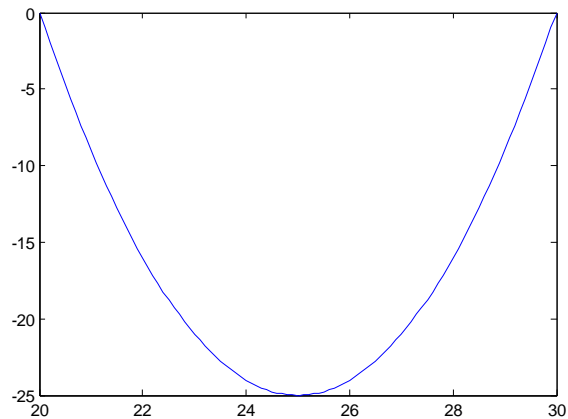


We need to find the roots of the polynomial and use these somehow in the plot range. Let's experiment at the command window:

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

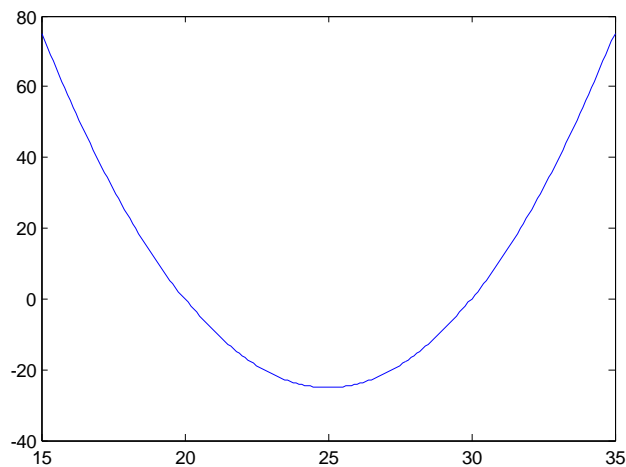
We need to sort the output of roots so that we have the smaller root first. Let's add the following code to the function:

```
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.

```
Editor - N:\Matlab\training\parabola.m
File Edit Text Go Cell Tools Debug Desktop Window Help
+ - 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    rts = sort(roots([a b c]));
6     x1 = rts(1);
7     x2 = rts(2);
8     d = (x2 - x1)/2;
9     x = rts(1)-d:.1:rts(2)+d;
10    y = a*x.^2 + b*x + c;
11    plot(x, y);
12 end
Untitled2 x parabola.m x
parabola Ln 9 Col 29 OVR
```



7.3 Testing, debugging, spying as it's running

Seems easy enough, right? But we never considered what parameters are valid for our function and what happens if Matlab can't find the roots!

```
>> parabola(1, -2, 4)
Warning: Colon operands must be real scalars.
> In parabola at 9
```

What happened? Let's spy on parabola as it runs. In the editor, go to line 9 where the error is occurring, and choose Debug | Set/Clear Breakpoint or press F12. Now run parabola. The editor opens with the cursor at line 9.


```

1 function parabola(a,b,c)
2 %PARA rts: 2x1 double = function goes here
3 % D 1.0000 - 1.7321i goes here
4 1.0000 + 1.7321i
5
6 rts = sort(roots([a b c]));
7
8 x1 = rts(1);
9 x2 = rts(2);
10 d = (x2 - x1)/2;
11 x = rts(1)-d.:1:rts(2)+d;
12 y = a*x.^2 + b*x + c;
13 plot(x, y);
14 end

```

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

Name	Value	Min	Max
a	1	1	1
b	-2	-2	-2
c	4	4	4
d	0 + 1.7321i	0 + 1.7321i	0 + 1.7321i
rts	[1 - 1.7321i; 1 + 1.7321i]	1 - 1.7321i	1 + 1.7321i
x1	1 - 1.7321i	1 - 1.7321i	1 - 1.7321i
x2	1 + 1.7321i	1 + 1.7321i	1 + 1.7321i

The problem is apparent: the parabola has complex roots!

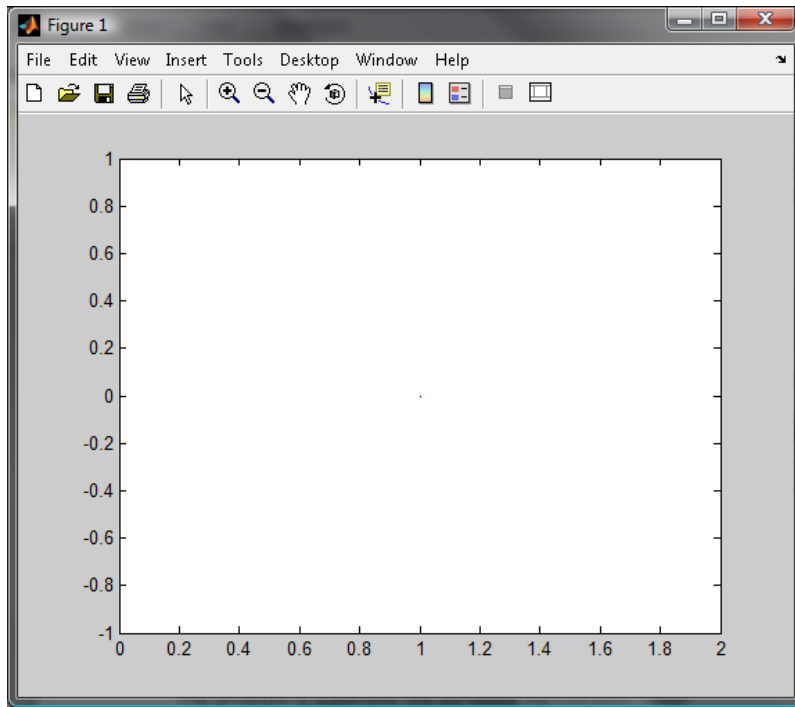
7.4 Exercises

7.4.1 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 $x = -\frac{b}{2a}$.

7.4.2 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.