University of Pennsylvania
School of Engineering and Applied Science
Computer Information Sciences

CIS 400 Senior Design

Financial Modeling: A system to test High-Frequency Trading Strategies

Danish Munir danishm@seas.upenn.edu, (CIS)
Divya Krishnan divyak@sesas.upenn.edu, (ESE)

April 24, 2009

Advisors: Professor Abraham Wyner, Statistics Department, Wharton
Professor Insup Lee, Computer Information Sciences

Table of Contents

Abstract

In the past few years, advancements in technology have enabled traders to be able to carry out trades at ever faster rates. Starting from trades being put physically at the stock exchange, to then being put over the wire, the telephone and finally over the internet, today trades can be executed at a split second's notice through specialized systems. This rapid maturity of trading engines and the level sophistication of communications technology has resulted in what is known as "high-frequency trading" which represents the ability to place trades at speeds that are only feasible by automated computer-based systems.

This new era of trading brings with it its own set of challenges, and requires a new breed of strategies by investors in search for profits. However given the high volume of trades occurring at each second, the resulting volume of data from such trades is overwhelming and is extremely hard for many investors to analyze, and to test their strategies on. Thus it becomes hard to take a strategy that is born out of intuition and to put it through robust testing before actually taking the risk of investing money using such a strategy.

This project has two objectives: One to provide a platform that allows simulating trades at ultra-high frequencies and allows investors to back-test trading strategies using historical data. The other, is to prove the hypothesis that when trading at very high frequencies and at very small time intervals, the correlation between assets breaks down in a non-linear way, and to demonstrate a strategy that exploits this statistical property to make money.

Goals

In order to be able to properly evaluate our work, it is necessary to understand not only its goals, but the motivation that is driving these goals.

Since trading at high frequency is now common place, there is a swath of financial engineers who strive to find the best ways to squeeze out every ounce of return possible from such trading techniques. However, many such trading strategies are based on intuition and 'gut-feeling' and are not backed by statistical theory.

Our first goal was to explore a hypothesis developed by our advisor Professor Abraham Wyner, and to examine this hypothesis in a thorough manner, both through rigorous statistical analysis and through proving it using empirical data. This hypothesis which has been explored by other academics as well is that the essentially, the correlation between the price movements of two assets is driven mainly by 'market-moving news'. Any other transactions that occur which are not in response to news actually cause the correlation between two assets to break down. This hypothesis results in a statistical property of the correlation between two assets decreasing at smaller time intervals. Our first goal then was to prove or disprove through actual data whether this breakdown in correlation happens at smaller time intervals or not.

Leading out of this hypothesis, we developed a trading strategy based on equal-weighted portfolios, rebalanced at regular intervals, which aimed to capture the difference in returns resulting from the varying correlations over varying time scales. We talk more about the details of this strategy later, but our next goal was to build a system that can test out this strategy which required trading at high-frequency intervals.

Once we started building this system, we realized that our work was extremely modular and had high value outside of the specific goal that we had initially sought. We therefore decided to expand out horizon and build a system that can be used to test any kind of high-frequency trading strategy with very little modification in code.

Our tangible objectives then included:

1. Proving or disproving statistically, the hypothesis held by our advisor regarding breakdown of correlation over smaller time intervals
2. Building a generic system that can be used to test high-frequency trading strategies
3. Testing our system by employing the rebalancing strategy

Previous Work

Although there has been a large amount of work done in the area of building and optimizing trading systems, it is still hard to easily find a system that has goals similar to ours. This is primarily because in the past most systems have focused on day-trading and not really tackled automated, high-frequency trading. Additionally, the bulk of the work that has been done has been carried out by large Investment banks and Hedge Funds. This has been, for obvious reasons, mostly proprietary work, and is hence unavailable for deep analysis. Our system hopes to provide a new insight into high frequency trading by examining the use of a specific family of strategies that exploit varying co-variances between assets.

Low-Frequency Trading

Previous work on this area has focused mostly on analyzing the performance of such portfolio in low frequency trading. The given theory (Fernholz 2007[1]) has not been implemented in any known algorithm. In addition, there has only been in-depth study on low-frequency trading and the fundamental concept that govern this strategy. This has been well-documented and accepted in industry.

High-frequency trading relies on intra-day volatility and the inherent nature of the security rather than the surface price that is seen in low-frequency trades. Relatively little study

has been done to study the optimal frequency of intra-day trading and an algorithm that can predict profitable trades.

Pairs Trading

A further strategy that many industry experts employ is the arbitrage strategy of pairs trading. Two assets are identified to have a relationship that is outside of historical norms. It is typical to calculate what the stable numbers between the two assets are (i.e correlation, covariance, returns/price ratio) and proceed to trade when the markets perceive a deviation in the numbers. Unlike other trading strategies, we are not concerned with how each individual stock performs, but the relative performance of the stocks. Pairs trading also has many variations, the most common a dollar-neutral strategy, which is the method that has been found to be the most effective. In dollar-neutral strategy, the same amount of asset that is held in a long position is held in the asset that is shorted. We use a modified version of this strategy where instead of a pair of assets, a pair of portfolios are maintained.

Costs

A further factor that few studies have attempted to incorporate is the idea of transaction and trading costs. Performing an analysis of identifying pairs and placing trades according to trading rules is the first step in evaluating a strategy. To fully realize the actual and expected returns requires the inclusion of these realistic costs as they will take away from the gross returns calculated. As borrowing costs and trading costs are hard to model, many studies ignore them to make the experiment simple. This continues to remain a challenge for us as well since trading costs vary greatly depending on the person or entity carrying out the transactions. However we do attempt to give some intuition about the size of the transaction costs through a metric called ‘turnover ratio’ which is a good proxy for transaction costs.

Penn Exchange Simulator

One attempt at modeling strategies in automated trading has been at the University of Pennsylvania, through the development of the Penn Exchange Simulator (PXS). This is part of the Penn-Lehman Automated Trading Project headed by Michael Kearns at UPenn. The aim of this project is to provide a real-time simulation that mimics market conditions and allows students to write agents or bots that can trade based on real time data drawn from the ISLAND Electronic Crossing Network (ECN). While this project has some overlap with ours as far as the goals are concerned, our area of focus is much more specific and somewhat differentiated. While the PXS system was built to enable understanding of the dynamics of automated trading, and to simulate best practices, our goal is to test out a specific family of hypothesis that rely on

high-frequency automated trading, and seek to exploit the varying covariance between assets over different time intervals.

Sample Size

One of the most important steps for any statistical strategy is to backtest the process over a historical period to determine if it would have been profitable (Gutmann 2008). When the number of successful historical data points is large, there is better statistical significance that the strategy is useful. With other financial strategies, unlike high frequency trading, uses daily data is the smallest time unit. For a good sample set, and to ensure that the strategy works over similar time periods as the present, data from more than 10 years ago is not used. Therefore, this limits the number of data points that can be used in historical backtesting.

High-frequency trading, on the other hand, generates a large number of trades during the backtest. The data used in this strategy is tick level data-the price of an asset every tenth of a second. Thus, even within a single day, there is more than 20,000 data points, of which a large number would be successful. When testing a strategy if the number of successful trades is large, there is more significance to the strategy than a relatively small number of profitable trades, which could be attributed to a lucky streak.

Rebalancing Strategy

The strategy we chose to implement as our default strategy is an equal-weighted rebalancing strategy with two identical portfolios.

This strategy requires that the investor have two stock portfolios that are initially identical except that one portfolio, L is a long portfolio that a user invests money in, or buys, while the other portfolio S, is the one which the investor shorts, or borrows. A long position is one when the investor has a net positive amount of money invested in the stock. A short position is one where the investor has a net negative amount of money invested in the stock. This is achieved by borrowing the stock, and selling it, while paying interest on the stock until it can later be repurchased and returned.

The two portfolios are identical replicas of each other at the start, with each portfolio having its total worth being split equally among all the assets it contains. Both portfolios are rebalanced at a fixed time interval, with the portfolio L being rebalanced at a high-frequency interval (eg. 30 seconds) and the portfolio S being rebalanced at a much longer interval (eg. 1 months). Rebalancing essentially means recalculating the portfolios worth using the current stock prices in the market, and then using this new net worth to determine what value each of the assets should be readjusted to through either buying or selling. This algorithm is captured in the following illustration.

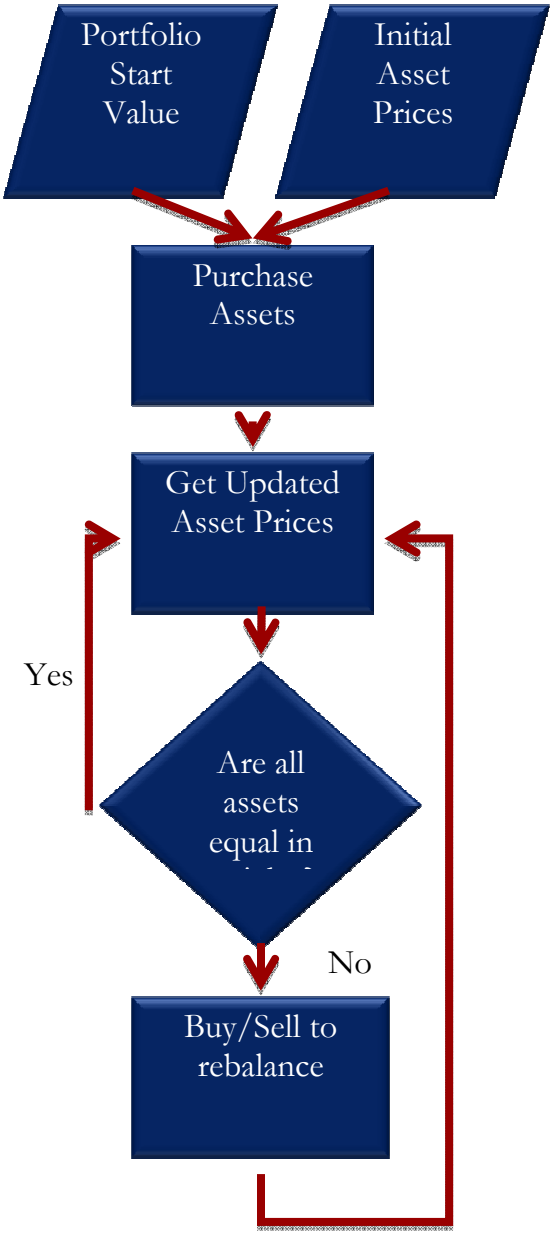

Figure 1: Rebalancing Algorithm represented using a flow diagram

In order to better understand how rebalancing works it might be worthwhile going through a simulated example of rebalancing happening at the seconds level. In the table below, a single portfolio is examined at different time intervals.

At the start of the day, at 9:30:00, the portfolio is created with a $1000 in cash. At this time Stock A is trading at $10 and Stock B is trading at $5. The very next second the portfolio is rebalanced, and equal amounts (in dollars) of A and B are bought so that now the portfolio holds 50 shares of A and 100 shares of B, each worth $500.

A minute later, lets say that the price of B rises from $5 to $5.5. As a result the holding in B is worth $550 now, while the holding in A is still worth $500, with the total portfolio being worth $1050. At this point, the portfolio is rebalanced and the new target value for each asset is determined to be 1050/2 = 525. 4 shares of B are sold to raise $22 in cash, which are then used to purchase 2 shares of A.

Time	Portfolio Value ($)	Asset A (Price x Qty = Total)	Asset B (Price x Qty = Total)	Cash ($)	Comment
9:30:00	1000	10 x 0 = 0	5 x 0 = 0	1000	Start with $1000 in cash
9:30:01	1000	10 x 50 = 500	5 x 100 = 500	0	Buy equal amounts of both assets
9:31:00	1050	10 x 50 = 500	5.5 x 100 = 550	0	Asset B price has risen
9:31:01	1050	10 x 50 = 500	5.5 x 96 = 528	22	Sell B to bring it close to 50%
9:31:02	1050	10 x 52 = 520	5.5 x 96 = 528	2	Buy A to bring it close to 50%
A few minutes later					
9:39:00	1024	9.5 x 52 = 494	5.5 x 96 = 528	2	Asset A price has fallen
9:39:01	1024	9.5 x 52 = 494	5.5 x 93 = 511.5	18.5	Sell B to bring it close to 50%
9:39:02	1024	9.5 x 53 =503.5	5.5 x 93 = 511.5	9	Buy A to bring it close to 50%

Table 1: Simulating a rebalancing strategy

After this sequence of transactions there is $520 worth of A, $528 worth of B and $2 worth of cash in the portfolio. Thus A and B are as close to being equal as is possible given that assets can not be bought or sold in fractions.

The next round of transactions starting at 9:39:00 shows a similar pattern where the drop in the price of A causes there to be less of A in the portfolio than there is B, as a result of which some of B is sold off, and some of A is bought.

Statistical Framework

The fundamental lesson learned in modern portfolio theory is that diversification of assets can lead to greater returns at lower risk. Thus, a portfolio, P, is simply a linear model where the weights, p_i, represent the amount invested in each asset X_i, such that the return for the overall portfolio is

$$r = \frac{dP}{P} = \sum_{i=1}^{n} p_i \frac{dX_i}{X_i} \qquad (1)$$

High-frequency data incorporate observations of these returns on a finer time scale, from daily down to every tenth of a second. If T is the given time period, and *i* is the basic unit of time within T, such that N*i*=T, then it can be shown with basic statistical theory that the product of the individual returns r_i is equivalent to the return over the entire time period T. This is known as the first moment.

$$R_T = \prod_{i=1}^{T} R_i \qquad (2)$$

A similar approach is used to find equivalence in the second moment, $r^2_{i.}$ Unlike the first moment, it can be shown that the second moments are not equivalent as the effects of covariance are present, the term that explains the relationship between two assets. Many studies have been conducted that showing evidence that correlation and covariance increase as the scale of the time interval increases. This provides an arbitrage opportunity in the stock markets, where an investor can take advantage of price differentials that occur within an asset.

The strategy employed here exploits the observation that returns diminish with longer time intervals between trades as the correlation increases. Two portfolios are held by an investor, each with identical assets. One portfolio is frequently traded, or rebalanced, within a given day. The second portfolio is rebalanced on a longer time interval such as every week, or month. The less-frequently rebalanced portfolio is held in a short position, while the high-frequency rebalanced portfolio is held in a long position.

Technical Approach

We have developed our system using the latest technologies in the Microsoft Developer stack. We use C# and the .Net Framework 3.5 for all of our components. There are two user front ends, one for the historical back-testing system and one for the live trading system. The historical testing front end is built using traditional Windows Forms, while the Live trading one is built using the more recent Windows Presentation Foundation (WPF) due to its superior graphical capabilities which allowed us to easily graph stock prices in real time.

Process Overview

The first, and major function performed by our system is that of allowing historical back testing of high-frequency trading strategies. In its current form the system tests out a high frequency, equal weighted rebalancing strategy. However by changing a few lines of code, any other high-frequency strategy can be easily tested. The figure below shows a flow diagram of the current system's performance.

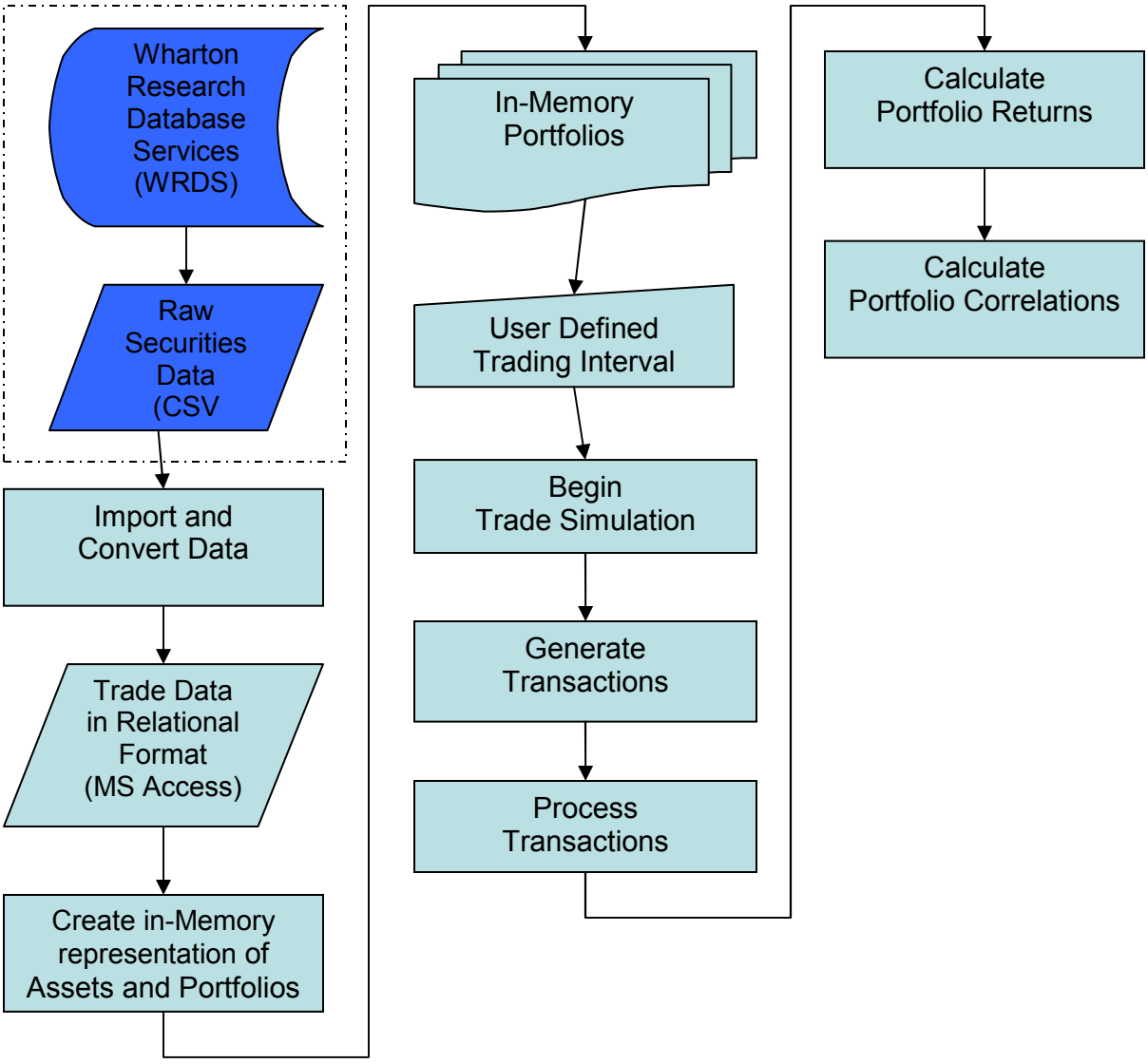

Figure 2: Flow Diagram showing how the system operates

As can be seen in the above figure, our system requires that raw securities data be provided as input. This data is then imported and converted by our Database system which creates multiple files, one for each asset or stock in the Microsoft Database Format (.mdb).

These database files are then used by our Price Engine to create in-memory representations of Assets and Portfolios. Next, based on a user specified time-interval, the Trading Engine creates Trading requests for each portfolio that the user wishes to test. Finally, the transactions generated by the trading engine are analyzed by our reporting engine which then calculates the covariances and correlations of the returns of individual assets within each portfolio.

System Components

Database System

The Database System comprises of two binaries once it is compiled. The DatabaseConnections.dll binary is responsible for handling all connections between each of the asset and portfolio databases and the rest of the components in our system. We incorporated functionality in our code to allow connection pooling so that at any given time the number of open connections to a given database is minimized and connections are shared whenever possible.

The DataEngine.dll binary is responsible for a number of tasks. In the first stage, it is responsible for taking as input the raw securities data and converting this data to the mdb format.

In our work we obtained this raw data from the Wharton Research Database Services (WRDS). WRDS hosts a number of databases out of which we used the New York Stock Exchange (NYSE), Trade and Quote (TAQ) Database. The TAQ database provided us with the each and every trade conducted for a given stock taking place each day since 1993. This data is obtained by accessing the Wharton CRSP website, defining the parameters for which the data needs to be obtained, including the Stock ticker, the trading dates and the minimum interval. The data is obtained in the form of comma-separated values (csv), which are stored by date in the format:

GM_2008_01_01.csv (TICKER_YEAR_MONTH_DAY)

Below is an excerpt from the data obtained from the TAQ database. As shown in the table, the TAQ database provides a listing of each and every unique trade that took place in the market on the particular day. The information provided for each trade includes the Symbol or Ticket, the Date and Time of the transaction, the Price and the Volume (Size) of the trade, and modality through which the trade was conducted in the final column. This last piece of information is not directly relevant for our system at the moment, although it could later be used to develop hypothesis about the different transaction costs resulting due to each modality of trading that could affect the decision to trade or not.

SYMBOL	DATE	TIME	PRICE	SIZE	EX
GM	20050103	9:26:57	40.07	1700	P

GM	20050103	9:30:38	40.65	234500	N
GM	20050103	9:30:39	40.65	100	T
GM	20050103	9:30:39	40.65	200	T

Table 2: Raw data obtained from the Trade and Quote (TAQ) Database hosted by WRDS

The DataEngine.dll binary is also responsible for loading and setting up the initial data for each asset and portfolio through the AssetManager and PortfolioManager classes.

Price Engine

The Price Engine which comprises of a single binary file, PriceEngine.dll, contains classes that are used to represent individual Assets and Portfolios in memory. The Asset class is where a large part of the logic to implement caching of trades in-memory is implemented. In our initial versions of the program, when we were directly using the raw csv files from TAQ without creating our own mdb representations, we would have to load entire csv files into memory in order to run quickly through them and perform calculations. However with the use of Microsoft Access Databases for each asset, we can implement caching of trades in-memory so that at any given time only a fixed amount of trades are loaded, while as the trading engine uses up trades to calculate returns during historical back testing, the Asset class pre-loads future trades for fast access.

The PriceEngine binary also contains a class called Transaction that represents each individual transaction including the time, price and quantity transacted, generated by the Trading Engine.

Trading Engine

The Trading Engine, which is the heart of high-frequency testing system, is contained within a single binary called TradingEngine.dll. This binary contains a number of classes that are used to setup a back-testing request, process multiple such requests, and to carry out live trading. The TradingRequest class creates trading requests with multiple combinations of portfolios and trading intervals in order to allow processing different kinds of tests simultaneously. The results generated from each test are stored in the TradingResult class. Finally the TradingEngine class is the one where the entire logic for second-level trades and rebalancing is contained.

Reporting System

The reporting system is contained within the TradingEngine.dll binary as well due to the intricate relationship between the two. The CovarianceCalculator class is where most of the heavy loading takes place in terms of calculating the individual assets returns, portfolio returns,

asset standard deviations take place. It is within this class that individual covariance and correlation matrices are constructed and then simplified into average covariance and correlation for a given portfolio.

Historical Backtesting Front-end

This is the front-end provided to the user which allows the user to create custom portfolios, import historical data and to add or remove assets from various portfolios. This Graphical User Interface application brings together multiple components of the system by allowing the user to seamlessly convert TAQ data in a single step to our mdb representation, and then use this converted data to test a strategy using a specified time interval. Finally, the summarized results of individual portfolios are provided within the GUI and the detailed results are stored in a text file which can be easily imported into MS Excel for easier viewing.

LiveTrades Front-end

The LiveTrades Front-end allows running a rebalancing strategy on a single portfolio of two assets using real-time stock quotes. These stock quotes are obtained by using an API for the Interactive Brokers Brokerage service which one of us had an account with. This Front end allows choosing two securities and then specifying the trading interval at which to rebalance them. The live prices for these securities and the resulting returns from the transactions proposed are plotted graphically on the front end.

Interactive Brokers provides a number of APIs for obtaining real-time trade data and for performing transactions online. These APIs however are targeted for use in either C++, Excel or Visual Basic 6. Since no API was provided for .Net, we initially had a very hard time trying to integrate this live data stream into our system. However we were fortunate enough to find a .Net wrapper for the C++ API which allowed us to interface with Interactive Broker's price feeds in a much easier way and allowed us to build this part of the system.

Challenges

Determining the 'actual' price

One of the challenges we face while building this system is that calculating returns on short, high frequency intervals is extremely challenging as the notion of a 'price' at a minute interval of time is tricky. Since market trades happen at an unpredictable frequency, it is hard to pin down what exact price should be taken at any given time since it is possible that at the

same split second, multiple transactions take place, or that none take place for quite a few minutes. We have developed algorithms to deal with this challenge that aim to strive for the most accurate real world transaction price possible. However our work is not complete and there is further room for improvement in the technique used for this purpose, since the method which is used to determine price at any given time has a critical impact on the trades made and the returns generated. The most optimal way to determine the legitimacy of the decisions made by our technique is to compare them to the decisions made by software systems developed by existing Hedge Funds or Investment Banks. However this requires access to proprietary data which is beyond our reach at the moment.

Performance and memory boundaries

Historical back-testing requires running a trading strategy on past data. The nature of the algorithms requires that for every single trade generated, the interim returns be used in the calculation of the ultimate returns, standard deviation and correlation of a portfolio. This brings about two opposing needs. The processing of a very large amount of data means that the data takes a very long time to process, and one way to speed this up is to load as much of it in memory as possible. However given the many gigabytes of data, loading it all at one time is just not feasible without access to some large kind of server, and even in that case, there are physical limits to how much data can be loaded at one time in memory.

We dealt with this problem first through rigorous optimization of our various algorithms. In order to do this, we employed code-profiling tools that helped us to measure accurately how many times each line of code was being run, and which methods were the biggest drain on running time. By doing this we were not only able to focus on the areas that would deliver the biggest performance boost, but were also able to identify many performance bottlenecks and iron out inefficiencies.

Integrating Real Time Prices

We had a very hard time trying to integrate real time prices into our LiveTrading system. The most frequently used sources of such price data are either trade brokerages or Bloomberg. Both usually require paid access and are quite expensive. We finally narrowed down to using Interactive Brokers because one of our team members already had an existing account with them which could be upgraded to avail the real time price data.

Once this account was obtained, there was still a challenge of integrating the price streams into our system. The existing APIs provided by Interactive brokers target C++, Visual Basic 6 and Excel. There was no API targeting .Net. This made our work difficult as it would require COM interoperability to call native C++ code using .Net. Thankfully, we were later on able to discover an API developed by a third party which provided a wrapper around the C++ Interactive Brokers API and allowed us to use it from within .Net with minimal hassle.

Results

Before we examine the results its worth revisiting our tangible goals outlined earlier.

1. Proving or disproving statistically, the hypothesis held by our advisor regarding breakdown of correlation over smaller time intervals
2. Building a generic system that can be used to test high-frequency trading strategies
3. Testing our system by employing the rebalancing strategy

We were able to achieve objective one by using the statistical framework outlined earlier in the paper.

Our second goal was to assess the hypothesis that the correlation between two assets decreases as the time interval at which they are rebalanced decreases. Our third and final goal was to test the hypotheses that the results generated by a trading strategy that rebalances equal weighted portfolios will increase as the time interval for rebalancing decreases.

To test these hypotheses, two portfolios were initially created. A sample strategy of 2-asset portfolios was tested, each containing shares of different stocks over two different years. The first portfolio consisted of Alcoa Incorporated (NYSE: AA) and The Walt Disney Company (NYSE: DIS) for the year 1993 (Figure 5.1a). The second portfolio contained American Express Company (NYSE: AXP) and The Boeing Company (NYSE: BA) (Figure 5.1b).

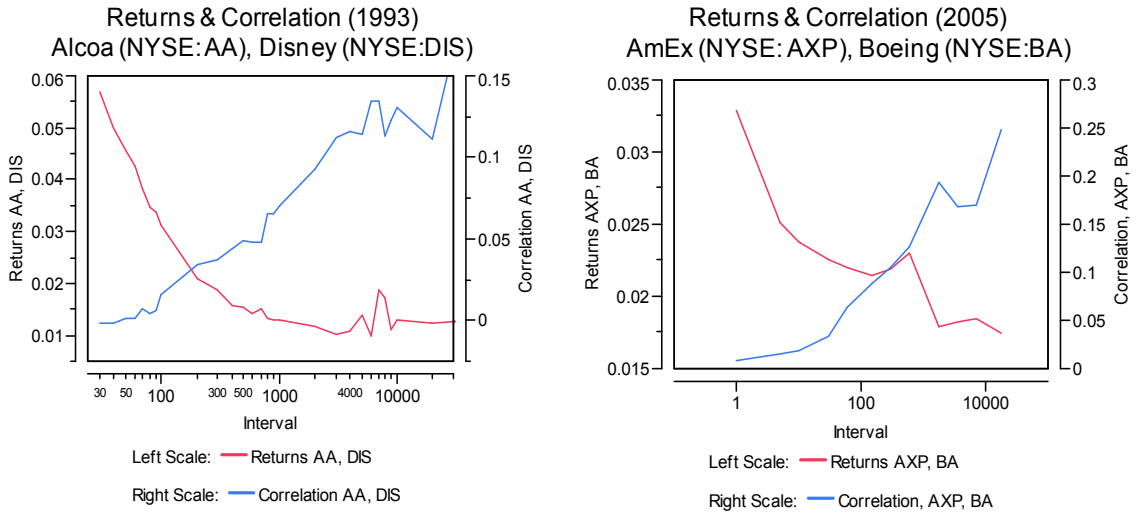

Figure 3: Correlation and Returns of 2-asset Portfolios

The trend for the correlation and the returns is as predicted in the theory. Furthermore, the increased level of sophistication in the technology is also seen in the above results. In 1993, the highest level of return occurred at the 30 second trading time interval, suggesting that

traders were unable to exploit price differences for smaller time units. In 2005, on the other hand, the highest returns occurred at 1 second.

Another one of the goals of this project was to create a platform that would allow end users to test different high-frequency trading strategies. The strategy that is employed in this simulation is to have two portfolios, each of equal value, consisting of the same assets, but in opposite positions. One portfolio is held in the long position (or where one owns the stocks that he is trading) and is traded at a high-frequency. The short portfolio (this occurs when a trader borrows assets, sells them, and then trades the proceeds) is held in a short position and is traded at a lower frequency. In addition, the borrowing cost of trading with a short position is also taken into account. Thus, whenever one borrows money, or assets, collateral is always given as insurance. Thus, Strategy 1 assumes that the investor put up 1/4 the value of the starting short portfolio as collateral, while Strategy considers 1/10. In other words, Strategy 1 has levered the portfolio by 4; Strategy 2 has levered the portfolio 10 times.

In 1993, a long portfolio of AA and DIS stocks was created, and rebalanced every 30 seconds. In tandem, anther portfolio, also holding the same stocks, was shorted as it was rebalanced every 1 month—a common strategy many portfolio managers utilize. The results of this simulation can be seen in Figure 5.2. It is shown that the returns of both Strategy 1 and Strategy 2 are much higher than the returns of the S&P 500 as well as the Dow Jones Industrial Average. One thing to highlight in these returns is the increased variability of the returns when the portfolio is levered at a higher rate. Thus, while Strategy 2 has a much higher return rate, it comes at the cost of higher risk, or variability.

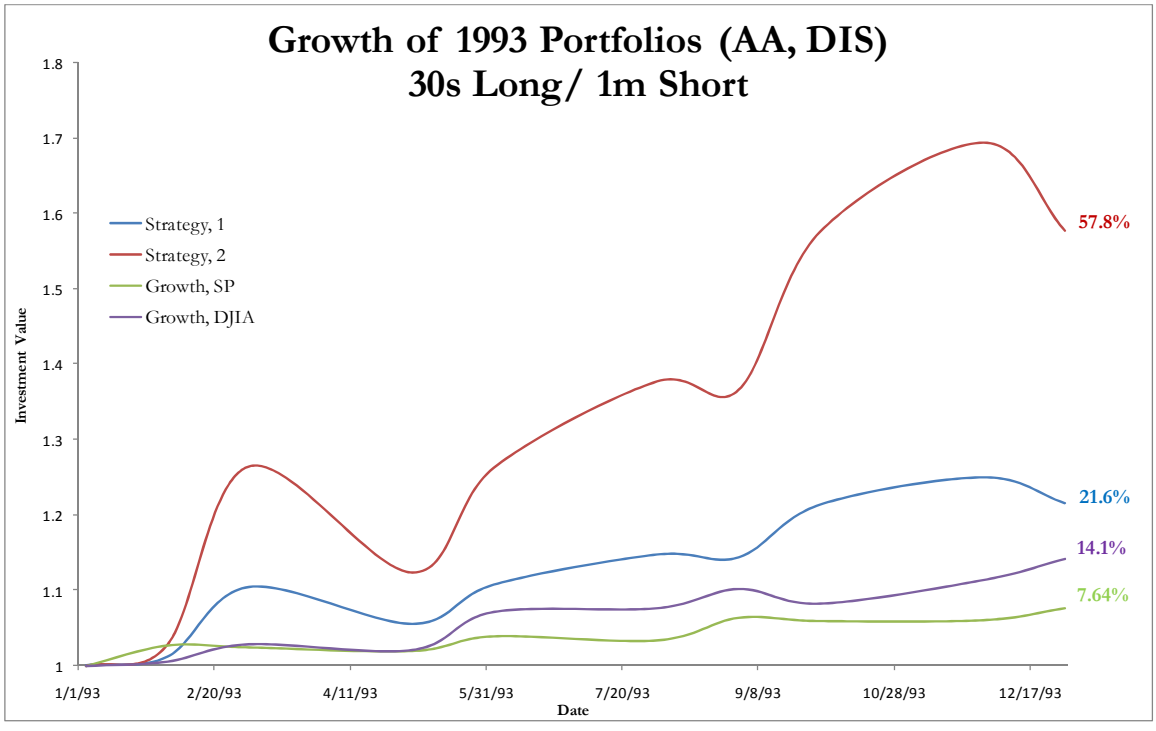

Figure 4: Portfolio returns for 1993

In 2005, a similar strategy was taken, except the frequently rebalanced portfolio's trading frequency has decreased to 1 second, with the lower frequency trades occurring at 1 month. Once again, as seen in the returns in Figure 5.3, the strategy does perform better than the market indices, with the highly levered portfolio having the greatest variability.

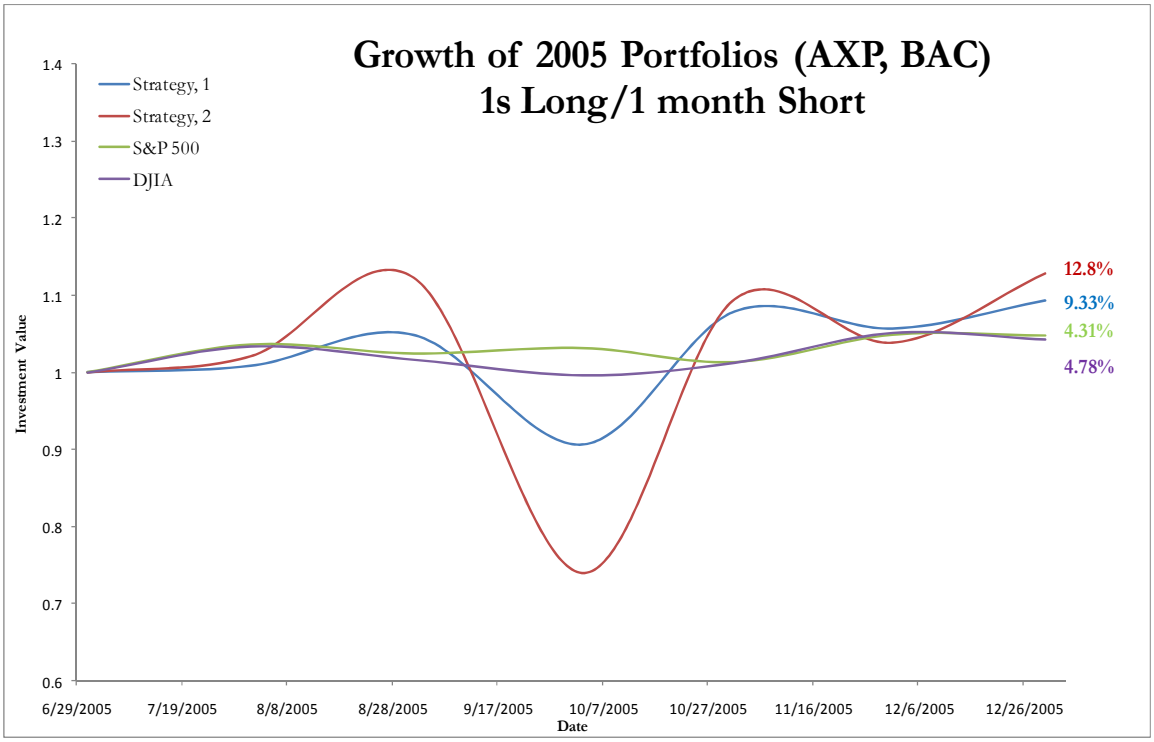

Figure 5: 2005 Portfolio Returns

To asses the risk of all the strategies, the Sharpe ratio was calculated for each. The Sharpe ratio is simply the return of the portfolio less the risk-free rate (90-day Treasury bill) divided by the standard deviation. A higher Sharpe ratio indicates a higher return for a lower risk level. The Sharpe ratio for the 2005 portfolios were 0.58 and 0.36 respectively for Strategy 1 and 2. In comparison, the Dow Jones Industrial Average, of whom all four stocks are a component, had a Sharpe ratio of 0.41 across the same time period, while the S&P had a ratio of 0.67. Similarly, calculating the Sharpe ratios for the 1993 portfolios showed that Strategy 1 and Strategy 2 each had ratios of 4.26 and 5.02 respectively which was much less than the Sharpe ratio for the S&P 500, whose ratio was 3.22. Other calculations were also made including the information ratio, to show that Strategy 2 would always be able to achieve higher returns due to higher effect of leveraging.

Conclusion

Working on this project was extremely challenging and rewarding at the same time, largely because it involved solving an open-ended problem. We started at the beginning of the year with a very small lead into a hypothesis that our advisor wanted to test, and during this

process were able to examine the strategy of rebalancing portfolios at different time intervals. In order to test this strategy we started out building a system that could merely simulate trades, but realized along the path that our system could be easily made flexible enough so that anyone could with a little bit of coding modify it to test other strategies.

Given more time, the first thing that we would like to implement would be a way of modifying trading strategies through some kind of user input rather than having to modify code. This is a little challenging at the moment since it would involve implementing some kind of trading language, or some markup system that is robust yet expansive enough to incorporate different kinds of trading scenarios and strategies. The most immediate benefit of this would be that it would eliminate the need of modifying the code, and would allow us to sell our product as a service instead of having to license it as a piece of code.

The most difficult aspect of this project was satisfying the needs of the various stakeholders in the project. Since this was an inter-disciplinary project across Computer Information Sciences, Electrical and Systems Engineering and Statistics, each of the three departments sought and prioritized different aspects of the project in different ways. As students who were ultimately working for a grade we at times had to make decisions, and undertake work that was done merely to satisfy the needs of the rubrics of the different departments, and to communicate the importance of the rest of the work to the parties who were not directly interested in it. This took away time and focus from the core goals of the project which are definitely worthy of further exploration. We intend to pursue this project after graduation and hopefully turn it into a viable commercial project in some form or the other.