Understanding MATLAB statements

Peter Lamb

Week 4

As we have previously hinted, the real power of MATLAB is in automating the repetitive tasks that are often involved in data analysis. Using MATLAB for this process has four main benefits:

1. Reduces the overall time to perform the analysis.
2. Eliminates human errors by guaranteeing accuracy during repetition.
3. Once an analysis script/function is complete it enables the whole analysis process to be repeated rapidly from the original raw data. This allows any additional data to be included into the analysis.
4. Detailed documentation is produced in the form of the MATLAB code.

MATLAB has a few coding tools or **statements** that can allow us to build loops to repeat a process many times. We will cover the most common and useful ones in this section.

Don the Star Wars t-shirt, we are entering the world of computer programming!

The if... elseif... else... Statements

This is the world's most common computer programming statement. It remains pretty much consistent across all of the major programming languages. It follows from the logical statements that we encounter in our human world every day. If the answer to a question is 'yes', do this, if the answer is 'no', do something else. For example, you are at a pedestrian crossing (and going to follow the rules to the letter!). If the crossing light is green, cross... else, don't cross. Simple. Writing this in language that MATLAB understands would look like this:

```
% crossing_light has the value 1 when the crossing
% light is green and the value 0 otherwise
if  crossing_light == 1
    cross_the_road; % hypothetical script of commands
else
    dont_cross_the_road;
end
```

There are a few things to notice here. The commands that set out the statement are `if`, `else` and `end`. The `end` command is there to inform MATLAB that any commands after the statement are not exclusive to the `else` condition, i.e. the `if`... `else` statement has finished. The `if` command is followed by the 'question' which should only return a true or false value. When using the 'Equals' expression you must use double = signs, e.g. ==. A list of common expressions and an example for each is given below:

Expression	MATLAB code	Example
Equals	==	`if` a == 5
Less than	<	`if` a < 5
Greater than	>	`if` a > 2
Less than or equal	<=	`if` a <= 5
Greater than or equal	>=	`if` a >= 5.2
Does not equal	~=	`if` a ~= 5

Sometimes the question cannot be broken down as simply as the first example. Imagine this time that you are driving a car and are approaching a set of traffic lights. You could break the situation down in the following way. If the light is green, you drive on. Else, if the light is red, you stop. Else, if you are already very close to the lights, you drive on, else, you stop. This introduces another `if` expression to evaluate, the 'If the light is red' question. In MATLAB code it would look something like this:

```
% traffic_light has the value 1 when the traffic light
% is green, the value 2 if the traffic light is amber
% and the value 3 if the light is red, proximity is
% the distance to the lights in metres.
if  traffic_light == 1 % the traffic light is green
    drive_on;
elseif traffic_light == 3 % the traffic light is red
    stop;
else % the traffic light must be amber
    if proximity < 10 % close enough to drive through
        drive_on;
    else % it is not safe to drive through
        stop;
    end
end
```

Combining the lessons on functions in the last section and what we have just learned about `if`... `else` statements, we will write a function to advise someone how to deal with traffic lights.

Exercise

1. Begin by creating a new Function M-File called ‘driving.m’.
   - From the **Home** tab on MATLAB ribbon select **New** and then from the drop down list select **Function**.
   - Change the name of the function to **driving**
2. Declare the input and output variables.
   - Set the input variables, using a comma ‘,’ to separate the two. Use `traffic_light` as the colour of the traffic light and `proximity` as the distance to the lights.
   - Declare an output variable, `advice` to contain a text value of what should be done, either ‘Drive on...’ or ‘Stop!’.
3. Write the `if`... `else` statements, setting the output variable to the required value where necessary.
4. Add some comments to the file to make it clear what is going on.
5. Save the file and then test the results.

```
>> advice = driving(1,5)

advice =

    Drive on...
```

Your finished function may look slightly different but should look something like this:

```
function advice = driving(traffic_light, proximity)
%DRIVING Advice on what to do when approaching traffic
% lights
%    The response is based on two input parameters:
%    1)  The state of the traffic lights.
%        Green = 1, Amber = 2 and Red = 3.
%    2)  The proximity of the lights in metres.
%
%    The output text is one of the following:
%        'Drive on...' or 'Stop!'
%

if traffic_light == 1 % the traffic light is green
    advice = 'Drive on...' ;
elseif traffic_light == 3 % the traffic light is red
    advice = 'Stop!';
else % the traffic light must be amber
    if proximity < 10 % close enough to drive through
        advice = 'Drive on...';
    else % it is not safe to drive through
        advice = 'Stop!';
    end
end
```

The For... Loop Statement

This statement is probably the most effective statement in saving time during data analysis. It is designed to repeat a set of commands a number of times. Let us revisit the example that finished Chapter 2. In that example we had created a graph to show the amplitude of one of the signals from our EMG data file. We had noticed that to repeat the process and show the plots for the remaining five signals would be repetitive and time consuming. The for loop will allow us to reuse the same lines of code without having to type them out a number of times.

To recap what we were doing, take a look at the code below. This function makes use of the **rectify** function that we developed last week.

```
function plot_emg(filename)
%PLOT_EMG Building on the EMG analysis from Week 2

% Load in the data from the specified text file
emgdata = csvread(filename);

% Convert the time column from frames to seconds
emgdata(:,1) = emgdata(:,1)/200;

% Baseline correct and Rectify the EMG data and convert
% the values from volts to millivolts
emgdata(:,2) = rectify(emgdata(:,2))*1000;

% Plot the graph of the signal with a title and axis
% labels
plot(emgdata(:,1), emgdata(:,2));

title(sprintf('The EMG amplitude of signal %d', 1));

xlabel('Time (s)');

ylabel('EMG amplitude (mV)');

% Save the current figure as a png file
saveas(gcf, sprintf('signal %d.png', 1),'png');
```

Most of this code should be familiar. However, we should take a closer look at the lines that use the **sprintf** and **saveas** functions. Once a figure has been created we can use the **saveas** function to save the plot as an image file. This image can then be used in reports or presentations. In order to use the function we must provide three variables; 1) the current figure, here we use the **gcf** function to ‘**G**et **C**urrent **F**igure’, 2) a name to call the output file and 3) some information as to the type of file we are wanting to create, in this case we use ‘jpg’ to indicate a **jpeg** image. The **sprintf** function allows us to create a text output by using **place-holders** and variables. The %d characters are used to specify the place-holder and also what type of variable is required. In this case we use %d to specify a whole number (see the documentation for **sprintf**, specifically the section about ‘formatSpec’). We can then pass the variables that will appear as the place-holders. This allows the majority of the text to be repeated but still allows certain values to change. The usefulness of this function will become apparent shortly.

Before we begin, consult the MATLAB documentation to get an explanation and some examples of how the `for` statement is used. The aim of this section is to plot and save the image for each of the six EMG signals.

Exercise

1. Modify the file ‘plot_emg.m’ using the `for` loop to save the figures from all of the six EMG signals.
   - Enclose the last three sections of the function in a `for`... `end` loop.
   - Set the looping variable as **column** and increment the value from 2 to 7.
   - Create a new variable, **signal**, to store the signal number as **column**-1.
   - Substitute references to the signal number 1 with the variable **signal** so it increments as we loop through the function.
   - Substitute references to column 2 with the variable **column** so it too increments as we loop through the function.
2. Change the function name to ‘plot_emgs’ and save to a new file with the same name.
3. Use the **csv** file to test the function.

```
>> plot_emgs('data/data_csv.csv');
```

These relatively slight modifications allow us to repeat these data analysis and plotting steps over and over again. In this example we have only repeated the steps six times but we could repeat this any number of times. The finished function should look something like this:

```
function plot_emgs(filename)
%PLOT_EMGS Building on the EMG analysis from Week 2

% Load in the data from the specified text file
emgdata = csvread(filename);

% Convert the time column from frames to seconds
emgdata(:,1) = emgdata(:,1)/200;

for column = 2 : 7
    % Set the signal number as column - 1
    signal = column-1;

    % Baseline correct and Rectify the EMG data and
    % convert the values from Volts to milli Volts
    emgdata(:,column) = rectify(emgdata(:,column))*1000;

    % Plot the graph of the signal with a title
    % and axis labels
    plot(emgdata(:,1), emgdata(:,column));

    title(sprintf('EMG amplitude of signal %d',signal))

    xlabel('Time (s)');

    ylabel('EMG amplitude (mV)');

    % Save the current figure as a png file
    saveas(gcf, sprintf('signal %d.png',signal),'png')

end
```

The Switch Statement

The `switch` statement is another way of offering a number of different code branches depending on the state of a variable. Using the `switch` statement informs MATLAB to inspect the variable and offers separate code chunks in various different **cases**. Using the EMG example we have been following through, imagine that the signals 1-6 are labelled as described in the following table.

Signal number	Muscle	Shorthand
1	Gluteus medius	GMED
2	Gluteus maximus	GMAX
3	Rectus femoris	RF
4	Adductor magnus	AM
5	Sartorius	ST
6	Biceps femoris	BF

It would be better if we could label any graphs that use these signals so that they use the name of the muscle, say 'Gluteus medius' or 'GMED' as opposed to just 'Signal 1'. The `switch` statement would allow us to do this easily.

Exercise

1. Create a copy of the file 'plot_emgs.m' and call it 'plot_muscles.m'.
   - Remember to modify the function name at the top of the file to match the file name.
2. Modify this function using the `switch` statement to add the correct name to the title of the six EMG signal graphs.
   - Create a switch statement using the variable **signal** within the `for` loop and just after the **signal** has been declared.
   - Use six cases for each of the possible **signal** values 1 to 6.
   - Use a variable called **muscle** to store the text for each muscle depending on which `case` is followed.
3. Replace the reference to **signal** in the remainder of the `for` loop with the muscle name
4. The '%d' place-holder (used for whole numbers) will need to be replaced with the '%s' place-holder (used for text strings).
5. Test the function to see how it works.

```
>> plot_muscles('data/data_csv.csv');
```

The finished function should look something like this:

```
function plot_muscles(filename)
%PLOT_MUSCLES Building on the EMG analysis from Week 2

% Load in the data from the specified text file
emgdata = csvread(filename);

% Convert the time column from frames to seconds
emgdata(:,1) = emgdata(:,1)/200;

for column = 2 : 7
    % Set the signal number as column - 1
    signal = column - 1;

    % A switch statement to get the muscle name
    switch signal
        case 1
            muscle = 'GMED';
        case 2
            muscle = 'GMAX';
        case 3
            muscle = 'RF';
        case 4
            muscle = 'AM';
        case 5
            muscle = 'ST';
        case  6
            muscle = 'BF';
        otherwise
            muscle = 'UNKNOWN';
    end

    % Baseline correct and Rectify the EMG data and
    % convert the values from volts to millivolts
    emgdata(:,column) = rectify(emgdata(:,column))*1000;
```

```matlab
    % Plot the graph of the signal with a title and
    % axis labels
    plot(emgdata(:,1), emgdata(:,column));

    title(sprintf('The EMG amplitude of signal %s', muscle
       ));

    xlabel('Time (s)');

    ylabel('EMG amplitude (mV)');

    % Save the current figure as a png file
    saveas(gcf, sprintf('muscle %s.png',muscle),'png')

end
```