Download the raw_acceleration.csv from the relevant section of the VLE, it contains the data needed for today's lab. Place the file in your working directory.

Exercise 6 – Writing your own data filter from scratch.

For this lab all you will need is a data file containing some collected accelerometer data. The data comes from an accelerometer mounted on an Enforcer. An enforcer is the tool the police use to break down a door when entering a property. It is essentially a hollow steel tube filled with concrete with a steel end and two handles. They typically weigh around 16kg and can deliver 3 tonnes of impact force. The file raw_acceleration.csv contains two columns of data, the first is the sample time in seconds and the second is the acceleration of the enforcer measured in m/s^2 as it is accelerated toward a door and finishes just before the impact.
The purpose of the program you will write is to apply a moving average filter to the data and then display the filtered and unfiltered data together. To do this we will break the program down into tasks:

1. Read noisy_data.csv from file.
2. Plot the unfiltered data against time on a correctly labelled graph.
3. Create a user-defined function to apply a moving average filter with a filter length, n, of 15 samples.
4. Plot the filtered data on the same figure as the unfiltered data, correctly labeling the graph and adding a legend.

To complete this exercise follow these steps:

1. Create a new m-file and save it as enforcer.m in your working directory. Ensure that raw_acceleration.csv is also in your current directory in MATLAB.
    a. Place comments at the start of the m-file with the name of the m-file and a line or two describing what the program will do.

    b. Insert the commands to clear the Workspace and Command Window.

    c. Use the csvread function to read in the data from the file raw_accleration.csv. As we want to read the whole file, the range arguments can be omitted. Also remember that there is some header information in the .csv file so you shouldn't read the first row of data. (See QG 11.2) The code should be in the format:

```
data = csvread('file_name.csv', R, C)
```

    d. Now we'd like to separate out the 2 columns of data into two column vectors, one for `time` and one for `unfiltered_data`. To do this we must index all the values in the rows and the particular column to get the new column vector (See QG 2.2). It will be in the format:

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

where r and c define the rows and columns we'd like in new_matrix.
Remember: we can use 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.

e. If you now save and run the program. You should find that you have the three variables in the Workspace, *data, time* and *unfiltered_data*, the last two of which should be column vectors.

2. We now want to plot this data. As time and unfiltered data are the same size vectors, this will be straight forward.
   a. Use the plot function to display the data, it will be of the format `plot(x,y)` where x and y are replaced by your variable names. (QG 10.1)

   b. Label the x and y axis. Don't forget to add units to the axis labels, in this case seconds and m/s^2.

   c. Save and run your program and you should see the data plotted on a new figure. Check labels are correct and that the data appears to be plotted correctly.

3. You should see that the data you have just plotted has some noise in it. To remove this we will look at implementing a moving average filter. As we have seen there are two ways to implement this filter, the first is using:

$$xf_t = \frac{1}{n}\sum_{i=t-n+1}^{t} x_i$$

The second is using:

$$xf_t = xf_{t-1} + \frac{1}{n}(x_t - x_{t-n})$$

Where n is the filter length, t is the current index, xf is the filtered data and x the unfiltered data. Either will work although the implementation will be different. We will use the second of these representations. The challenge in this exercise will be to figure out how to program this filter.

   a. First we need to create a function, open a new m-file in Editor. Define the function in the first line of code: `function filtered_data = moving_average_filter(unfiltered_data)` Add some comments regarding what the function will do below the function definition and then save the file as moving_average_filter.m. Note the function will have one input argument and one output argument.

   b. Create a variable n for the number of values used in the moving average filter, in this case assign the value 15 to n.

   c. We now need to undertake the filtering of data. To do this we want to use a for loop as we need to filter each sample in the data. (QG 9.) We can use a for loop of the format:

```
for t = 1:???
    [code you want to implement]
end
```

   where ??? is equal to the length of the data we are filtering (think about how we can get this value).

   d. Something you have to think about is what will happen if we are trying to index a previous value of the filtered data (xf_{t-1}) that doesn't exist; when t = 1, xf(0) doesn't exist. To get round this we can use an if statement to check if the index, t is equal to 1, and if so set filtered_data(1) to unfiltered_data(1). Otherwise you can use the formula.

   e. In addition when a value of x_{t-n} doesn't exist at an index, e.g. when t is 5, the value of x_{t-n} will be x(-10) and as such an error will occur. We can guard against this using an *if* logic structure:

```
if xn < 1
      xn = 1;
end
```

where `xn = t-n`; used to index a previous value of x.

f. You will now have to correctly write the function code that will implement the filter, the code should have the following format:

```
for t = 1:???
      [some code]
      if [some logic]
        filtered_data(1) = unfiltered_data(1);
      else
          if xn < 1
                xn = 1;
          end
      filtered_data(t) = [some code]
      end
end
```

This is an example structure, it is up to you as to how you code the filter. You can check for syntax errors using the message indicator in editor as covered last week and use debugging tools covered also.

g. Save your function m-file. Now we need to call the function from your program. In enforcer.m, place a line of code calling the function after the plot function. The format of this code is already defined in the function file you have just created.

h. Save enforcer.m and run. Check to see if any errors have occurred. If not, you may have correctly coded your function. Check you have a new variable filtered_data in the Workspace that is the same size as unfiltered_data. You may want to check they contain different values at this stage to give you an idea if the filter has worked. You may not know for sure until after the next step of plotting the data.

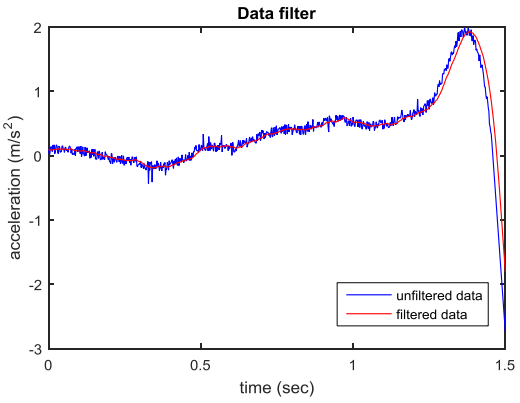
4. We will now plot the second lot of filtered data against the original unfiltered data. To do this we can either create a new plot with both data sets or we can just add the data to the current plot by using the hold function (QG 10.1), we'll do the former.

   a. Cut the earlier plot, xlabel and ylabel functions from enforcer.m and paste at the end of the file.

   b. Alter the plot function again to plot unfiltered_data against time and filtered_data against time on the same plot. Use the string input argument to change the appearance of this second plot, i.e. 'r' will change the colour of the line to red. Use `help plot` in the Command Window for the correct syntax.

   c. After the plot use the ***legend*** function to add a legend to distinguish the two plots.

   d. Save and run your program. Are you getting two plots, one of which is a smoother version of the unfiltered data? It should look like the figure below:



If not, the function you wrote may not be working correctly. You can place breakpoint at the point in enforcer.m where your function is called and then run your program. It will cause the program stop at that point in the code, signified by the green arrow. Now use the step into button on the debugging toolbar to step into your moving_average_filter function. You can then use the step button to manually step through your code to see what is happening and if your filter is working correctly. See if you can use the debugger to help fix your code.

Challenges

5. If you finish this successfully, try and add the ability for the user to select the strength of the moving average, allowing for a variation in the length of the moving average, n. See how changing the value of n effects the acceleration data.

6. Finally as a real challenge, work out that the peak velocity of the enforcer is just before it hits the door. To do this you will need to undertake some numerical integration, you'll need to make use of a for loop to do this. You can use the equation below where v is velocity, a is the filtered acceleration and dt is the time step, in this case 0.002. Plot the calculated velocity vector against time.

$$v_t = v_{t-1} + \left(\frac{a_t + a_{t-1}}{2}\right) dt$$

End of exercise 6