MATLAB – Lab 4
File I/O and Plots

Download the files from the relevant section of the VLE, place them in your working directory.

Exercise 4.1 – Reading and writing data files.

In this first exercise you'll work on an m-file that will read data from a .csv file and then save a section of that data to a new .csv file. The data comes from a power meter attached to a generator that is powered by an undershot waterwheel (data from a second year design and manufacture 2 assignment). The data is captured at 100Hz and gives the power in mW over a short period of time. We'll look at the power data for two different waterwheels. The main difference between the two waterwheels are the gearing ratio between the waterwheel and the generator.

1. Ensure the read_write.m, waterwheel_data_1.csv and waterwheel_data_2.csv files from the VLE are in your working directory and remember to set the current directory in the MATLAB window to the same directory.

   a. Open the m-file in Editor and read through it, there are sections of code missing that you will need to complete.

   b. Open the waterwheel_data_1.csv file in Excel such that you are aware of what the file contains. A quick way to do this is to right click on the file in the Current Directory window and select Open Outside MATLAB. You'll note there is a header row containing text describing the variable in each column.

   c. In read_write.m alter the csvread function on line 15 such that it will not read the first row of waterwheel_data_1.csv (remember that the csvread function uses zero indexing).

   d. Save the file and run it. Check the workspace. There should now be a new variable named data. Is the size correct?

2. Now say we are only interested in the power data. We have two options, we can either read in just the second column from the .csv file or we can read the whole data set and then access the column data from the entire data array. We'll implement both options.

    a. Given that we have already read in the whole data, let's use our indexing method we used in Lab 1 to set a new variable power_1. Remember that to access values in a matrix we can use:

    ```
    new_matrix = old_matrix(r,c);
    ```
    (QG 2.2)

    where r and c define the rows and columns we'd like in new_matrix. (See Quick guide 2.2) Remember we can replace r or c with the colon operator, :, to get all the values in the rows or columns or use a:b to get all values between row or column a and b.

    b. Save and run the m-file and see that a new variable has appeared in the Workspace that contains just a column vector called power_1.

    c. Now we want to read the power data from the second data file using additional range arguments in the csvread function (QG 11.2). The issue here is that without reading the file we don't know the full length of the data so we can't add a value for R2. We can look at the data in Excel to see how many rows there are but the difference in indexing makes this easy to get wrong and you can't automate it.

    Another option is to use the size function (QG 3) and read the whole file within that function, this time reading from waterwheel_data_2.csv e.g.

    ```
    s = size(csvread('file_name.csv', R, C));
    ```

    This will give us the number of rows and columns in the data without adding the whole data set to our Workspace. There is no way of finding the size of the data without reading the file. While it takes time to read the file twice, we are not storing the whole data set in memory using this method. We can use the csvread function in the following format:

    ```
    power_2 = csvread('file_name.csv', R, C, [R1, C1, R2, C2]);
    ```

    where R and C are the row and column from which you'd like to start reading and R1 and C1 are the top left index of what you'd like to read and R2 and C2 define the bottom right index. If we want to read a single column, C, C1 and C2 will be the same value. R and R1 will also be the same. We can then replace the variable R2 with the value of s(1) (the number of rows in the data) in the second csvread function.
    Try implementing the two above lines of code in the correct order.

    d. Save the file and run, you should now have 4 variables in the Workspace, two of which contain the powerdata sets from waterwheel_data_1.csv and waterwheel_data_2.csv.

3. Now save one set of the power data sets to a new .csv file using csvwrite (QG 11.2).

    a. Still working in read_write.csv record the data in power_1 to a file called power_data.csv. This can be done using the csvwrite function, e.g.

    ```
    csvwrite('file_name.csv', M, R, C)
    ```

    where M is the matrix of data to write, R is the row and C the column index of where to start writing the file, in this case 0 and 0.

    b. Implement the csvwrite command, save and run read_write.m and check that a new power_data.csv file has appeared in your current directory and contains the correct data.

End of exercise 4.1

Exercise 4.2 – Visualising your data

In this exercise we will continue working with the code we produced in exercise 4.1 but will add functionality to produce a 2D plot of the data to compare the two sets. By plotting the data it makes it easier to determine the difference between the two waterwheels. We'll then look at producing a surface plot of some further data.

1. Open read_plot.m in the editor, this is how your read_write.m file should have looked at the end of Exercise 4.1. We will now add the functionality to plot the data you've read from file.

    a. The program so far will read in the two sets of power data. We now want to present the data on the same 2D plot. One thing you may notice is that the data sets are not the same length. This can be verified in the Command Window by typing `length(power_1)` and `length(power_2)`. We can plot 2 data sets of different size using the following:

    ```
    plot(X1, Y1, X2, Y2)
    ```

    providing X1 and Y1 are the same length and X2 and Y2 are.

    b. So far we have the data for Y1 and Y2, we need the data for X1 and X2 which in this case will be time. If we know the sample rate is 100Hz (0.01 sec at dt), we can create two vectors for time_1 and time_2 that are of the same length as power_1 and power_2 respectively. We can do this by creating a vector using `time_1=start:inc:end` where start is your first value (in this case 0), increment is the sampling time dt (already defined in the code) and end is the total duration of the data set in seconds. (QG 1.2) To get end, you can calculate using the following:

    ```
    total_time_1 = (length(power_1)-1)*dt;
    ```

    Implement this for time_1 and time_2 in the code.

    If you save and run it at this stage, you can check the Workspace to ensure the length of time_1 is the same as power_1 and same for the second set. Note for the plot function, it does not matter is X and Y are column or row vectors, however length(X) and length(Y) must match.

    c. Now we can use the plot function as presented above to plot the two sets of data. We also would like to label the axes, this is done using `xlabel` and `ylabel` functions. Use `help xlabel` to determine the correct syntax. (QG 10.1) As we are presenting two data sets, we require a legend. To do this use the `legend` function. Again, type `help legend` in the Command Window for the correct syntax and implement it in code. You can also add a title using the `title` function.

    d. Save and run the program. Does the plot make the data more readable, what can you determine from the plot regarding the power output of waterwheel 1 and waterwheel 2?

    e. It may be easier to see the data side by side. To do this we want to add a new figure which can be done using the command `figure;` We can then use the subplot function which is of the format

    ```
    subplot(m,n,p)
    ```

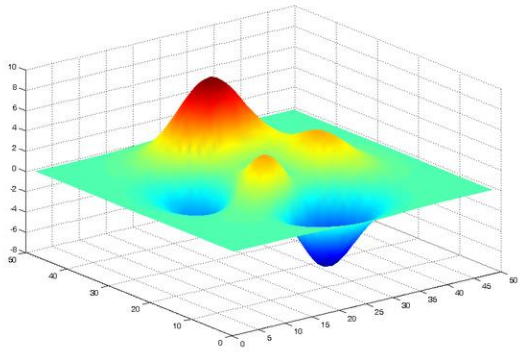
    where m is the number of rows of plots you'd like, n the number of columns of plots and p the current plot you're using.
    In our case we want to plot the two data sets side by side, so one row and two columns. To set up the first plot we can use:

    ```
    subplot(1,2,1)
    plot(time_1, power_1)
    ```

    While this is the current plot, you'll need to add labels and a title but not a legend as there is only one data set. Then go to the second plot using subplot(1,2,2) and then plot the second set of data and correctly label the plot. Save and run the program. Notice the axes have autoscaled. You can specify the axes range using axis([X1, X2, Y1, Y2]) where X1 is the minimum X value and X2 the maximum, same for Y1 and Y2 on the Y-axis.

f. Challenge: Calculate the mean power of each waterwheel to determine which one has the highest average power over the period measured. Display which was the best design and give the average power to the command window using the `disp` function.

Area of Interest



2. We are now going to produce a 3D plot, in this case a surface plot. In order to produce a surface plot we need three bits of data, a vector of x values, a vector of y values and a matrix of z values. (QG 10.2) The matrix has the same number of columns as values in x and the same number of rows as values in y. An example is shown below:

	x vector		
	0.00	1.00	2.00
y vector			
0.00	0.197	0.197	0.201
0.50	0.189	0.194	0.183
1.00	0.147	0.172	0.251
1.50	0.135	0.368	1.292

z matrix

We are going to use the area_of_interest.txt which was on the VLE. area_of_interest.txt contains measured forces across a surface. The samples are all 5mm apart.

a. Open a new script and save it as surface_plot.m. Add comments at the start of the file including the filename and a description of what we will be doing, mainly creating a surface plot from some data. Also clear the command window and the workspace. Now create two variables *dx* and *dy* and set both to 0.005.

b. Now we need to load in the data from the text file. The values in the file are separated by a tab. We need to use `dlmread`, passing two input arguments, firstly the filename as a string, and secondly a string containing the delimiter '\t'. Assign it to a variable *force*.

c. We have our matrix *force*, in order to use the surface plot we need the two vectors, one will be the x position and one will be the y position. We need to create these ourselves. We know that the x position vector should be the same length as the number of columns in *force*, starting at zero with each value 5mm apart. Likewise the *y* position vector should be the same length as the number of rows in *force*, starting at zero and incrementing by 5mm. Firstly use the `size` function to find the dimensions of *force* and set to a variable *dimensions*. You'll now need to create the vectors, there are a few ways of doing this:

```
max_x = (dimensions(2)-1)*dx;
x = 0:dx:max_x;
```

or

```
x = dx*(0:dimensions(2)-1);
```

Create vectors for both *x* and *y* positions.

d. We are now ready to use the surf function to create the surface plot (QG 10.2). We pass three input arguments, first the x vector, then the y vector and finally the *force* matrix: `surf(x,y,force)` . We are going to produce a shaded plot without grid lines, to do this we use the `shading interp` command. Finally make sure you label the axes. Now save your script and run the program. Hopefully you should see a surface plot of the data.

e. Have look and trying some of the other 3D plots like mesh, contour and pcolor with data. (See Quick Guide 10.2)

End of exercise 4.2