Reproducible research

Peter Lamb

Week 6

MATLAB offers many features that can help make your analysis run smoothly and efficiently. In this module we will look at a raw data set and create a script to perform several procedures. The benefit of scripting an analysis is that it allows you to easily trace back all of the operations performed on the data, which makes pinpointing mistakes much easier. The example in this section uses time-series data (one trial consists of many measurements: length of time for a trial × number of samples taken per unit of time), which are particularly difficult to manage using spreadsheets. Imagine having Excel sheets (rows for time samples, columns for variables) for each trial and different files for each participant: finding mistakes would be extremely painstaking, fixing a simple mistake that occurs on every sheet would take days, if not, weeks of work. By automating your analysis with a script, the same procedures can be performed multiple times, i.e. in a loop, and your single mistake only needs to be fixed in one spot. We will also look at two options to help share your code with supervisors and collaborators.

Script for a single trial

Commenting and organisation

Before we start our script, let's take a bit of time to think about how best to make our script easy to read and navigate. We've already seen how to include comments in Week 3. Comments allow you to describe in words what your code is doing and are preceded by `%`, anything on the line afterwords is not executed. Any script or function can also be divided into sections using `%%`. Sections make reading a long script easier as the different steps are clearly divided and highlighted by the MATLAB editor (see here for more details). Sectioning also lets you run the code for one section at a time, which is helpful in larger projects. It is often a good idea, before you start writing the code for your script, to write all the comments in as much detail as possible. This will help you stick to the plan, even when you get bogged down with annoying little bugs and problems.

Reading in or loading data

To make our analysis reproducible we want it to run through its entirety just by running the script. This means we will not use the import wizard to read in our data, as this would introduce the possibility of reading in an old or wrong dataset, forgetting where on the computer the data are stored, how the data are formatted, etc.

First, I encourage you to adopt a file and folder organisation system. For example, create a parent folder on your computer named after the project, and within that folder you might have a **code** folder, a **data** folder, a **documents** folder, and a **plots** folder (see Figure 1 for an example). In this case keep all your data, whether raw or processed in the **data** folder, m-files (scripts or functions) in your **code** folder, and so on.

Assuming this file and folder organisation, let's navigate to our parent folder – in our case the **6-Reproducible_research** folder. Next, to read in the file 'P1_C2_T1.txt' in the **data** folder we will create a *string* for the full file path. Navigating to the parent folder is important because the filename below is relative to the current directory.

```
filename = 'data/P1_C2_T1.txt';
```

There is, however, one potential problem if you want to share your data and code with a collaborator or supervisor. The forward slash direction '/' in the filename variable is specific to Mac (Unix actually) operating systems; Windows operating systems use backslashes in filenames. An alternative method that makes your code more portable and shareable is to use the **fullfile** function.

```
>> filename = fullfile('data','P1_C2_T1.txt')

filename =

    'data/P1_C2_T1.txt'
```

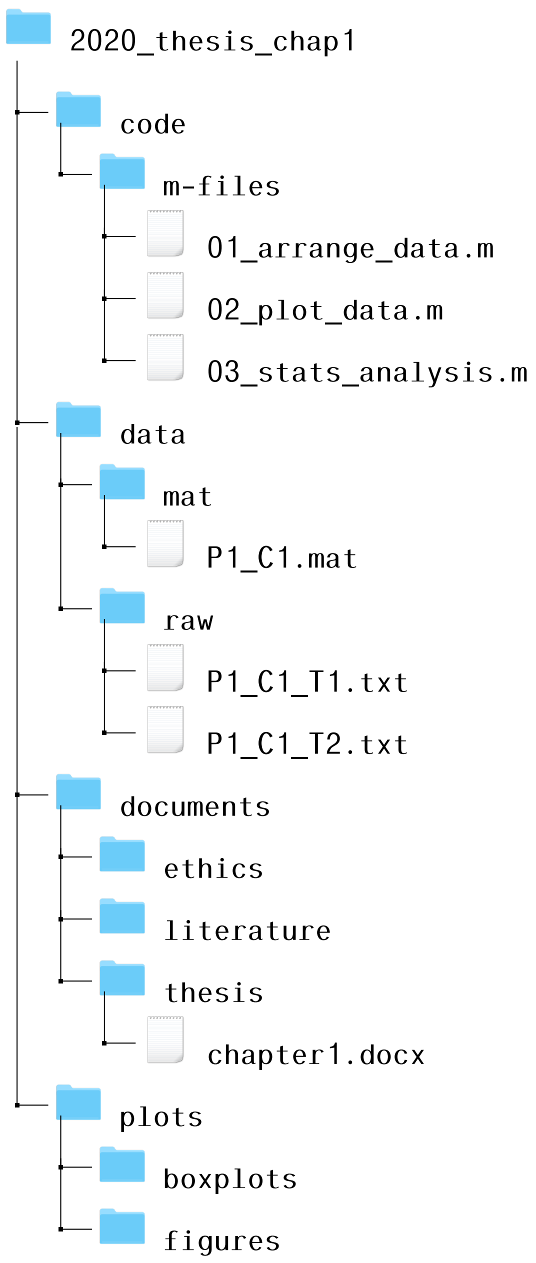

Figure 1: Example project directory organisation

If run on a Windows operating system, the slash will be changed. Now we have a string referring to the file we want to read – notice that the instructions for creating the string are all part of the code, no interaction with dialog boxes is needed, we just need to have the project folder as our **Current Folder**.

Reading in data from formatted files can be the most difficult part of scripting an analysis. For an overview of the various 'input/output functions', type `>> doc iofun`. If you open the text file with a text editor, you will notice there are several lines describing the participant, equipment and data collection parameters. On line 10 the numeric data begins, with rows corresponding to time samples, or measurements, and commas separating each variable until the final variable, at which point the line ends. The data are several kinematic variables describing a golf swing. To read in our text file we will use the **csvread** function. Again, see the documentation for how to use the function. Since we want to start reading in data from row 10, we want to employ a row offset of 9. We will read the data into MATLAB and store it as a matrix in a variable named **golfData**.

```
golfData = csvread(filename, 9, 0);
```

You should now be able to see the variable **golfData** in the workspace, where it is described as **486** $\times$ **37 double**. Double refers to the precision of the variable and has implications for memory use. Considering the precision is only really important when dealing with very large datasets (e.g. >5 GB).

Extracting discrete data

We have read in 486 time frames of 37 time-continuous variables describing one golf swing by one participant; in this example we are interested in extracting a few key variables. In golf swing biomechanics there is an important variable known as the 'X-Factor', which represents the difference in axial rotation between the upper torso and the pelvis segments usually taken at the start of the downswing (see Figure 2). When the downswing starts – **defined by the change in rotation direction of the pelvis** – the pelvis usually leads the rotation causing the X-Factor to increase.

We want to extract the following variables from the dataset:

1. the X-Factor value (at the start of the downswing),
2. the maximum X-Factor value,
3. the time between the start of the downswing and the maximum X-Factor value.

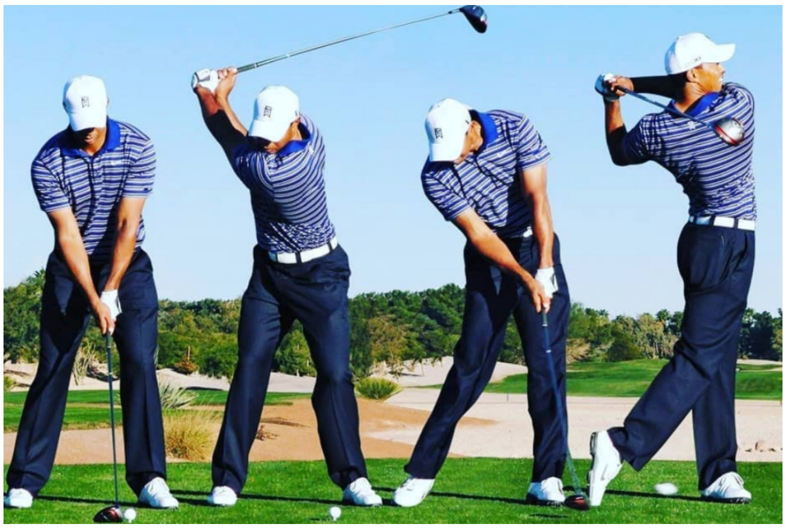

Figure 2: Depiction of the X-factor variable. The second image in the sequence shows Tiger Woods at the transition from the backswing to the downswing, defined by the change in direction of pelvis rotation. Notice the difference in pelvis angular position compared to the thorax; this difference is known as the *X-factor* and it is thought to initiate the stretch-shortening cycle in the early downswing.

While previewing the file 'P1_C2_T1.txt' you may have noticed that the sampling frequency was recorded as 240 Hz; this is needed to compute the time elapsed between frames. We will start by writing out the comments to direct our thoughts on how to code this exercise.

```
% Calculate the X-Factor throughout the swing


% Determine the frame number of the start of the
% downswing and extract the X-Factor value.


% Calculate the maximum X-Factor (actually minimum
% because angles are negative in the backswing).


% Determine the number of frames between the start of the
% downswing and the maximum X-Factor.

% Convert from frames to seconds.
```

Seems simple enough, let's get to it. But first a couple notes about the dataset: pelvis axial rotation is column 14, and thorax axial rotation in column 17. By biomechanical convention, clockwise rotation is negative and counterclockwise is positive. Imagine a line passing through the hip joint centres and a line passing through the shoulder joint centres. At the address position (first image in the sequence in Figure 2) these lines would be roughly parallel with the target line (line between the ball and target) – this is our reference position, so parallel to the target line is represented by an angle of 0°. When the golfer turns these segments away from the target the angle will be negative and when the golfer is turned toward the target the angle will be positive. Plot columns 14 and 17 to see for yourself.

```
% Calculate the X-Factor throughout the swing

xfVector = golfData(:,17) - golfData(:,14);
```

The variable **xfVector** is the time-continuous difference between the pelvis and thorax angles.

```
% Determine the frame number of the start of the
% downswing and extract the X-Factor value.

[~,minIndex] = min(golfData(:,14));

xf = xfVector(minIndex);
```

Here we used the **min** function because the values for this part of the swing are negative. We then extracted the value of **xfVector** at the frame number held in **minIndex**.

```
% Calculate the maximum X-Factor (actually minimum
% because angles are negative in the backswing).

[xfMin,xfMinInd] = min(xfVector);
```

Here we use the **min** function to get both the value and the frame number of the minimum value of **xfVector**.

```
% Determine the number of frames between the start of the
% downswing and the maximum X-Factor.

```

```
framesToMin = xfMinInd - minIndex;

% Convert from frames to seconds.

timeToMin = framesToMin / 250;
```

The variable **timeToMin** is 0.2160, which tells us that the maximum separation of the pelvis and thorax occurred 0.216 s into the downswing of the measured swing.

Likewise, we can define any variable for extraction according to our research question. In a large study with multiple participants and trials, this process of reading in data and calculating a dependent variable can be performed automatically by using carefully set up loops.

Visualising data

It is always a good idea to plot your data, either to include in your write-up or just as a way to double check that what you have calculated makes sense. See the **figure** and **plot** documentation if any of the commands below are new or confusing.

```
% Plot data
figure('Units', 'centimeters',...
    'Position', [5 5 12 8])

frames = 1 : length(golfData);

time = frames / 250;

plot(time, golfData(:,14), 'b-')

hold on

plot(time, golfData(:,17), 'r-')

plot(time, xfVector, 'k--')

plot(time(minIndex), minVal, 'ro')

plot(time(xfMinInd), xfMin, 'bo')
```

```
xlabel('Time (s)'); ylabel('Angle (deg)')

legend({'pelvis','thorax','x-factor',...
    'start downswing','max x-factor'}, ...
    'Location', 'best',...
    'Box', 'off')
```

The above code should produce the following plot.

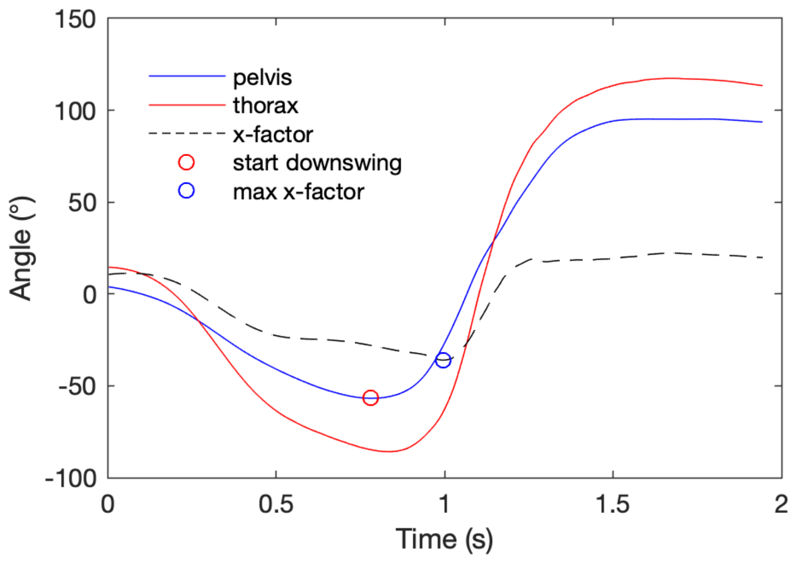

Figure 3: X-Factor, pelvis and thorax angular displacement plotted versus time. Minimum X-Factor and pelvis angular displacement are highlighted. Check that time separation between minimum values of 0.216 s appears correct.

Sharing your code

Let's return to the idea of commenting our code to improve readability – improving your code's readability is not just for you. MATLAB offers two useful features for sharing your processing and analysis with supervisors or collaborators: 1) publishing to a formatted file and 2) a live script.

Publishing your m-file

Open the m-file 'xf_script.m' located in the code folder. The script contains essentially what's been covered in this document so far, with a few minor additions. Notice the placing of %% signs to indicate sections, and a few helpful text descriptions of the code. In the **Editor** window, click on the **Publish** tab and then the drop down box below the **Publish** button. Choose **Edit Publishing Options...**. Here you can choose several options for publishing your m-file; select either a Word file or pdf as the **Output File Format**, and then click the **Publish** button. The result is a formatted documented that's formatted your comments as regular text, your sections as sections (the first one is the title) and even included a Table of Contents if you didn't suppress it in the publishing options. MATLAB also offers assistance for creating formatted text; when in the **Publish** tab of the **Editor** you will see some buttons that help you create sections, equations, bold text, etc.

Publishing your script as a document this way allows you to build a report in the Editor that shows your results. Your supervisor may not use MATLAB and, therefore, might not have MATLAB installed to check what you're doing. The published script allows him or her to read detailed descriptions of your code and justifications and interpretations of your analysis with all the code and output shown. Notice how I've left out the semicolon on line 52 to output the value of the *stretch* variable to the document.

Live script

Open the 'xf_script_live.mlx' file in the code folder to view the Live Script version of this script. MATLAB live scripts are files that contain your code, show your figures and formatted text. Live scripts are a great way of sharing your code because it gives the collaborator a look at the code behind the output, but is much more readable than a simple m-file.

To produce a live script click **Save as** in the in the **Editor** tab and select **Live Script** with the .mlx extension. Similar to publishing your m-file in the **Live Editor** tab there are several useful tools for formatting your document. While a Live Script is useful for explaining your code and showing your process, it does rely on your supervisor or collaborator using MATLAB. If they do not use MATLAB, you may choose to publish your script to .docx, or .pdf file instead.