# Self Study 2 – Performance Marking


## Introduction

In this week's lab you looked at logical and relational operators and also `if/elseif/else` structures. You are now going to put to the test what you've learned as part of this week's self-study. Once again you will be submitting your work to ScriptCheck and once again, you will be writing your code in the editor window.

## The Scenario

In Lab session 1 you looked at the buggy acceleration data from the 2nd year Design and Manufacture module. There were 16 buggies in total and you were given their vertical acceleration data in $m/s^2$ as they completed the course. As we do this every year, I'd like you to create a program that will mark the suspension performance based on the measured RMS (root mean square) value of acceleration. As a starting point, I'd like you to provide me with the best buggy in terms of RMS value (lower the better) and their respective marks. The marking scheme is shown in table 1.

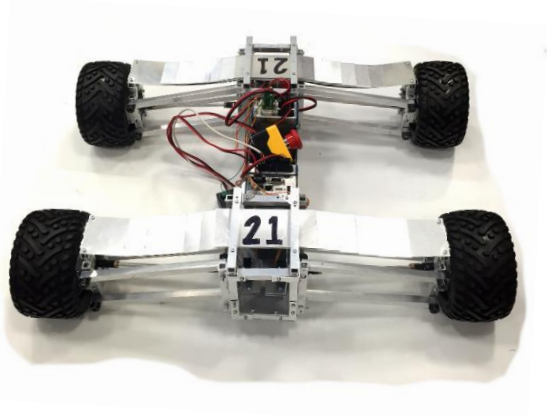


*Table 1-Suspension performance marking scheme*

| RMS acceleration over simulated course ($m/s^2$) | Mark awarded |
|---|---|
| >=2.51 | 0 |
| 2.01 to 2.50 | 1 |
| 1.51 to 2.00 | 2 |
| 1.01 to 1.50 | 3 |
| 0.51 to 1.00 | 4 |
| <= 0.50 | 5 |

To solve this, you're going to use some of the concepts we covered this week and last week. Follow the steps below to complete the task. Note that the instructions will require you to think about what you've learned and apply it. The result of the task is to determine which tutors buggy was the best and what marks they received. All tasks should be completed programmatically in your .m file. Remember, when example script is provided in the sheet, anything in red needs to be overwritten with the correct code. Also remember you can save and run your script in MATLAB after completing each step to see how you are getting on.

## Instructions

1. Create a program .m file called performance_marking.m. Don't forget to add a comment at the start of the script containing the file name and some useful comments describing what the file will do, i.e. marks buggy performance based on acceleration data. The different variable names used are given below. You are welcome to copy them into your code when needed to avoid typos:

```
buggy_rms
best_rms
best_rms_index
mark_best
winning_team
answer
```

2. Programmatically clear the workspace and command window. (QG 0)

3. In your program load the `self_study_2.mat` file into the workspace (QG 11.1). The .mat file contains a matrix, `acceleration_data`, containing the vertical acceleration data of each buggy in $m/s^2$, 16 columns give the data for each of the 16 buggies. The .mat file also contains a 1x16 cell array called `tutors` that contains the tutor names in string format in the same order that the acceleration data is provided. Think of a cell array as a row or stack of boxes, where each cell is an individual box. The boxes can each contain any data type, be it a string, a matrix or a scalar numeric. They are similar to clusters in LabVIEW.
If you index a cell array using standard parentheses (), you can this of it as picking up the box so you end up with a cell. If you use curly brackets or braces {} you are getting the contents of that box so end up with the value inside the cell. We'll use them a few times during the course so to understand more check: https://uk.mathworks.com/help/matlab/cell-arrays.html

4. Now you have the `acceleration_data` loaded from the `.mat` files, find the RMS value of each column. The RMS value is found by squaring each value in a vector and then finding the mean of these values before finding the square root of that mean. You'll need to use the functions `sqrt` and `mean`. Use help for information on how these functions work, e.g. `help sqrt`. Note that mean will find the mean of the data in each column when applied to a matrix. Also remember the use of the dot operator. Assign to the variable `buggy_rms`.

5. Now you should have the RMS values for all 16 buggies in a row vector. You need to find the buggy with the best RMS value. You should already know and have used the function that will do this (look at QG 3). Remember to add the additional output arguments using square brackets and a comma separating variable names. This will allow you to get the index as well as the RMS value. Assign these to variables `best_rms` and `best_rms_index`. (QG 3)

6. Now use an `if/elseif/else` structure to determine the mark for the buggy with the best RMS and assign it to the variable `mark_best`, (QG 8) here's some lines to start you off:

```
if best_rms >= 2.51
      mark_best = 0;
elseif conditional statement
      Additional logic statements and lines of code go here.....
      .....
else
      Line of code here
end
```

This should result in a value for `mark_best` being assigned.

7. You now want to get a string containing the winning buggy's tutors name and assign it to a variable `winning_team`. Remember `tutors` is a defined as a cell array containing a row vector of cells. Each cell contains a string of the various tutor's name. The best way to get `winning_team` is by indexing the contents of the correct cell in the cell array `tutors` using the index of the desired buggy as shown below. Note the use of braces not brackets.

```
winning_team=tutors{index};
```

8. We would normally use `disp` to find out what the results of our analysis is but as you will be using ScriptCheck you can't use `disp` as ScriptCheck doesn't read the command line. Instead we will use `sprintf` to write to a variable called `answer` which ScriptCheck will then read. As you won't be familiar with how to use `sprintf` the format is shown below (type help sprint in the command window (QG 12.4). Cut and paste the script below into your code and replace a and b with the correct variables. Note the three dots allow you to continue a line of code across multiple lines in the editor.

```
answer=...
sprintf(['%s buggy had the lowest acceleration and gets %i marks.']...
, a, b)
```

where `a` is a string containing the tutor's name you got in step 7 and `b` is the mark they received. Don't include a semi-colon at the end of this command to see the result in the command window when you run the code.

9. Make sure you save your .m file. Now run it and the correct tutor's buggy and their performance points should appear in the command window. Once happy, submit your code in scriptcheck to see if you are correct. http://scriptcheck.leeds.ac.uk

If you are stuck I will go through a similar exercise on Tuesday in the example class.