Testing Environment:

Testing was carried out on a Windows machine, since our module is compatible with windows at the moment. We used Microsoft Excel as a tool to manually compute data. The application within our module is a .exe executable file. This can be executed on the command line as well as doubling clicking the application. Tests were performed using both methods, whilst running Windows 7 32-bit and 8.1 6.4-bit as OS's when carrying out these tests.

The limitation was that many of these tests were carried out for a small number of entries, since it would be very time consuming to verify output manually for larger chunks of data. In taking in all different types of records, we assume that if it works for a small input then it should then work for a large input size.

Overview Of Test Data:

Using 10 trade entries, we tested our module by varying the value of n and th (“window size for simple moving average” and “threshold value” respectively). There were 3 main test cases for which were analyzed:

1. Setting parameters n = 3 and th = 0.0005
2. Changing n = 6 and keeping th = 0.0005
3. Changing th = 0.0008 and keeping n = 3

These test cases were repeated with a standardised sample data file that was 100227 records in size.

We must note that the default values for the parameters are set as n = 3 and th = 0.00005. These default values were used upon double-clicking.

We then carried out testing using other teams’ modules. We used the same parameter and input file and generated the output file. This helped us compare our results with theirs. To sum it up, we tested our module by varying every parameter one by one, while keeping the others constant.

Testing Process:

First, we chose 10 trade entries from the original input file, and loaded it onto an excel spreadsheet. Then we manually entered the MSM strategy formulas one by one, to compute whether a buy or a sell signal was to be generated. The R(t) values were calculated in a extra column using an excel formula. Next the SMA values were calculated in a new column using the R(t) column and an excel formula. The SMA column was then used to calculate the TSV(t) value in another column also using an excel formula. Finally we manually compared the TSV(t) value to our threshold to compute whether to buy or sell. We then saved this information into an output file. We used this process for each of the input data, firstly for 10 inputs, parameters n = 3, threshold = 0.0005, and concurrently ran it using our module. We did this for all three test cases.

Once that was done, we had 3 output files generated by running the input on the module and 3 output files, which were manually computed. Each pair of files were compared by running th Unix command 'diff' on each corresponding pair. From this, we concluded that the output generated by the module matched the manually computed results.

We then used other teams’ modules to do the same, and compared the output. Furthermore, we compared our application for speed performance. We found that our module was taking longer than expected.

One of the main attributes to the time it takes is due to the way Ruby is run on Windows in particular. Ruby-generated applications, as a result, do not work well with Windows OS systems.

We can further deduce that Ruby, itself, is not comparable in terms of time performance to many other common languages and, in fact, proves itself to be slower than all other popular languages, namely Java which was primarily used.

COMPARISONS:

* Group 4 produced results in ~90-100ms, however results did not vary with parameter changes. Consistently produced all TRADE transactions from original file. Same number of trades found
* Group 3 found same number of trades. Produced 1515ms in log file. 1890 ENTER records produced
* Group 7: 1698 ENTERS produced in 357ms, also 16485 trades