Understanding MATLAB scripts and functions

Peter Lamb

Week 3

Scripts

A MATLAB script is simply a list of commands or instructions. Script files are saved as text files with the file extension **.m** to associate them with MATLAB.

Executing a MATLAB script file

An example script file has already been created and can be found in **3-Scripts_and_functions\code.**.

Browse to this folder and then load the script file called 'emgscript.m' into the **Editor**. The commands that make up this script should look familiar from the previous chapter. We can use this file to run all of these commands at once. Make sure '3-Scripts_and_functions' is your **Current Folder** (more on this later); from within the **Editor** window click the **Run** icon from the toolbar. If a dialog box opens, click **Add to Path** rather than **Change Folder**. You should see a figure appear containing the plot we produced in the previous chapter.

Because we have seen the commands before the script should make sense to us. However, someone who saw the script and those commands for the first time may find the code a bit confusing. MATLAB allows us to add text to these files that will not perform any actions. Putting the % character at the beginning of a line instructs MATLAB to ignore any following text it finds on the same line. This enables us to add a description/comment above a single or group of commands to remind us what is going on. Have a look at the file called 'emgscriptwithdescription.m' to see how comments can make a file easier to understand.

Creating your own MATLAB script file

We will now modify the existing script file to plot the first three of the EMG signals, each in a separate plot but within the same figure – a subplot. When MATLAB executes the **plot** function it draws the requested plot in the currently activated axis. If no axis has been created and activated, say after a blank figure has been created using the **figure** command, MATLAB draws the plot in a new axis. In order to create a number of plots within the same figure we can use the **subplot** command to create a series of new axes into which we can draw the required plots.

Exercise

1. Use MATLAB's help documentation to find out how to use the **subplot** function.

2. Save a copy of the file 'emgscriptwithdescription.m' and use this as the base for the following additions. Choose any file name you like.

3. Before the first plot on line 33, add a **subplot** command to specify that the figure should have a 3 row, 1 column arrangement and to plot into the first axis.

   · Type the following:

```
subplot(3,1,1);
```

   · Add a comment above the **subplot** command to describe what action is being performed.

```
% Setup the figure to have subplots arranged
% in 3 rows and 1 column and then activate
% subplot 1
```

4. Repeat this process to plot the next two signals in the second and third slots in the figure.

   · Type the following:

```
% Activate the second subplot in the
% arrangement
subplot(3,1,2);
```

```
% Plot the EMG signal 2 against time and add
% axis labels and a title
plot(emgdata(:,1), emgdata(:,3));
xlabel('Time (s)');
ylabel('EMG amplitude (mV)');
title('The EMG amplitude of signal 2');

% Activate the third subplot in the arrangement
subplot(3,1,3);

% Plot the EMG signal 3 against time and add
% axis labels and a title
plot(emgdata(:,1), emgdata(:,4));
xlabel('Time (s)');
ylabel('EMG amplitude (mV)');
title('The EMG amplitude of signal 3');
```

5. Run this new script (in my case I called it 'myscript.m'), either by clicking the **Run** button from the toolbar of the **Editor**.

```
>> myscript
```

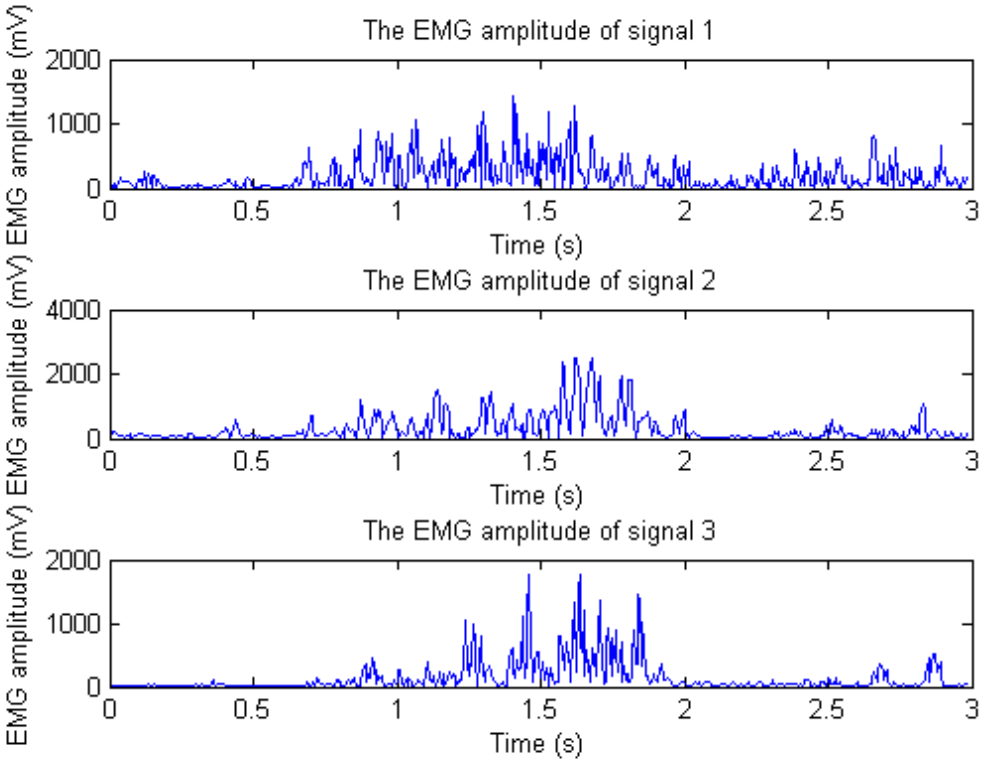


Figure 1: A figure showing three subplots of the first three EMG signals

Functions

We have already had some experience in using MATLAB's in-built functions. Let us now take a closer look at how these functions work. However, in order to get a good look at how these functions work we will have to look at a function that is not an in-built one. This is because all of the code in MATLAB functions is hidden. Take a look at the file 'my_abs.m' in the 'code' subfolder, which contains the function **my_abs**. The aim of this function is to perform exactly the same operation as MATLAB's **abs** function that we have used previously.

As you can see, this function appears similar in many ways to the **scripts** we have just been using. However, there are some fairly significant differences. The main thing to notice is that the function is defined at the start of the file using the command **function**. This is a requirement and this line must be the first

command that MATLAB encounters (ignoring any comments). The function declaration for the example in 'my_abs.m' is the following:

```
function abs_data = my_abs(input_data)
```

This line instructs MATLAB in a few ways.

- It instructs MATLAB that what follows is a function.
- It declares an output variable called **abs_data**. This instructs MATLAB that at the end of the function it should return the value contained in the variable **abs_data**.
- It states the name of the function, in this case **my_abs**.
- It creates a variable called **input_data**. This will be the value that is passed to the function when it is called. For example if we call the function **my_abs** by typing:

  ```
  >> testdata = my_abs(emgdata);
  ```

  then within the function the variable **input_data** would be exactly the same as the variable **emgdata**.

Try using this function as we have before by typing[1]:

```
>> addpath('code')
>> testdata = my_abs(emgdata);
```

Notice how we only see the variable **testdata** appear in the **Workspace** and not any of the intermediate variables e.g. **squared_data** or **square_root_squared_data**. This is one of the main benefits of a function. However, in the same way that we do not see any of the function variables in the workspace, we also do not see any of the workspace variables in the function. The only workspace variables that can be used in the function are the variables that are passed into the function. Similarly, the only function variables available back in the workspace are the ones returned from the function. All clear!? If not, try not to worry too much, that is about as complicated as things will get with function use and you will see and experience many more examples of these principles.

[1] More on the line **addpath('code')** later

Writing your own functions

Using all of the skills we have gathered so far we will write a new function called **rectify** that will rectify EMG signals. MATLAB has a tool that will help to create the template of a function automatically. Either in the **Home** tab or the **Editor** window, click the dropdown arrow under **New** and choose **Function**. For now we will assume that the input to the **rectify** function is just a single column vector.

Exercise

1. Create a new Function and name the function **rectify**
   - From the **Home** tab on MATLAB ribbon select **New** and then from the drop down list select **Function**.
   - MATLAB will create a new file and add the required code and an example function name. Change the function name (declared on the first line of the new file) from **Untitled** to **rectify**.
   - Change the function input and output variables to more meaningful names. E.g. `raw_emg` as the input and `rectified_emg` as the output (just helps with readability).
   - Replace the function comments with a brief description of what we are trying to achieve.
   - Click the **save** button. Choose the **code** folder (Notice how MATLAB has already chosen the name of the file as 'rectify.m'). Click **Save**.
2. As we are creating this function look at some of the ways that the **Editor** helps us.
   - Notice how the input and output variables are underlined in orange. The bar to the right of the window has an orange square and a line next to the function declaration.
   - Try hovering the mouse icon over the underlined text and see what information MATLAB gives us.
   - It tells us that one variable is not used (`raw_emg`) and the other is unset (`rectified_emg`).
   - Orange underlining is MATLAB's way of letting us know that although the code may work we may be in need of some helpful warnings.

- Red underlining lets us know that MATLAB has encountered some commands that will not work and will result in errors.
- Try typing something ridiculous and see if you can get the **Editor** to give you some red writing.
- Remember to delete your nonsense!

3. Add the commands that will baseline correct and rectify the signal.

   - The function **mean** will calculate the average value of the signal. This can be used to baseline correct.

   ```
   baseline = mean(raw_emg);
   baseline_corrected_emg = raw_emg-baseline;
   ```

   - The function **abs** will calculate the absolute value of the signal. This can be used to rectify.

   ```
   rectified_emg = abs(baseline_corrected_emg);
   ```

4. Hopefully you should see a green square at the top of the right hand border to indicate that no warnings or errors have been found.

5. Click the **save** button to save your changes.

6. Test out the function with a column from the variable **datacsv**.

   - Plot some raw data from **datacsv**

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

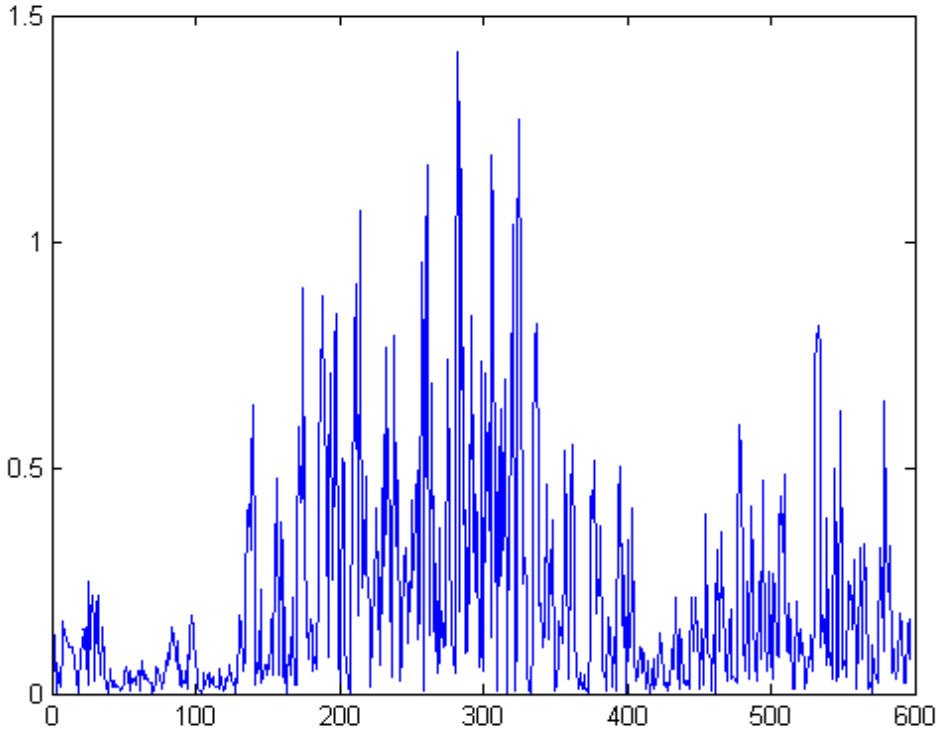
   - Rectify the same raw data

   ```
   >> testdata = rectify(CSVData(:,2));
   ```

   - Plot the new rectified data

   ```
   >> plot(testdata);
   ```

   - Depending on which column you decided to plot you should see something like this:


1.5
1
0.5
0
0
100
200
300
400
500
600