INITIAL TESTING DOCUMENTATION, SENG3011 – The G's

Authors: Saishav Agarwal, Yuwei Chen, Rashim Rahman, Hariharen Veerakumar

OVERALL SOFTWARE ARCHITECTURE

The architectures for Sections 1 and 2 are described with Figures 1 and 2 respectively.

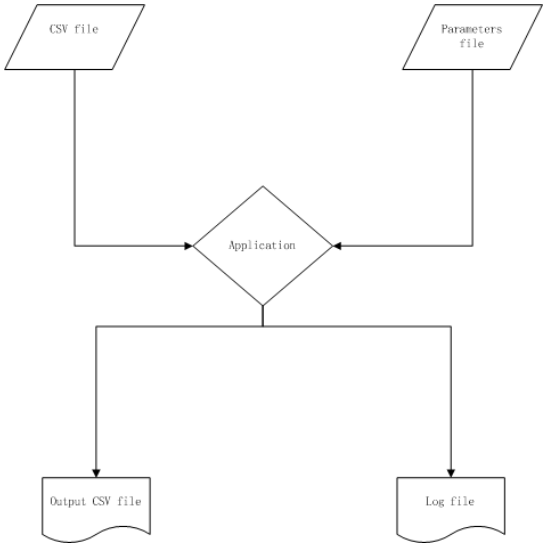

Figure 1: Visual description of how the application interacts with its inputs and outputs

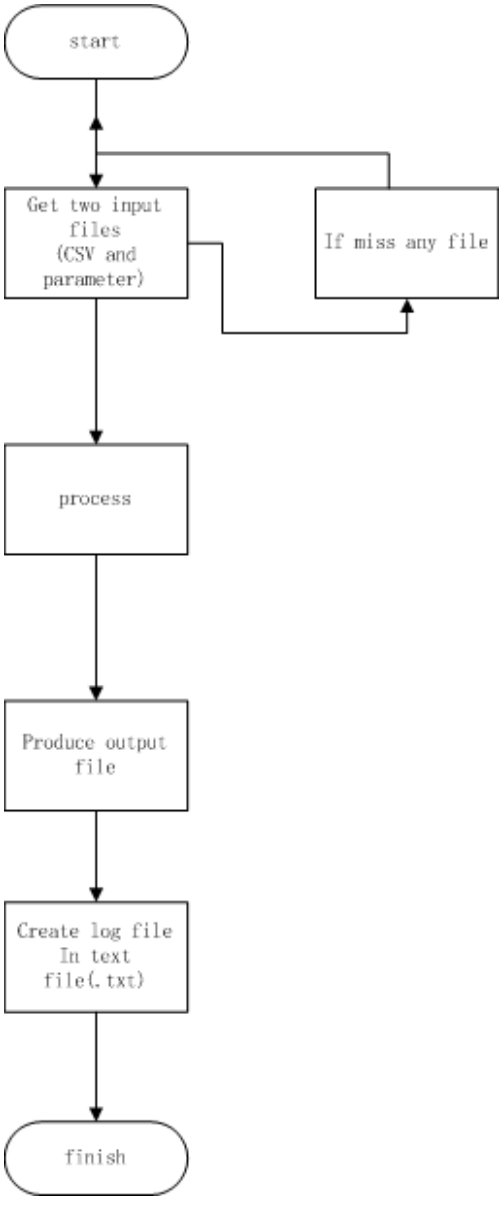

Figure 2: the lifecycle of the application's usage from start of use to success

The architecture was designed in a manner such that it was in line with the methodology required for applying a financial strategy. The inputs include:

- a parameters file
- input SIRCA CSV file

The parameters file is included to allow a more versatile use of the module. Although it is recommended not to edit any files, it can be done at their own risk – these errors cannot be controlled by anyone but the user. Nonetheless, the application allows the use of flags to change the parameters safely.

The log file, despite being a requirement, proved to be useful for analysis and comparison of the module.
The lifecycle of the application within the module is designed to account for:

- Catching potential exceptions
- Ensure correctness and safety through the lifecycle
- Designed to achieve the result using minimal steps

The process is defined as an algorithm, known as a 'financial strategy', and is currently limited. However, the order in which the files are read and produced are imperative and, hence, are placed in their respective positions accordingly in the lifecycle.

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.

DETAILS REGARDING TEST DATA

The following are the testing files included with this document :

Input file with 10 trade entries

trades10input.csv

Parameters	Manual Testing process files	Output Files(Manual Testing)	Output Files (Module Generated)
n = 3 th = 0.0005	trades10testingprocess.csv	trades10output.csv	trades10AutoOut.csv
n = 6 th = 0.0005	trades10testingprocessN.csv	trades10outputN.csv	trades10AutoOutN.csv
n = 3 th = 0.0008	trades10testingprocessTH.csv	trades10outputTH.csv	trades10AutoOutTH.csv

Performance Testing against 3 other modules

Our log file	MSM Module A	MSM Module B	MSM Module C
log.txt	logTestA.txt	logTestB	logTestC.txt

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 an 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. We then repeated for the larger set of data.

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 was compared by running the 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.

COMPARISONS:

- A: found same number of trades. Produced 1515ms in log file. 1890 ENTER records produced
- B: 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
- C: 1698 ENTERS produced in 357ms, also 16485 trades

CONCLUSIONS

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.

In regards to results, there appears to be minor differences in results between the different modules. Given the test cases provided, we are yet to determine where all cases have been accounted for. This will be done via more unit tests.