Foundations of Computation – ECE358

Design Documentation

Final Version

Tejas Mehta: 997486581
Neil Newman: 997382443
Stefan Hadjis: 997701170

Please Note:

The major source of information for how to use this software is given in README.txt. It explains the file interactions, inputs, and steps to use the software.

This document is a supplemental document explaining the algorithm in a bit more detail and summarizing the Action codes, but note that (due to careful planning), this document is very similar to the description of the project outlined in the original design document and the original plan for the algorithmic trading is very similar to the final algorithm implemented.

The software interactions described in the figure on the following page remained the same and the implementation of the PairsTrading algorithm (done using a 4-state Finite State Machine) remained the same as well. The only major changes in implementation were specific details, such as finding thresholds for when to trade etc., and also to fix bugs and improve the performance of the strategy. Please see comments in the code for specific implementation details.

However for the most part the project was completed as originally described.

Output File Information:

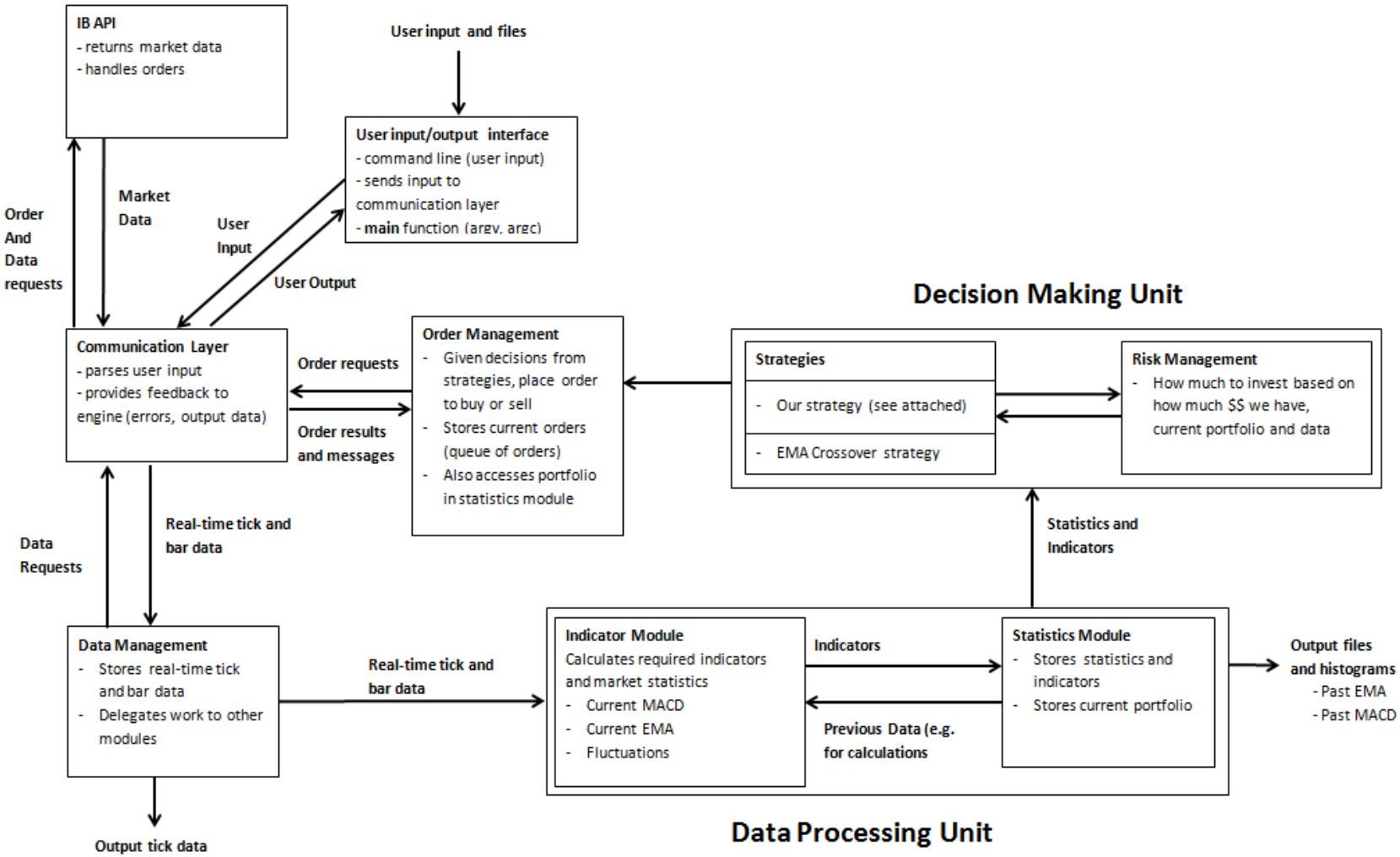
User output (e.g. prices, EMAs, MACD Histrogram, etc.) is presented in text files which are automatically updated and appear in the root directory. For example:

PEP_EMA_Time5s.txt

Represents the EMA of the Pepsi stock updated every 5 seconds.

During debugging certain debug messages were added to these files (to help us understand what was happening). We considered removing these but we kept them as an indication of what is happening in our code and that the code is in fact working as expected.

Design Flow Chart / Sequence Diagram



User Input (Action Codes and Files):

Upon starting the executable and connecting to the TWS (see steps in README.txt) the following inputs are available along with example input files in the **Examples** directory. We have also provided a simple summary of the methods called.

Code	Name	Action (performed by the Communication Layer following parse_code)
1000	Connect	1. Call OnConnect() in API to connect to Trader Workstation 2. Display to user error or output connected successfully
1001	Disconnect	1. Call OnDisconnect() to disconnect from Trader Workstation 2. Display to user error or output connected successfully
1010	Display Current Time	1. Call OnRequestCurrentTime() 2. Display to user error or output connected successfully
2000	Request Real-time tick data	1. Call OnReqMarketData() in API 2. Return to communication layer 3. Store data in the Data Management Module, which will continue to ask for that market data and update the output file
2001	Request real-time bar data	1. Call OnReqRealTimeBars() in API 2. Return to communication layer 3. Store data in the Data Management Module, which will continue to ask for that market data and update the output file
2400	Request EMA	1. Data management module calls OnReqEMA(), which every Bar Size seconds requests new market data through OnReqMarketData() and then sends the data to the Indicator Module to calculate EMA 2. After calculating, the indicator modules stores the EMA in the statistical module which outputs the file.
2401	Request MACD, signal and histogram	1. Data management module calls OnReqMACD(), which will in turn call two OnReqEMA() with the bar sizes and calculate the difference between the two EMA's once they are valid. 2. After calculating, the indicator modules stores the MACD in the statistical module which outputs the file.
3000	Cancel Realtime tick data	1. Call OnCancelMarketData() in API 2. Output success or failure to user
3001	Cancel Realtime bar data	1. Call OnCancelRealTimeBars() in API 2. Output success or failure to user
3400	Cancel EMA	1. Data management module calls OnCancelEMA(), which halts requests for new market data and then stops sending data to the Indicator Module
3401	Cancel MACD	1. Data management module calls OnCancelMACD(), which halts requests for new market data and then stops sending data to the Indicator Module

6000	Place an Order **Sample Input File:** order.txt	1. Call OnPlaceOrder() in API to buy or sell 2. Output success or failure to user
6900	Clear All Positions	1. Clear order queue datastructure in order management
8000	EMA Crossover (non-stop) **Sample Input File:** pair.txt **To Stop:** canemacrossover.txt	1. Data management module calls OnReqEMA() twice, with two different bar sizes, and for each bar size requests new market data through OnReqMarketData() and then sends the data to the Indicator Module to calculate EMA 2. The Indicator Module calculates the EMAs and stores in them in the statistics module 3. Statistics module sends the results to the strategies module and risk management module to calculate when orders should be placed based on crossover strategy, and continues until a stop signal is sent
8008	EMA Crossover (one order then stop) **Sample Input File:** emacstopwin.txt **To cancel:** canemacrossover-stopwin.txt	1. Data management module calls OnReqEMA() twice, with two different parameters d1 and d2, and for each time requests new market data through OnReqMarketData() and then sends the data to the Indicator Module to calculate EMAs. 2. The Indicator Module calculates the EMAs and stores in them in the statistics module 3. Statistics module sends the results to the strategies module and risk management module to calculate when orders should be placed based on crossover strategy, and stop after slow-d2 and fast+d1.
8888	Algorithmic Pairs Trading **Sample Input File:** pair.txt **To stop:** canpair.txt	See description below
9000	Stop Automatic Trading	1. Data Module stops requesting data and stops calculation in indicator module and strategy module
9999	Exit the program	1. Program (main function) terminates

Algorithmic Pairs Trading

Algorithmic pairs trading involves considering pairs of stocks which are expected to have a high correlation. Consider Coca Cola© and Pepsi© as an example – while their current prices may differ significantly one might expect that trends in their prices may potentially follow similar trends. Our algorithm aims to exploit this knowledge.

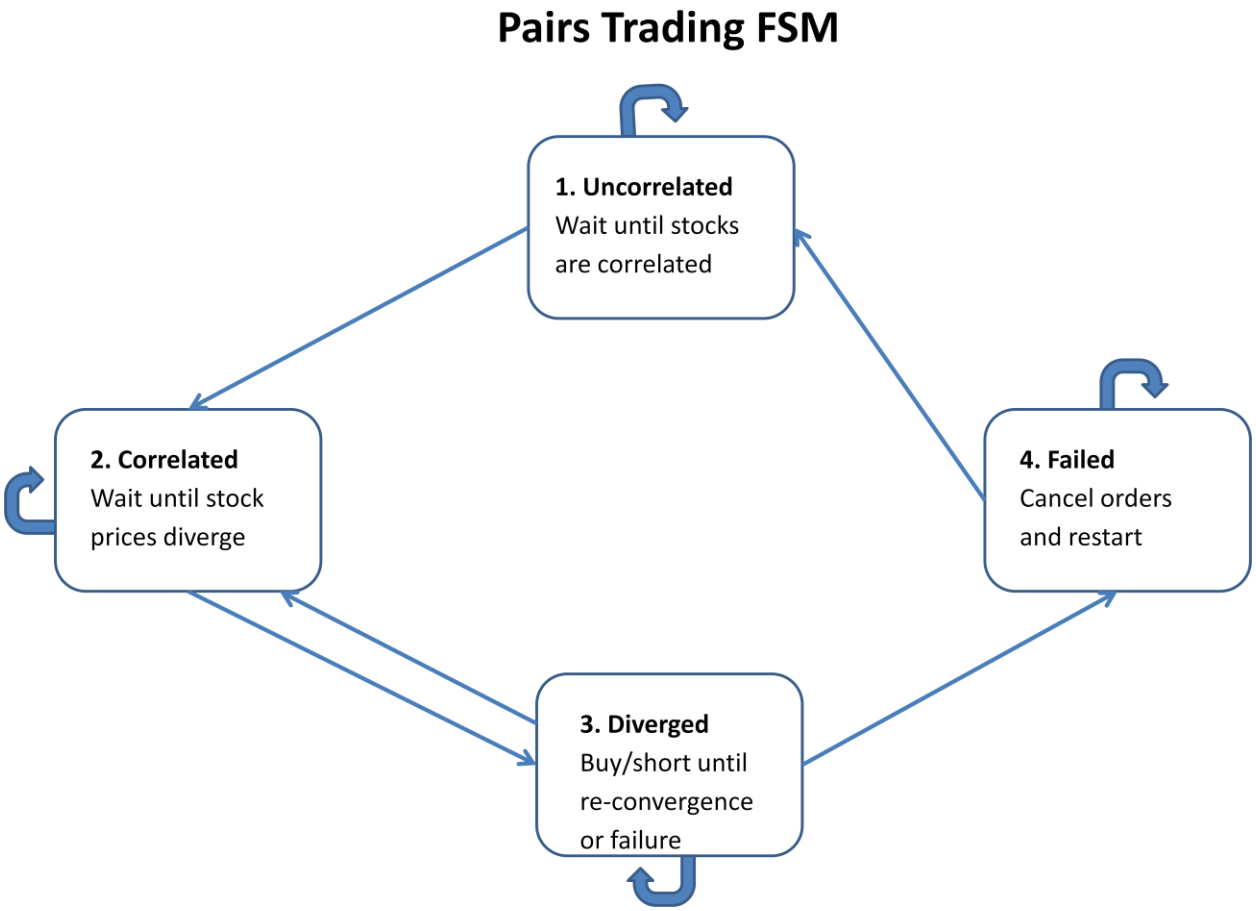
Once pairs are determined, the algorithm performs the following steps (described in detail below), which are summarized in the Finite State Machine below:

Step 1. Wait until pairs appear to be correlated

Step 2. Once pairs are determined to be correlated, wait for a divergence in performance (defined below)

Step 3. To exploit this divergence, **short** the stock which is declining and **buy** the stock which is rising.
Repeat step 3 until the market is again correlated, then go to step 2

Step 4. If the algorithm **fails** (defined below), restart from step 1



Step 0 – Find potentially correlated pairs

The algorithm requires knowledge of pairs which may be correlated. Wikipedia lists these examples, and the two included in our Examples test suite include:

- Coca-Cola (KO) and Pepsi (PEP)
- Wal-Mart (WMT) and Target Corporation (TGT)

Others exist, e.g. Altera and Xlilinx.

Step 1 – Determine if Pairs are Correlated

At the start of the algorithm, the initial price of each stock in the pair is recorded. Each time step thereafter the % deviation from this original price is calculated for each company. For example if Coca Cola is at 1.00 at time 0 and 1.04 at time 3, then its **%Change** at t=3 is 5%.

We define the pair (A,B) to be **correlated** if the absolute value of their price difference is below a threshold T1 for N consecutive time steps, i.e.

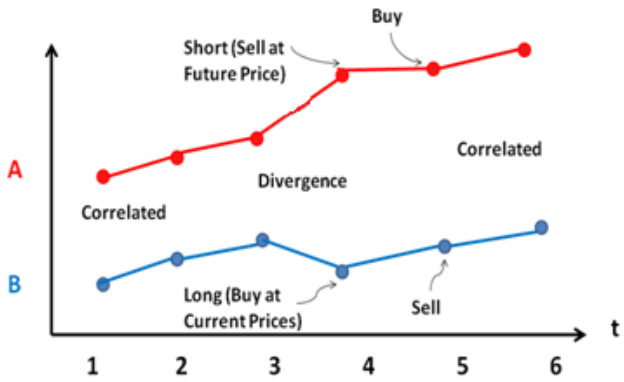
$$| \%Change_A[t] - \%Change_B[t] | < T1 \qquad \text{for} \quad t_1 \ldots t_2 \text{ where } t_2 - t_1 = N$$

Where T1 and N have been determined empirically through experimentation (see PairsTrading.h)

An alternative definition relies not on the prices of the stocks at each time, but rather their EMA. This is another parameter which we will experiment with.

Step 2 – Wait for Divergence

Once we have determined that the pairs are correlated, wait for them to diverge. See e.g. to the right: the stocks diverge at t=4.



Step 3 – Buy and Sell stock

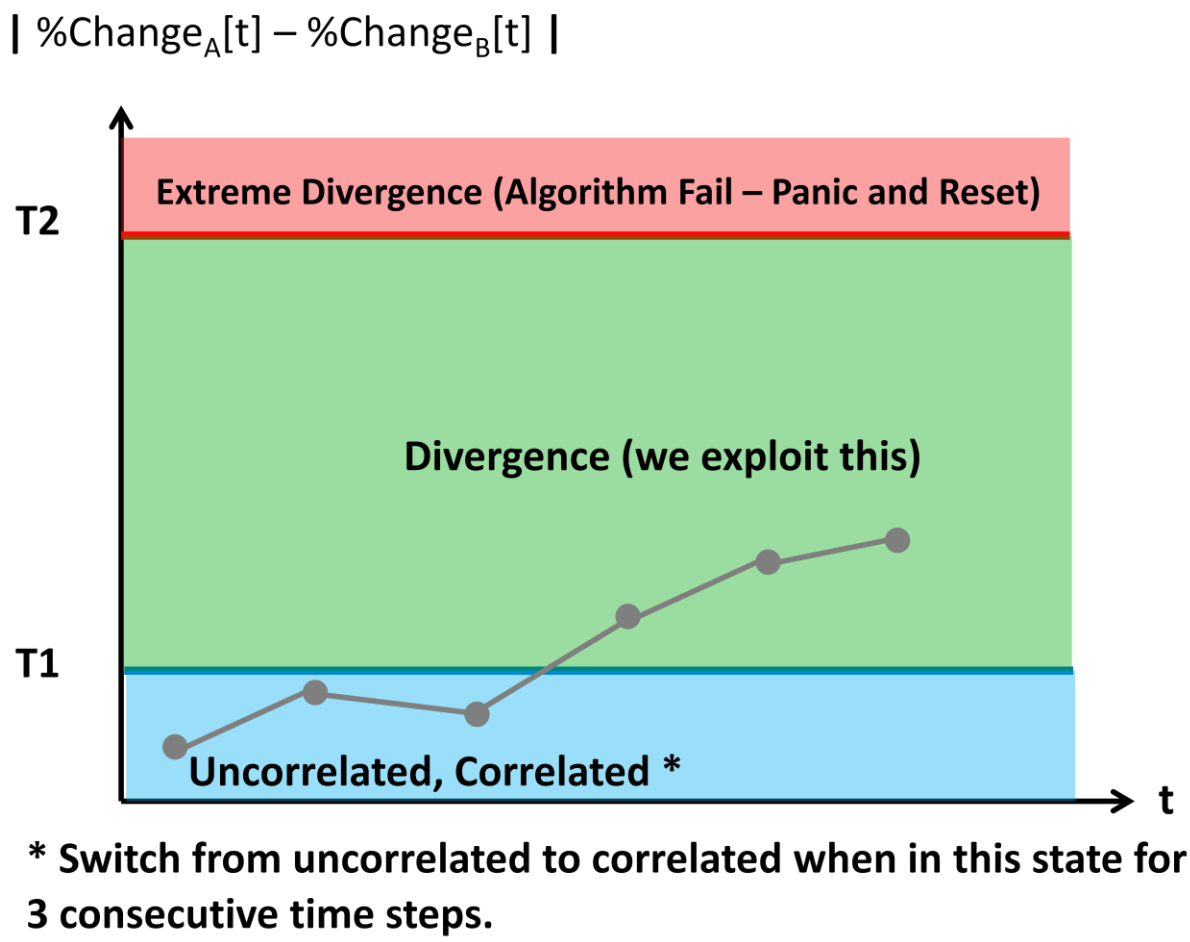
Conceptually, if we are confident that the stocks are correlated (step 1), we expect that in the future they will be correlated again. Therefore A should decrease, B should increase, or both. Therefore we will short A (the rising stock) as we expect it to fall, and buy B (the decreasing stock) as we expect it to rise.

Specific Details

The data that our algorithm considers is the absolute difference of %Change as a function of time, i.e.

$$| \%\text{Change}_A[t] - \%\text{Change}_B[t] | \text{ as a function of time.}$$

There are three regions in this plot which are of interest:



Below T1, the stocks are not yet diverged, i.e. either uncorrelated or correlated (step 1)

Between the range T1 and T2, we repeat step 3: At each time instant, we continue to buy the stock that is decreasing and short the stock that is increasing, until the difference falls below T1 (at which point we stop buying and selling and buy/sell to clear our current positions and return to parity) or goes above T2 (in which case we've failed, so we again buy/sell to undo our positions). The amount we invest at each time step is a function of how long we have been in the divergence region, increasing the longer there has been a divergence. This is determined by the **Risk Management** module, which is simply a function inside PairsTrading.cpp (see the source.)

Step 4 – Algorithm Failure (or intermediate step between State 2 and 3)

Above T2, the stocks have diverged and we immediately clear our positions sell all stock we have and return the first step.

Note: The above description was in the original design document. One minor edits was made to the FSM: in the original FSM presented above, State 4 undoes all orders (e.g. if we were buying a stock, sell it). This is in fact the same functionality which we want once our stocks re-converge (i.e. exit divergence but successfully as opposed to due to failure) therefore we also enter State 4 in when leaving State 3 to return to State 2, i.e. a slight modification to the above diagram makes us pass through state 4 for 1 step before returning to state 2. Hence State 4 should not be called FAIL but rather UNDO ORDERS.

Implementation

The Data Management module sends real-time data to the Indicator Module which calculated the difference above, based on real-time prices or the EMA of each stock. This is stored in the Statistics Module and used by the Strategies Module to determine which of the three ranges above we are in. Once in the divergence stage, the Strategies module would invoke the Risk Management module to determine how much to invest.

Additional Details

For examples of all input files, see Readme.txt and the Examples folder.

The Security Definitions in the order input file is as follows:

```
ORDER ID
TICK
ACTION = {"BUY", "SHORT", or "SELL"}
ACTION AMOUNT
"LMT"
LIMIT AMOUT
```

EMA and MACD Notes

In order to calculate the EMA, the indicator module allocates an instance of a subclass called EMA, with parameters bar size and number of periods. The EMA object dynamically allocates an array of size equal to number of periods and an integer for the current EMA value. Then the EMA object collects data from OnReqRealTimeBars() and stores it in the array. Once the array is filled the EMA value is calculated. From then on the algorithm uses the current price from OnReqRealTimeBars() and the old EMA value to calculate the present EMA value, and discards the array which held the old bar data.

$$EMA = Price(Today) * k + EMA(y) * (1 - k), where\ k = 2/(N + 1)$$

In order to calculate the MACD, the indicator module allocates two instances of EMA, with parameters bar size, given by the user, and number of periods – probably 12 and 26. Once the two EMA are valid the module continually subtracts the EMA 26 – EMA 12, and inputs this value into a new EMA with same bar size as above and a period equal to 9. Then once the MACD-9 line is valid (after nine time instances) the histogram is calculated and outputted to a file until a cancel request is made.

Risk Management

For EMA Crossover, see EMACrossover.cpp. The amount invested is a function of the angle of the crossover.