# Getting started with MATLAB

Peter Lamb

Week 2

## Importing and exporting data

The first stage of the data analysis process is to load the raw data into the MATLAB environment. This can often be one of the hardest parts of the process as there is a huge variety of data file types that you can encounter and often many files need to be combined. We will cover the most common methods of data importing in this section.

## Loading data from text files

Most data collection software has an option to export the data into a text file format (often called ASCII text, Comma Separated Variables (CSV) or Tab delimited). These formats can be viewed in a standard text editor such as Microsoft Notepad, or Apple's TextEdit. The variables are separated into **columns** by a 'delimiter' character, such as a tab, a comma or a semi-colon and **rows** by a new line and/or carriage return character. Comma (often saved with the '.csv' extension) or tab (often saved with the '.dat' or '.txt' extension) delimited formats are the most common. MATLAB has an 'Import data' function that is designed to import a wide variety of data file types.

**Exercise**

1. Locate and copy the folder containing the reference material data for these MATLAB tutorials. Save the contents to a place that you can easily find.

2. Open MATLAB, reset the display and browse for the example files

- From the **Home** tab on MATLAB's ribbon select **Import Data**.
- Use the **Current Directory** window to browse for the folder where you saved the reference material.

3. Load the two example data files into MATLAB's text editor and inspect the differences

- Browse to the folder called **data** within this week's folder structure.
- Right click on the file 'data_csv.csv' and choose 'Open as Text'.
- Do the same for the file 'data_tab.txt' (or simply select 'Open').
- Inspect the two files in the **Editor** window to see the different ways the same data have been saved.

4. Import the data using MATLAB's **Import Wizard**

- Right click on the file 'data_csv.csv' and choose **Import Data...**
- The **Import Wizard** should have correctly set the column separator to 'Comma'. For **Output Type:**, choose 'Numeric Matrix'.
- Click **Import Selection** to complete the import process and create the variable 'data' in the workspace.
- Repeat the process to import the data from the file 'data_tab.txt'.
- You should see two variables called, **datacsv** and **datatab** in the **Workspace** window. Double-clicking on either will open it in the **Variables** window, which allows you to view the contents similar to a spreadsheet.
- Clear the workspace by clicking the **Clear Workspace** button in the **Home** tab or by typing `clear`.
- Import either of the previous files and select 'column vectors' instead of 'Numeric Matrix'. Now you have separate variables (vectors) in the workspace for each column in the file.

## Saving and loading data through a MATLAB data file

When you start to perform more complex and time-consuming analysis you will need to save the changes to your data so that you can continue in future sessions. MATLAB allows you to save all of the variables in your workspace into one file. This MATLAB data file has the '.mat' file extension.

**Exercise**

1. Begin using the text commands and get a list of the current variables.

   · Within the **Command Window** and next to the '>>' symbol, type the command **who** and press enter. You should see the following:

   ```
   >> who

   Your variables are:

   CSVData          TABData
   ```

2. Save the workspace variables into a file called 'Data.mat'.

   · Type the command save Data and press enter.

   ```
   >> save Data
   ```

   · 'Data.mat' will appear as a new file in the **Current Directory** window.
   · Delete the file 'Data.mat' from the **Current Directory** window.

3. Use the same command but this time in the **Function** format

   · Type the function save('Data.mat') and press enter.

   ```
   >> save('Data.mat')
   ```

   · You will notice the information in the brackets is highlighted purple, the single quotes surrounding Data.mat indicate a **string**. MATLAB uses *syntax highlighting* to help you read your code.
   · Use the additional input parameter to save only the variable 'CSVData'. Use the **up arrow** to recall the previous command and then modify it to say save('Data.mat', 'CSVData') and press enter.

   ```
   >> save('Data.mat', 'CSVData')
   ```

4. Clear the workspace and reload the data from the Data file.

   · Clear the workspace using the **clear** command.

   ```
   >> clear
   ```

· Load the variable in the MATLAB data file by double clicking on the 'Data.mat' file in the **Current Directory**. Or type:

```
>> load('Data.mat')
```

## Exporting data to Excel

After completing the data analysis you may need to export the data so it can be used in another program. For example, you may wish to save data into a Microsoft Excel file for viewing, sharing with others or for importing into SPSS or R for statistical analysis.

**Exercise**

1. Investigate and use MATLAB's **xlswrite** function to export the data into an excel file.

   · Load the help documentation for **xlswrite** by typing the following and pressing enter. It may take some time to load the documentation as MATLAB needs to connect to the internet.

   ```
   >> doc xlswrite
   ```

   · Examine the **Syntax** information and **Examples** then type (or copy, paste and modify) the command to export the data in the **CSVData** variable into an excel file called 'TestExcelData.xls'.

   ```
   >> xlswrite('TestExcelData.xls', CSVData)
   ```

   · Open the file 'TestExcelData.xls' in Excel to confirm that the process worked. (NOTE: Double-clicking on the file in the **Current Directory** will begin the **Import Wizard**. To open in Excel, right click the file and choose 'Open Outside MATLAB'.)

# Basic data analysis

## Accessing elements in a matrix

Now that we are able to get the data into MATLAB we are ready to start processing them. In section **??** we introduced the structure of a matrix/array; being a

collection of rows and columns. In MATLAB we describe a matrix as having an **m** number of rows and an **n** number of columns. Where **m** and **n** are used to represent a range of possible values. Let us have a look at a simple matrix with 4 rows and 2 columns; a 4 by 2 matrix or more commonly noted as a $(4 \times 2)$ matrix. Assume that the matrix has the variable name **D**.

$$\mathbf{D} = \begin{bmatrix} a & b \\ c & d \\ e & f \\ g & h \end{bmatrix}$$

Each value in the matrix is called an **element**. To access each element we need to specify which row and which column the element belongs to. For example, to reference the element *'f'* we would specify row 3 and column 2. In MATLAB we would write:

```
>> D(3,2)
```

Now, say we want to reference elements *'a'* and *'b'*, i.e. all the columns in row 1. In MATLAB we can use the colon (:) to specify 'all'. So, in MATLAB we would write:

```
>> D(1,:)
```

This simply tells MATLAB to select row 1 and all columns. To select all the rows in column 2, i.e. elements *'b'*, *'d'*, *'f'* and *'h'* we would write:

```
>> D(:,2)
```

If we wanted to select all of the columns from rows 2, 3 and 4 we could use the colon (:) to specify a range. To select all rows from 2 to 4 we would write, **2:4**. So in order to select the required elements from the matrix we would write:

```
>> D(2:4,:)
```

**Exercise**

1. Use MATLAB's random generator function to create a $(10 \times 4)$ matrix of random numbers. (NOTE: Because we are using a random number generator your matrix values will be different from your classmates!)

- Use the $fx$ button to the left of the >> character in the **Command Window** to search for a 'uniformly distributed random generator' function.
- Examine the help information to determine the correct syntax to create a matrix of random numbers with 10 rows and 4 columns.
- Save the output of the function into a variable called **var**. Use a semi-colon (;) to prevent the output from being displayed on the screen.

```
>> var = rand(10,4);
```

- Here we say we have executed or *'called'* the function **rand** and provided or *'passed'* the two parameters/variables '10' and '4'.
- Double-click the variable **var** in the **Workspace** window to view the values in the **Variable Editor**.
- Extract the value in row 6 and column 3 and store this value into a variable called **m6n3**. Verify, using the **Variable Editor**, that the correct value has been extracted.

```
>> m6n3 = var(6,3);
```

- Extract all the rows from column 3 and store these values into a variable called **n3**. Verify, using the **Variable Editor**, that the correct values have been extracted.

```
>> n3 = var(:,3);
```

- Change the element in row 5 of column 2 to the value 999. Verify, using the **Variable Editor**, that the correct value has been changed.

```
>> var(5,2) = 999;
```

- When you have finished, clear the workspace using the **clear** command.

```
>> clear
```

## Using basic mathematical operators

MATLAB can interpret a number of basic operators to provide simple data calculations without having to use the in-built functions. We will look at the 5 most common; *Addition*, *Subtraction*, *Multiplication*, *Division* and *Power*.

| Operator | Typed Character |
|---|---|
| Addition | + |
| Subtraction | − |
| Multiplication | * |
| Division | / |
| Power | ^ |

Here are some examples of how to use these operators from the **Command Window**. Set two variables **a** and **b** to the values 2 and 3 respectively.

```
>> a = 2;

>> b = 3;
```

To add the two variables, **a** and **b**, to create a new variable **c** we would write:

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

To subtract variable **a** from variable **b**, to create a new variable **c** we would write:

```
>> c = b - a;
```

To multiply two variables, **a** and **b**, to create a new variable **c** we would write:

```
>> c = a * b;
```

To divide variable **a** by variable **b**, to create a new variable **c** we would write:

```
>> c = a / b;
```

To create a new variable **c** as **a** to the power of **b** we would write:

```
>> c = a^b;
```

## Plotting the data

In this section we will look at how to plot graphs of some real data and use these plots to inspect the data more closely. The data that we loaded in the previous section, in the variable **CSVData**, contains EMG data from 6 muscles with the values expressed in Volts. The first column of the data is the frame number and there are 200 frames per second. There is something slightly wrong with the data and this will need correcting before any further analysis can be performed.

**Exercise**

1. Load the file 'Data.mat' to add the variable **CSVData** back into the workspace.

2. Use the plot tool from the **Variable Editor** to plot the EMG data in the second column.

   · Display the variable **CSVData** in the **Variable Editor**.

   · Highlight the second column by clicking on the number '2' at the top of the column.

   · In the **PLOTS** tab, click the plot button  to display a graph of the data.

   · Close the plot window by clicking the red 'X' in the top right corner (or the red button on a Mac).

   · Take note of the output from the plot tool that has appeared in the **Command Window**.

3. From the **Command Window** use the plot function to plot the EMG data in the second column.

   · Type the following:

   ```
   >> plot(CSVData(:,2));
   ```

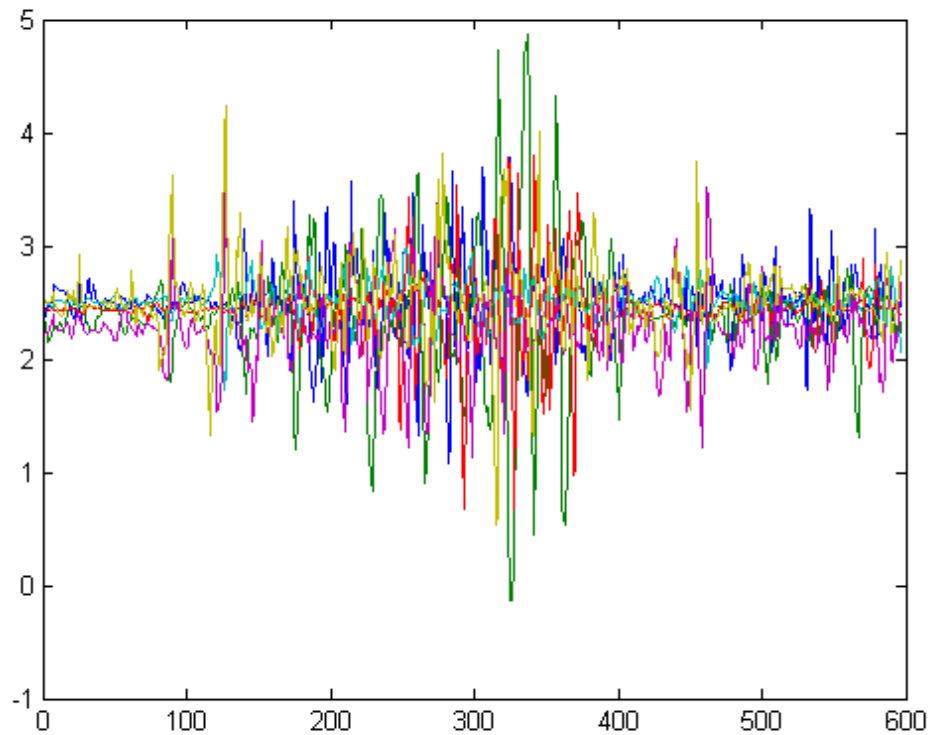   · You should see the same plot as before:
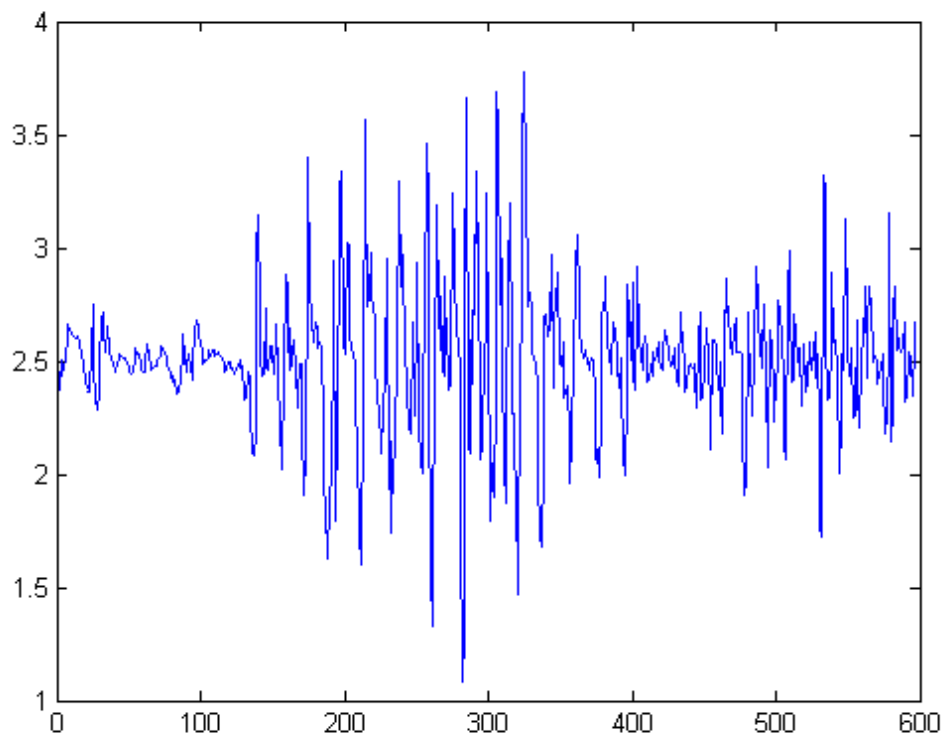
4. Use the plot function to plot all six of the EMG channels on a single graph. Keep this plot or **figure** open for later.

- Type the following:

```
>> plot(CSVData(:,2:7));
```

- Notice how none of the EMG signals are centred around zero Volts.

## Processing the data

You should now be getting the hang of how to access the contents of the data. We will now look at how to perform some basic analysis on these EMG data. Let us presume our aim is for the EMG data for each muscle to be plotted on separate graphs with time, in seconds, along the $x$-axis and the baseline corrected, rectified EMG amplitude signal, in mV, on the $y$-axis. From what we have seen from the data we have four main tasks to complete before we can display the results.

1. Correct the first column of data from frame numbers to time, in seconds.

2. Baseline correct the EMG signals so that they are centred around zero.

3. Rectify the EMG signals (convert values to positive amplitudes).

4. Convert the Volts (V) to mV for each EMG signal.

**Exercise**

1. Duplicate the variable **CSVData** to create an identical new variable called **emgdata**.

   · Copy the **CSVData** to a new variable **emgdata**. Type the following:

   ```
   >> emgdata = CSVData;
   ```

2. Change the values in the first column to show the time in seconds.

   · Divide all of the values in column 1 by 200 to convert from frames to seconds. Type the following:

   ```
   >> emgdata(:,1) = emgdata(:,1)/200;
   ```

3. Baseline correct the EMG signals by centring them at zero.

   · Use MATLAB's **mean** function to find the mean value for each EMG signal. Type:

   ```
   >> emgmeans = mean(emgdata(:,2:7));
   ```

   · Delete the mean value for the EMG signal from all of the values. After pressing **Enter** to submit the first line, use the up arrow to prevent re-typing the whole line again.

   ```
   >> emgdata(:,2) = emgdata(:,2) - emgmeans(1);
   >> emgdata(:,3) = emgdata(:,3) - emgmeans(2);
   >> emgdata(:,4) = emgdata(:,4) - emgmeans(3);
   >> emgdata(:,5) = emgdata(:,5) - emgmeans(4);
   >> emgdata(:,6) = emgdata(:,6) - emgmeans(5);
   >> emgdata(:,7) = emgdata(:,7) - emgmeans(6);
   ```

4. Use the plot function to plot all six of the corrected EMG channels on a single graph in a **new figure**.

   · Type the following:

   ```
   >> figure;
   >> plot(emgdata(:,2:7));
   ```

   · Notice how all of the EMG signals are now centred around zero Volts. You should be able to compare this plot with the plot from the previous exercise.
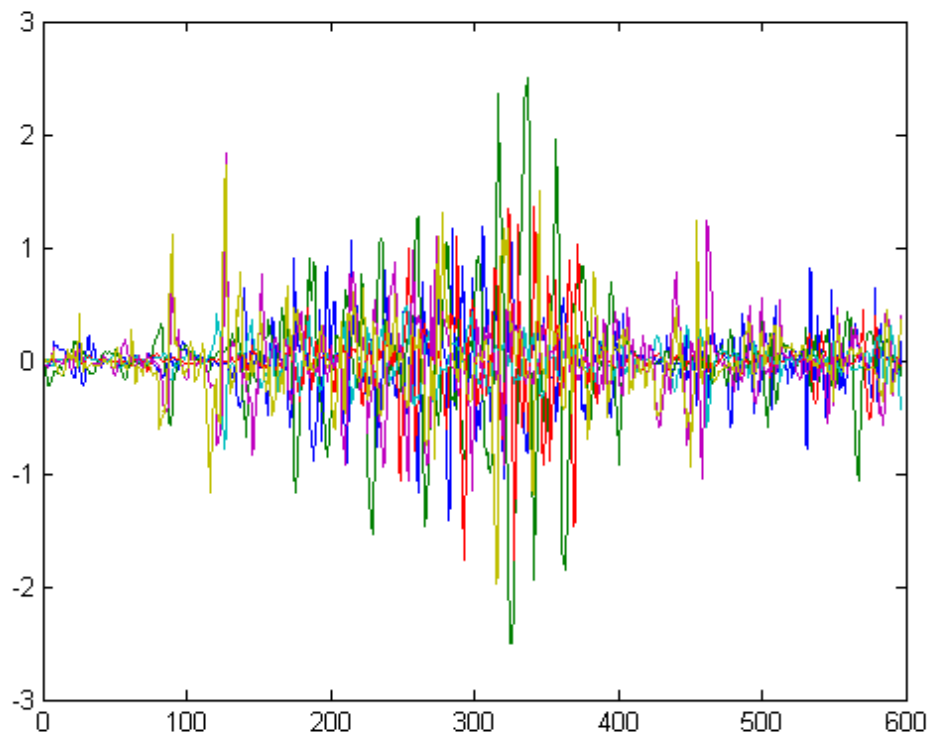
5. Rectify all of the EMG data, i.e. get only the magnitude of the signal, ignoring whether this is positive or negative. This is also known as getting the **absolute value**.

  · Close all of the current figures by using the **close** function.

```
>> close all
```

  · Use the *fx* button next to the command prompt (>>) to search for a MATLAB function to find the **absolute magnitude** of each EMG value.
  · Use this function to rectify the EMG data.

```
>> emgdata(:,2:7) = abs(emgdata(:,2:7));
```

6. Convert the EMG signals from Volts (V) to milli Volts (mV).

  · Multiply all of the values in the EMG columns by 1000. Type the following:

```
>> emgdata(:,2:7) = emgdata(:,2:7)*1000;
```

7. Plot the EMG data for the first EMG signal. The time column should be on the $x$-axis and the EMG amplitude in mV should be the $y$-axis

   · Use MATLAB's help documentation to find out how to plot the $x$ and $y$ data on the required axes.

   · Use the **plot** function to plot the graph of signal 1 (found in column 2 of the variable **emgdata**).

   ```
   >> plot(emgdata(:,1), emgdata(:,2));
   ```

   · Using the help documentation find out how to add 'Time (s)' as an $x$-axis label, 'EMG amplitude (mV)' as a $y$-axis label and 'The amplitude of Signal 1' as a title for the current plot.

   ```
   >> xlabel('Time (s)');
   >> ylabel('EMG amplitude (mV)');
   >> title('The EMG amplitude of signal 1');
   ```

   · We could now repeat this process for each of the other five signals but this would be repetitive and time consuming. In the next section we will look at a number of ways to prevent the repetition and speed this process up.
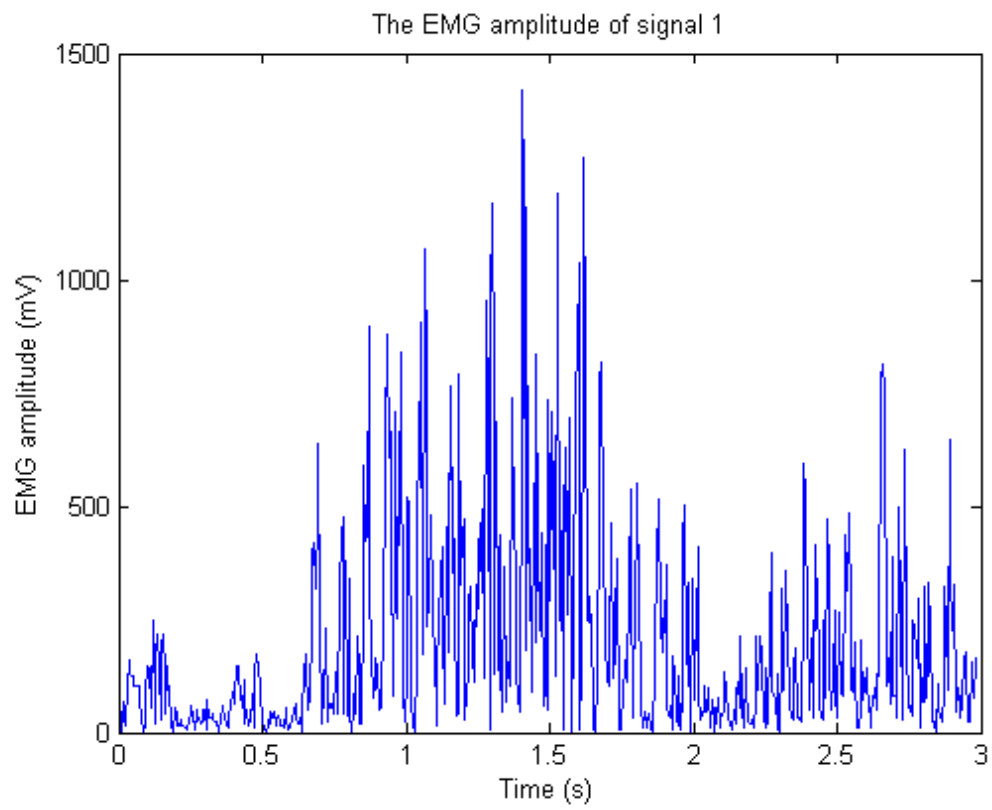
13

Figure 1: A plot of the EMG amplitude of Signal 1 against time with axis labels and a title