High Frequency Trading

Jorge Guijarro Ordonez
Department of Mathematics
jguiord@stanford.edu

Samuel Persson
Department of Management Science & Engineering
joelpe@stanford.edu

Adam Wells
Department of Statistics
adamw356@stanford.edu

Hujia Yu
Department of Management Science & Engineering
hujiay@stanford.edu

Abstract

This paper explores two high frequency trading strategies. The first utilizes a Recurrent Neural Network to predict volume weighted average price movements from a variety of features generated from order book characteristics. The second is a high frequency statistical arbitrage strategy that incorporates stochastic control and principal components analysis. We show that both strategies produce encouraging initial results and discuss future work that would enable these strategies to be used in a real-world environment.

1 Introduction

Throughout the course of this project, our group worked in parallel on two different trading strategies, one involving Deep Learning and the other High Frequency Statistical Arbitrage, with the initial goal of combining the strategies or choosing the better performer towards the end of the quarter. However, as the quarter progressed, neither became clearly superior and, additionally, we were unable to fully combine the two since the Thesys backtester was incompatible with the deep learning frameworks. Therefore, instead of dropping one entirely, here we will discuss both due to the significant amount of work that went into forming each strategy and the encouraging results we obtained with both of them.

To this end, this report will be organized as follows. First, next section describes the general background for our two strategies, introducing the basic ideas behind limit order book dynamics and presenting a statistical overview of the behavior of financial data in high frequency. Then, section 3 describes our recurrent neural network strategy, addressing in particular the issues of feature generation, model formation, implementation of the strategy, and including our main results and conclusions. Section 4 is devoted to our statistical arbitrage strategy, and contains a description of our basic mathematical model and of some of the extensions we implemented (PCA and stochastic control), followed by a discussion of our methodology for hyperparameter tuning and its main results. Finally, sections 5 and 6 suggest some possible directions for future work and present our general conclusions.

2 Background

2.1 Limit Order Book Dynamics

We first introduce limit order book(LOB). For high frequency trading, multiple outstanding limit orders are posted to an electronic trading system and are summarized by stating the quantities posed at each level: this is known as the **limit order book**. The LOB data gives traders insight into supply and demand of market micro structure, and short-term price movements and is key to guiding high frequency trades.

Upon trading, the order tracking system automatically trades the best possible pair of orders. Figure 15 in Appendix shows that the price we see posted on the website represents the **mid-price**, which corresponded to the average of the best bid and lowest ask in limit order book. For high frequency traders usually we have to pay spread, meaning that we don't trade at the mid-price since we are executing orders directly from limit order book.

2.2 Statistical Overview of High Frequency Price Data

Before we dive into trading models and implementation details, we need to have some basic understanding of what high frequency data is and how they differ from traditional daily price financial data in order to make better modeling decisions. Some natural questions to ask is that:

1. Is there repeating patterns in high frequency data ? or is it more of a random process with no patterns? and
2. Can we use traditional time series models to model high frequency data ?

To answer question 1, we just simply compute and plot autocorrelations on high frequency price. According to Wikipedia, Autocorrelation is the correlation of a signal with a delayed copy of itself as a function of delay. [8]. We compute autocorrelation of data using equation below:

$$R(\tau) = \frac{E[(X_t - \mu)(X_{t+\tau} - \mu)]}{\sigma^2} \tag{1}$$

where $\tau = t - s$ is the time lag between the time we started observation(s) and the time lag we look at(t) when measuring correlation of a data series with itself. Notice that this function is a even function ($R(\tau) = R(-\tau)$), and therefore is symmetrical around 0.

Figure 1 shows computed autocorrelation results from MSFT log return data. Left graph is autocorrelation of MSFT daily log return and right graph is that of high frequency data(within a day).

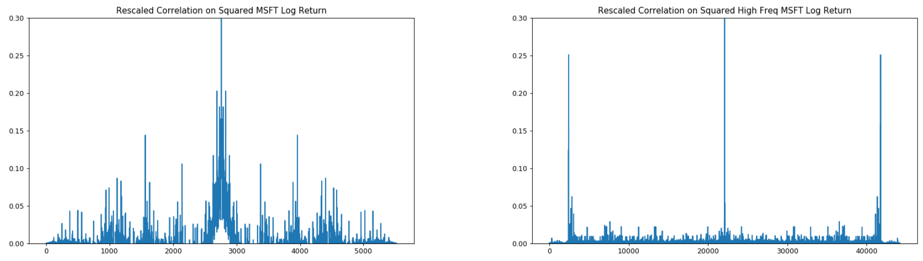


Figure 1: Autocorrelation on High Frequency Price vs. Daily Price (MSFT) [Left] Daily Price [Right] High Frequency Price.

According to these two plots, daily price return data exhibits significantly stronger patterns of autocorrelation, whereas high frequency data seems to have close to random correlations with itself as a function of time. This result gives us warnings that maybe we don't want to choose models that heavily rely on autocorrelation assumption of the underlying return distribution.

Since high frequency price return distribution fails to meet strong autocorrelation assumption of time series models. We should not assume the same autocorrelation property for high frequency price movements. As we will show below that this poses difficulties in choosing the most correlated high frequency pair to trade.

3 Recurrent Neural Network Strategy

3.1 Overview

In recent years, enormous advances in data collection, storage, and computational power have enabled the rise of deep learning and subsequent widespread use of neural networks for tasks such as image recognition, object detection, and natural language processing.

A Recurrent Neural Network (RNN) is one particular class of neural network that is specialized for processing a sequence of values $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(\tau)}$ and have been used with great success for language modeling, speech recognition, image captioning, and more. True to its name, this sequence of vectors is processed by applying the following recurrence formula at every time step:

$$h_t = f_W(h_{t-1}, x_t) \tag{2}$$

where h_t is the *new state*, h_{t-1} is the *previous state*, x_t is the input vector at time t, and f_W is a function with parameters W that is shared across each timestep. For a 'Vanilla' RNN, this process is typically given by:

$$h_t = tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

While useful as a simplistic model, this type of network often suffers from either exploding or vanishing gradients when processing longer sequences of data due to repeated multiplication of W during backpropogation. Numerous methods, like gradient clipping, Long Short-Term Memory units (LSTM) [3], and Gated Recurrent Units (GRU) [4], have been proposed to remedy this issue by improving gradient flow. For this project, we decided to use LSTM model due to its more prevalent use in existing literature. LSTMs are a special kind of RNN that are good at learning long-term dependencies. This is done through the use of 'input', 'forget', 'output' and 'cell' gates which optionally let information pass through a given unit and therefore enable the network to 'remember' information for longer periods of time. A mathematical and visual representation of a single LSTM unit are shown below [5].

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

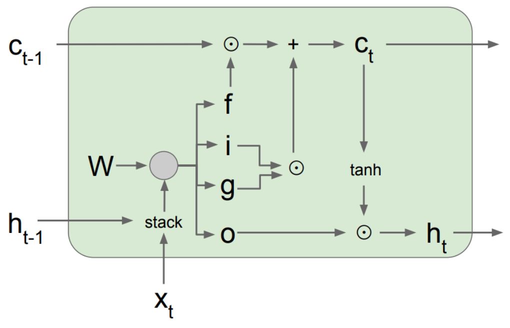


Figure 2: *LSTM Unit*

For this project, our intuition was to generate features from the order book from the previous n timesteps, feed this into a RNN, and make a single prediction about a future event. This is known as a *Many-to-One* architecture. Each of the n LSTM units will take as inputs a feature vector generated from the order book from the corresponding timestep and the outputs of the previous layer. Only the last unit will make a prediction. The figure below displays this model.

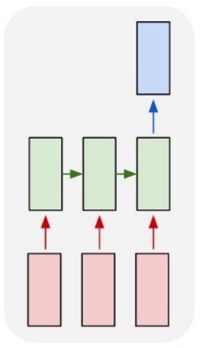


Figure 3: *Many-to-One Architecture* Input vectors are in red, LSMT Units are green, and the prediction is blue. This diagram would correspond to the timesteps $t-2, t-1, t$

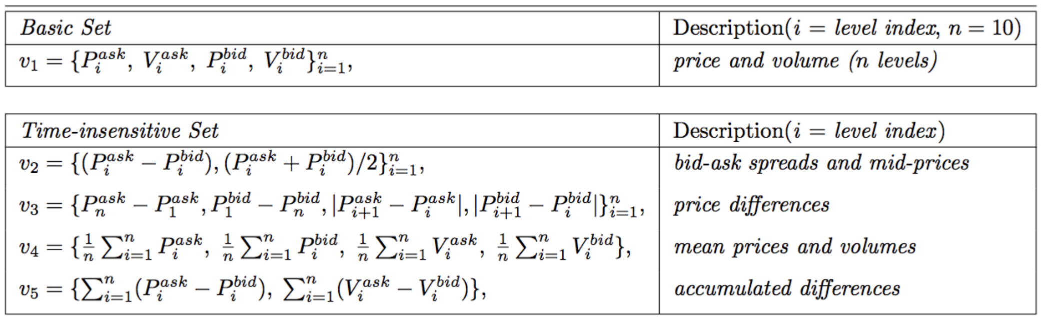
3.2 Feature and Label Generation

Throughout the quarter, we experimented with a variety of features and labels that would be used for formulate the RNN model. Overall, we found that careful selection of these were paramount for model success.

We primarily used features that shown to have predictive power in existing literature. In particular, models incorporating features proposed by Kercheval in *Modeling high-frequency limit order book dynamics with support vector machines* [6] seemed to produce the most promising results. These included raw order book data, such as bid and ask prices for 10 levels, and generated features like mid-prices, bid-ask spreads, mean prices and volumes, and accumulated price and volume differences. Additionally, we chose to include features for volume weighted average price, bid size percentage, and price and volume changes.

Regression and classification tasks were analyzed. In both cases, we initially thought about predicting mid-price movement since this had been explored with good success in academic literature [6, 7]. In a regression setting, this would simply correspond to the change in mid-price from the previous timestep. In a classification problem, we would create a (-1,0,1) label to denote whether the mid-price decreased, had no change, or increased from the last period. We also created similar labels derived from the Volume Weighted Average Price (VWAP). Given a sequence of trades $(p_1, v_1), ..., (p_T, v_T)$, this metric is calculated by:

$$VWAP = \Sigma p_t * v_t / \Sigma v_t \tag{3}$$



Basic Set	Description(i = *level index*, $n = 10$)
$v_1 = \{P_i^{ask},\ V_i^{ask},\ P_i^{bid},\ V_i^{bid}\}_{i=1}^{n},$	*price and volume (n levels)*

Time-insensitive Set	Description(i = *level index*)
$v_2 = \{(P_i^{ask} - P_i^{bid}), (P_i^{ask} + P_i^{bid})/2\}_{i=1}^{n},$	*bid-ask spreads and mid-prices*
$v_3 = \{P_n^{ask} - P_1^{ask}, P_1^{bid} - P_n^{bid}, \lvert P_{i+1}^{ask} - P_i^{ask}\rvert, \lvert P_{i+1}^{bid} - P_i^{bid}\rvert\}_{i=1}^{n},$	*price differences*
$v_4 = \{\frac{1}{n}\sum_{i=1}^{n} P_i^{ask},\ \frac{1}{n}\sum_{i=1}^{n} P_i^{bid},\ \frac{1}{n}\sum_{i=1}^{n} V_i^{ask},\ \frac{1}{n}\sum_{i=1}^{n} V_i^{bid}\},$	*mean prices and volumes*
$v_5 = \{\sum_{i=1}^{n}(P_i^{ask} - P_i^{bid}),\ \sum_{i=1}^{n}(V_i^{ask} - V_i^{bid})\},$	*accumulated differences*

Figure 4: Selected features proposed by Kercheval (2013)

The VWAP is a frequently used metric in algorithmic trading settings. A typical trading strategy is to enter a long position when there is a VWAP cross above because this signals that there is upwards price momentum.

Regardless of which setting and label we used, we quickly realized that label imbalance would be an issue. When predicting movement in the next second, our models would simply predict ≈0 for regression or the 0 class (no change) for classification. Upon further inspection, it became evident that this was due to a little order book movement on a second-by-second time frame. Since a very large percentage of our labels corresponded to no movement, the model would simply learn to predict that nothing will happen in the next time period. One partial solution to this issue is to alter the time horizon on which the label is generated. By increasing the time window of prediction, this class imbalance became less of an issue because the VWAP was more likely to change over this longer time period.

3.3 Model Formation

When formulating our model architecture, a number of decisions had to be made. The first of which was determining what cost function to use. As mentioned in the previous section, class imbalance posed an initial hurdle. To further encourage our model to predict non-0 values, we chose to use a weighted cross entropy cost function:

$$-1/m \sum_{i}^{m} wy^{(i)} log(\hat{y}^{(i)}) + (1 - y^{(i)}) log(1 - \hat{y}^{(i)}) \tag{4}$$

By placing higher weights on movement-related labels, we can further penalize the model for not identifying when the VWAP changed in the next time window. This encourages our network to predict values other than 'no change'.

A softmax layer was used as the output layer of the network. This causes the RNN to output probabilities for each class. For example, one possible output of the model would be [0.30, 0.30, 0.40], corresponding to the (-1,0,1) labels.

With this architecture in mind, we trained a separate RNN model for each stock of interest and chose the models that performed best on a validation set. Many hyperparameters, including the length of the prediction window, number of LSTM units, cost function weights, execution probability cutoff, learning rate, and dropout rate were tuned using a coarse-to-fine approach. One parameter of particular note is the execution probability cutoff. The intuition behind this parameter is that we don't want to execute a trade unless the model is sufficiently confident in its prediction. We therefore set a minimum probability bound for the trade to actually be executed. This should increase the precision of our execution actions as low confidence predictions would be disregarded.

Choice of these hyperparameters for each model heavily influenced the final results, making the tuning process extremely time consuming especially given that training was done on CPUs, not GPUs. The universe of stocks we were able to test was thus limited due to time constraints.

3.4 Strategy

The figure below displays our execution strategy. Simply put, at time t we will generate feature vectors from the previous n time steps from the order book. These will be processed by our RNN and output a single array of probabilities. If the maximum probability in this array corresponds to the VWAP moving up or down, we then check if this probability is above the execution probability cutoff. If so, we execute accordingly and then close our position at the end of the time window of prediction. The size of our order is exponentially weighted based on the predicted probability. We chose to close after this window because we did not want to continuously accumulate positions, which would essentially be a directional bet on stock price movement over the course of the trading period.

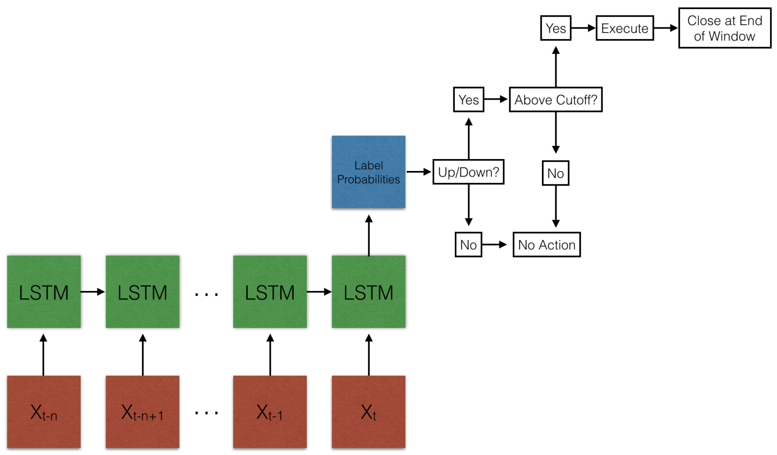


Figure 5: Execution Strategy

3.5 Results and Discussion

The universe we decided to analyze consisted of high market-cap companies in the technology sector. These companies were chosen for a few reasons. First, we were unable to use the Thesys backtester for our RNN models because it was incompatible with the TensorFlow and Keras frameworks. To minimize the affect of trading across the spread and to assume that our trades would cause minimal market impact, we restricted our universe to liquid stocks. Five stocks were tested due to the time consuming nature of tuning a model for every stock we wanted to trade.

Below is a plot of PnL over a single day for stocks MSFT, QCOM, TXT, NVDA, and CSCO when trading at the mid-price. The trends were relatively consistent with what we saw across the entire test set (aside from MSFT, which performed better during other days in the test set). The results were encouraging in that majority of the tuned models had positive returns and generally exhibited steadily growing profits without a lot of volatility. The returns were also consistent with a baseline of $50/$1Million Average Investment, which was suggested by Professor Borland. As previously mentioned, we opted for high execution probability boundaries to increase the precision of our models. This caused our models to generally trade a few hundred to a few thousand times per day.

	$ per Trade	**Precision when Executing**
MSFT	15.4	0.476
QCOM	63.7	0.546
TXN	56.1	0.539
CSCO	34.3	0.515
NVDA	55.6	0.537

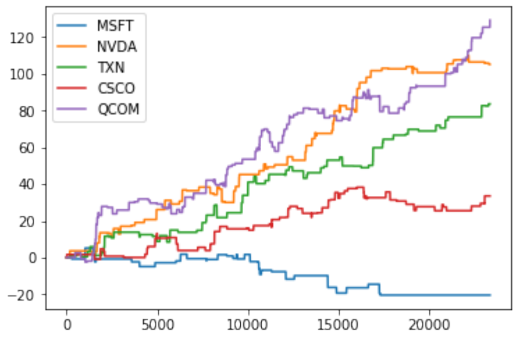


Figure 6: Single Day in Test Set Example

Table 1: Entire Test Set Results

One can quickly note how the MSFT model underperformed compared to the others. Upon further research, this appeared to be due to poor tuning of the cross entropy weights and execution probability cutoff. The model predicted 0 most of the time and as you can see from the chart, trades were made much less frequently than with the other models.

The largest caveat of our results are that these charts illustrate PnL when trading at the mid-price. When we tested our strategy by trading at the bid-ask, the profits were generally slightly positive, but not in a manner that made us confident that our strategy could be used in a production environment in its current state. Additionally, the scalability of our strategy is poor. We had to tune a separate model for each stock, which is difficult to do if we want to trade a large number of them.

Overall, we were pleased with the predictive power of our model and there are a few natural steps that could make this a viable trading strategy. First, an ensemble method could be used to incorporate magnitude of VWAP movement. Right now, the model only predicts the probability of the movement, not the magnitude. Creating a separate model and then combining these predictions could help in formulating more intelligent trading decisions. Secondly, with more data and access to GPUs, it would be possible to train a single model to predict this movement rather than tuning a model for each stock.

4 Statistical Arbitrage Strategy

4.1 Overview

The second strategy that we tried for this project involved statistical arbitrage (and, more particularly, pairs trading) in our high frequency setting, building upon the ideas of the celebrated Avellaneda-Lee paper [1].

The basic setup for this strategy may be summarized as follows:

1. First, we **identify historically correlated pairs** of stocks (which ideally are also very liquid so that the bid-ask spread is reduced and we can minimize the price impact).
2. Then, at a certain frequency which is chosen beforehand, we **linearly regress the past log returns** of each pair during a certain training period, and look at the corresponding time series of residuals. By hypothesis, we assume that this time series defines a mean-reverting stochastic process, in the sense that the residuals should oscillate between a mean value (economically, this means that we expect that the historical correlation that we observed will be the same in the future, except for some temporary mispricings that we will exploit to try to make profit).
3. We further assume that these residuals follow an Ornstein-Uhlenbeck process $dr_t = \kappa(a - r)dt + \sigma dW_t$, and **fit its parameters from the observed data** using some estimation method. In our case, we decided to use the method described in the appendix of [1], which calibrates the model via an AR(1) time series.

4. We then **execute trades if the last observation is far from the equilibrium** of the fitted OU process. More precisely, we define a "s-score" as $(r_{\text{last obs.}} - \text{mean}(OU))/\text{std}(OU)$ and, for some fixed thresholds that are chosen beforehand, we open/close a long/short position in a dollar-neutral way and with a certain leverage if the corrresponding threshold is crossed, as depicted in the figure below.

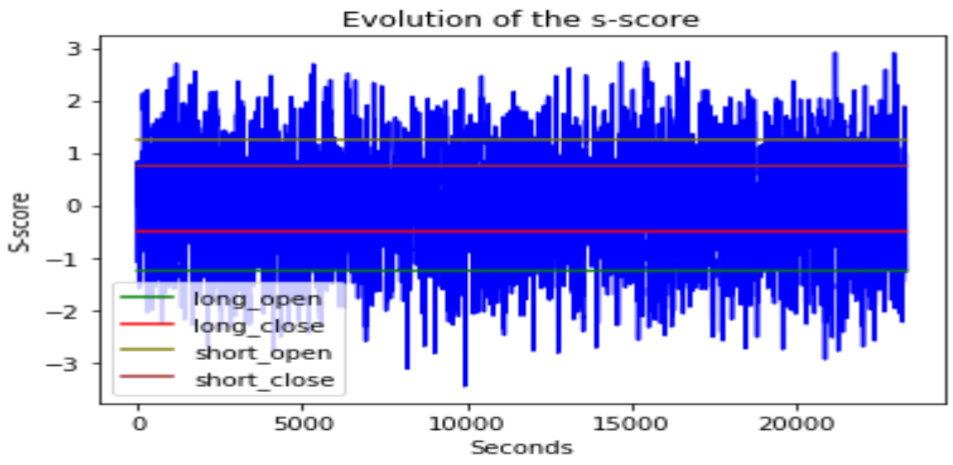


Figure 7: Example of the evolution of the s-score for the pair Apple and Google, with the corresponding fixed thresholds

In the following subsections we give the details for the implementation of this strategy, and in particular we describe how we identified the pairs we were going to trade and how we tuned the corresponding hyperparameters (frequency, size of the training period, selection of the thresholds, and leverage/risk exposure). The final subsections contain a description of our main results and suggest possible areas for further research.

4.2 Finding Correlated Pairs

Finding correlated pairs and trading on their correlation is the key to profiting in our model. Therefore this section will discuss how correlated pairs are defined and what are the findings and obstacles in identifying correlated high frequency pairs and trading them.

Correlation between pairs is computed the following way:

1. Define a pool of stocks we want to compute correlation matrix on. In this project we chose top 15 tech stocks with largest market cap.

2. Query and align data base on date time interval from Thesys database. Here I will be looking at time intervals of $[1, 5, 10, 20, 30, 60]$ seconds.

3. Preprocessing: convert to log return, standardize(zero-mean and normalize).

4. Compute correlation matrix and output the pair that has the highest correlation.

Following the above procedures, the most correlated high frequency pair for 60 seconds time interval for a 1-day window is `(QCOM, AAPL)`. Figure 8 below shows a snapshot of price movements during that day, where the left graph shows that between `10am-12pm`, two stocks move closely with each other indicating high correlation. Ideally, this is the pattern of stock movements we want to identify, predict and trade on.

However, after some more exploration into our data, results indicate that correlation between high frequency pairs is highly unstable, and therefore hard to capture and predict. More analysis is shown in section below.

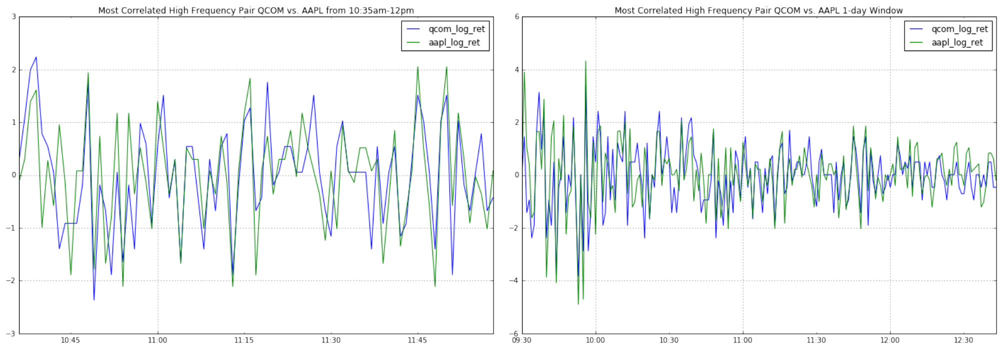


Figure 8: Most Correlated High Freq Pair Example. QCOM vs. AAPL, Correlation: 0.701826, 60 seconds interval, Date: 20170407

4.2.1 Concerns with Using Correlation: Unstable Correlation for High Freq Pairs

This section explains how correlation for high frequency pairs is highly unstable, which can make it difficult for us to find the more profitable pairs to trade. More specifically, the first question this section addresses is that: can we trade the same correlated pairs across different time intervals ? In other words, is correlation between a pair an inherent pattern between the two stocks or does it more or so depends on many outside factors ? To answer this question, we computed the most correlated pair out of 15 top tech companies across different time intervals ranging from `1 second` to `60 seconds`, the most correlated pair corresponds to each time interval and their correlation are shown in Table 2 below.

We can see that most correlated pair for different time interval is different. However, the pair `(QCOM, TXN)` and `(AAPL, FB)` appeared twice for different intervals, so maybe we could choose pairs based on how often a pair is the most correlated pair in the table. The point here is that correlation varies depending on how often we look at the data and execute our strategy, therefore we want to measure correlation dynamically as we change trading times intervals and can't assume static correlation between pairs.

Time Interval(s)	**Most Correlated Pair**	**Correlation**
1	(AAPL, FB)	0.253802
5	(AAPL, MSFT)	0.420605
10	(QCOM, TXN)	0.50241
20	(FB, AAPL)	0.565774
30	(QCOM, TXN)	0.609936
60	(QCOM, AAPL)	0.701826

Table 2: Most Correlated Pair for a 1-day Window. Correlation matrix is calculated on price data of the top 15 market cap tech stocks. Note that the most correlated pair is different for different time intervals.

In Table 3, we took a step further and show correlation between `(QCOM, AAPL)` over different time intervals to further show that correlation changes quite drastically as time interval changes and correlation between a pair cannot be assumed a property of the pair, but rather depends on outside factors.

Time Interval (s)	**QCOM AAPL**
1	0.185366
5	0.312954
10	0.408662
20	0.491515
30	0.588978
60	0.701826

Table 3: Correlation between QCOM AAPL for 1-day window.

The second question this section addresses regarding high frequency data correlation is that: Is there any predicative power in calculating correlated pairs for the same trading time interval ? In other words, if I fix my time interval and calculate the most correlated high frequency pair for today, can I use the same pair to trade tomorrow or over the next 7-day window ?

To answer this question, we pick the most correlated pair `(QCOM, AAPL)` as a result from Table 2 above, and compute its correlation over the next 20 day period. Figure 9 shows the evolution of correlation between the same pair. We discover that, sadly, there does not seem to be consistent patterns in which correlation follows even for the same correlated pair. Therefore, we cannot make the assumption that high frequency pairs that are tested to be correlated today will also be highly correlated tomorrow, as there is no inherent predictive power in correlation.

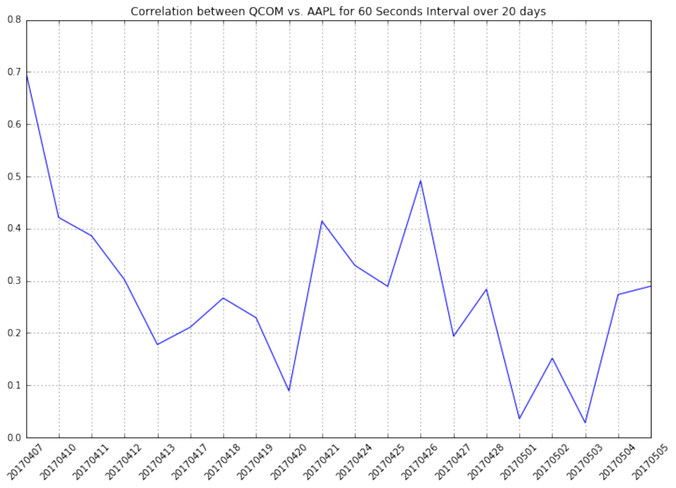


Figure 9: Correlation for `(QCOM, AAPL)` for 20-day Window.

From the above analysis, we make the following conclusions regarding identifying most correlated high frequency pairs:

1. Most correlated pairs differ by time intervals. Depending on how often we trade, we should calculate most correlated high frequency pair corresponds to that time interval.
2. Correlation for same high frequency pair changes by time intervals. We cannot make the assumption that there is one pair of stocks that will always be correlated in a high frequency setting.
3. Correlation does not have predictive power over next few days. We simply cannot take the pair that was tested to be correlated yesterday and trade on it for the next few days, since correlation may have changed entirely.

4.3 Incorporating Principal Components Analysis (PCA)

PCA is a natural extension of the pairs trading model to trade a portfolio of stocks rather than just two. PCA is used for extracting factors from data, which can be traded on. This approach uses historical log return data on a cross-section of N stocks going back M time intervals in history, depending on how often we execute measurements and trades.

Given the correlation matrix from section 4.2, calculating PCA is fairly easy. We simply just take the eigenvectors and eigenvalues of the correlation matrix, and rank eigenvalues in decreasing order, and rank its corresponding eigenvectors at the same time. A plot of the eigenvalues shows that the spectrum contains a few eigenvalues that contribute significantly more than the rest of the spectrum, as shown in Figure 17 in Appendix. So if we were to finish incorporating PCA in trading strategy, we would want to take a variable number of eigenvectors, depending on the estimated time interval, in such a way that a sum their corresponding eigenvalues exceeds a given percentage of the trace of the correlation matrix. This way we are essentially getting rid of noise in the portfolio by choosing factors that only explain a given percentage of the total variance in the system.

On the other hand, after we compute the eigenvectors and sort them with respect to eigenvalues, each stock return in the investment universe can be decomposed into its projection onto this pool of eigenvectors and a residual. This means that we keep only the significant eigenvalues/eigenvectors of the correlation matrix and add a noise diagonal matrix for the purpose of conserving the total variance of the system.[1]

In Figure 16 we have plotted coefficient values for the first eigenvector of the correlation matrix of our universe of stocks. This dominant eigenvector is associated with the 'market portfolio', and therefore should be all positive. Its coefficients corresponds to how much in each stock this market portfolio consists of. If we were to implement trading based on PCA factors, we would trade stocks according to higher-ranking eigenvector coefficients.

Also eigenvector coefficients can be used to evaluate correlation between pairs as well, the high ranking eigenvectors have coherence property - neighbors of a particular company tend to be highly correlated, therefore we can identify pairs by looking at nearby coefficients of high-ranking eigenvectors.

4.4 Stochastic control and optimal threshold selection

Once we have described how we can choose the stocks we want to trade, in this section we discuss how we select the best thresholds for our strategy. We consider two different options:

1. The first method consists simply in selecting a **fixed, constant threshold** for each pair of stocks and each trading day, as done in [1]. These thresholds must be calibrated rather manually to optimize the expected profits obtained through our strategy, and we describe the tuning process in detail in the following section.
2. The second method consists in using very recent research on **stochastic control** (cf. [2]) to compute *automatically and dynamically the best thresholds at each time* at which we trade. In this case no calibration is required, and there is only one "urgency hyperparameter" (ρ, which measures how fast we want to trade) that must be chosen.

The results obtained by using (and tuning the parameters of) both methods will be described in the next section, and here we just describe in more detail how the second method works.

The main idea is actually quite simple, and is to view the thresholds as stopping times maximizing some expected utility. More precisely, in this method the optimal times to exit a long or a short position when at time t the observed residual is $r_t = r$ are given, respectively, by the stopping times maximizing the following criteria functions:

$$H_+(t,r) = \sup_{\tau_+} \mathbb{E}_{t,r}[e^{-\rho(\tau_+ - t)}(r_{\tau_+} - c)] \qquad H_-(t,r) = \sup_{\tau_-} \mathbb{E}_{t,r}[e^{-\rho(\tau_- - t)}(-r_{\tau_-} - c)].$$

Here c represents the transaction costs of exiting the position, so $r_\tau - c$ is the profit we make when closing the position, and $\mathbb{E}_{t,r}[e^{-\rho(\tau_+ - t)}(r_{\tau_+} - c)]$ is just the expected profit adjusted by a discount factor (as commented before, ρ is just a measure of how fast we want to trade, or of the value of money in the future).

Similarly, the optimal times to open the positions computed above when at time t the observed residual is $r_t = r$ are given by the criteria

$$G(t,r) = \sup_{\tau^+,\tau^-} \mathbb{E}_{t,r}[e^{-\rho(\tau^+ - t)}(H_+(\tau^+, r_{\tau^+}) - r_{\tau^+} - c)\mathbb{1}_{\tau^+ \wedge \tau^- = \tau^+} + e^{-\rho(\tau^- - t)}(H_-(\tau^-, r_{\tau^-}) + r_{\tau^-} - c)\mathbb{1}_{\tau^+ \wedge \tau^- = \tau^-}],$$

and again this simply means that we maximize the expected profit of going long when the best time to go long is before the best time to go short (i.e. when $\tau^+ \wedge \tau^- = \tau^+$), and this profit is the expected profit made while closing ($H_+(\tau^+, r_{\tau^+})$) minus the cost of opening the position ($-r_{\tau^+} - c$), adjusted by a discount factor; and likewise for going short.

Then, to actually compute these optimal stopping times, one needs to solve the following Hamilton-Jacobi-Bellman variational inequalities

$$\max\{(\mathcal{L}-\rho)H_+(r);(r-c)-H_+(r)\}=0=\max\{(\mathcal{L}-\rho)H_-(r);(-r-c)-H_-(r)\}$$
$$\max\{(\mathcal{L}-\rho)G(r);(H_+(r)-r-c)-G(r);(H_-(r)+r-c)-G(r)\}=0$$

where $\mathcal{L}=\kappa(a-r)\partial_r+\frac{1}{2}\sigma^2\partial_{rr}$ is the infinitesimal generator of the OU process.

Solving these equations is actually a computationally difficult problem that involves first solving a system of ordinary differential equations and then a non-linear system of equations depending on the solutions of the ODEs, and there are many numerical issues that we had to address in our code (overflows, choice of numerically stable algorithms to solve the corresponding equations, approximations in degenerate cases, etc.). Also, the process depends on the calibration method that we used to find the OU parameters, the numerical algorithms that we use to solve the equations, and the choice of the utility functions that we make.

However, once the parameters a,κ,σ of the OU process have been calibrated and a ρ has been chosen, finding the optimal thresholds is now a completely automatic process that can be tabulated, and in fact the optimal thresholds are just functions of these parameters (and hence dynamic since these parameters are calibrated continuously). In particular, if we plot two of these thresholds along with the path of the last observed residual between Apple and Cisco during some seconds, we obtain the graph displayed below.

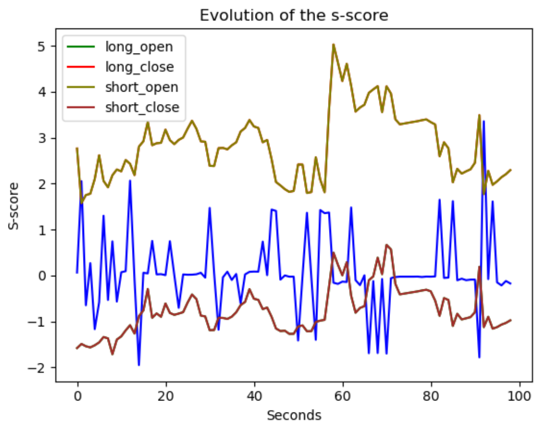


Figure 10: Example of the evolution of the last observed residual for the pair Apple and Cisco, with two of the corresponding dynamic thresholds

4.5 Hyperparameter Tuning of StatArb Model

Depending on the underlying architecture of the model we have 3-5 different hyperparameters, excluding the decision of which assets to trade. When using fixed thresholds for generating signals our model consists of three values for the thresholds, frequency for updating returns, and the number of time steps we base our regression on. If we generate signals by stochastic control we have an urgency parameter instead of fixed thresholds. This hyperparameter has a large impact on the number of trades the strategy makes.

The environment in which our model is implemented in differs significantly from the setting of Avellaneda and Lee. In [1] they are using daily returns and calculate residuals based on the last 60 days, while our goal is to implement this on a second level basis. Therefore, using the values of their hyperparameters did not transfer well into a high frequency setting.

The hyperparameter tuning process was done through two different approaches, grid search and random search. Grid search is a systematic exploration of the parameter space. By defining subspaces for each hyperparameter and then doing an exhaustive search we can find the combination of hyperparameter values that yields the best results. The approach also enables sensitivity analysis where we can observe the isolated effect of changing one parameter value. Figure 11 shows how the

PnL per trade varies with the interval size. As the figure suggests there is no clear linear dependency between the PnL per trade and interval length. Depending on which training size we use the PnL per trade behaves differently. We therefore believe that there is a dependency between the different hyperparameters where we need to find the optimal combination, which requires extensive search in the parameter space.

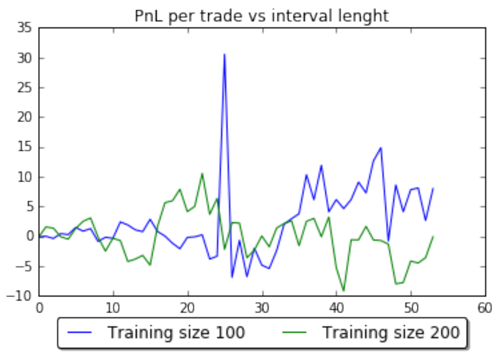


Figure 11: PnL per trade against different values of the interval length for two different training sizes.

The problem with grid search is that the approach is computationally expensive. The number of simulations needed scales exponentially with the number of hyperparameters and the number of values to explore in the subspace. This combined with a long simulation time using Thesys made the approach inefficient, simulating ten days of trading for five different pairs required roughly an hour.

To be able to explore a larger subspace we utilized a type of random search. By defining possible subsets for each parameter and draw a random sample for each hyperparameter every simulation we can simultaneously search through the hyperparameter space. We evaluate the performance each simulation and store the best performing setup. The downside of random search is that the tuning process becomes a black-box method where it is hard to isolate the effect of any single factor. Given the dimensionality of our problem and the long simulation time, the number of hyperparameter searches we were able to do per proposed strategy is not sufficient to conclude that the acquired hyperparameters are close to the optimal values for that training set.

We had two approaches for the process of picking pairs of assets used in the algorithm. As described in section 4.2, one process consisted of using correlation between returns as a metric for deciding which pairs seemed promising. Although we observed that the correlation is highly volatile over time we decided to let the picked pairs be static, meaning that we did not develop a dynamic strategy that continuously updates which pairs to use. Therefore, the model utilizes pairs that appeared correlated in a data set separated from training, validation and test set. The other approach was based on how the pairs performed on a training set. This was done by evaluating all possible pairs in the selected sample and picking the pairs that appeared to be promising in the sense that they had a consistent upwards sloping PnL. The process may seem arbitrary, but as will be shown in the next section, it delivered profitable results out of sample as well.

To evaluate which setup of hyperparameter performs better we used the metric PnL per trade and average trade size.

$$\frac{\text{PnL}}{\text{average trade size} \times \#\text{trades}}$$

The metric is a good indication of how profitable a specific strategy or a model setup is. However, one drawback we noticed is that it tends to favor making a few highly profitable trades over a more consistent stream of less profitable bets. A negative effect of this is that it is harder to conclude whether the few but profitable bets were due to randomness or if the underlying architecture actually provides tradable signals.

We conducted simulations on data from April and May 2017. Our training set consisted of 10 trading days between April 10, 2017 to April 24, 2017. The validation set was based on 5 days of trading in May 1, 2017 to May 5, 2017. And the test set was 5 days of trading between May 8, 2017 to May 12, 2017. The assets we considered were the 15 technology companies with the highest market capitalization at the beginning of 2017, this was done to avoid potential look-ahead and survivorship biases.

4.6 Results on Validation and Test Set

After hyperparameter tuning we had six proposed models to evaluate, and then select the strategy that appeared to be most promising. Table 4 shows the architecture of the six proposed models and their performance, in the sense of average PnL per trade (scaled by the average trade size), on the validation set. Not all models generalizes particularly strongly beyond the training set. While only one model is losing money out of sample, just three of the proposed models appear to be significantly profitable. The best performing model on the validation set was a model where signals are generated with stochastic control and using PCA for creating a portfolio against which to regress the returns. This model had an average PnL per trade of $0.000691. However, as previously mentioned, the metric tends to favor models that makes a few good trades. The best performing model had only about 10-15 trades per stock in five days of trading. We believe that this is due to the stochastic control component which in general appears to be resulting in fewer trades.

Table 4: Model architectures and performances on validation set.

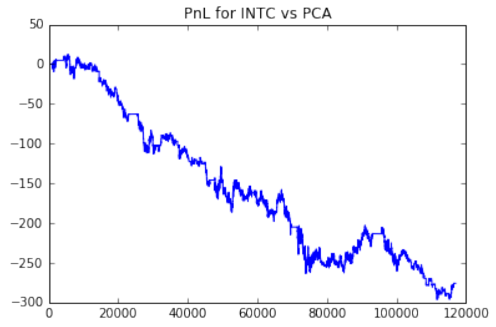
Thresholds	**Method for selecting pairs**	**Interval**	**Training size**	**PnL per trade ($)**
Fixed	Correlation	21s	575	0.000412
Fixed	Performance	21s	575	0.000528
Fixed	PCA	9s	279	-0.000000507
Stochastic control	Correlation	21s	575	0.0000222
Stochastic control	Performance	21s	575	0.0000792
Stochastic control	PCA	9s	279	0.000691

To gauge how our strategy will perform out of sample, without any biases, we conducted a five day simulation on a test set. We chose the model that had the best performance on the validation set, which was a model using stochastic control and PCA. The out of sample performance on the test set was 0.000871 in average PnL per trade. These results are encouraging, and signalizes that there is an underlying predictability in the strategy. However, it is worth noting that there are many aspects that differ from a real life setting which makes this number an upper bound for what this type of strategy could perform. Two strong simplifications with our model are that we disregard transaction costs and all executions are made at the midprice. Furthermore, the simulations do not account for the possible market impact our strategy would impose, although our order sizes are relatively small in comparison to the liquidity available in large technology companies.

In figure 12 we can see how the PnL changes over time for two stocks regressed on the PCA portfolio. This model regressed six different stocks on the PCA portfolio, which in turn consisted of 15 stocks. The two stocks behave very differently, one appears to be consistent in making money whereas the other stock is consistent in losing money. This could point towards a dynamic, or more thorough, process of selecting stocks could further improve the result.

In figure 13 we can, once again, observe how the PnL changes over time but this time the PnL is aggregated for all six stocks we used in the model. The graph does not point towards any specific conclusion regarding the consistency of the model. The profits does not appear to be driven by a few lucky trades, but neither is it continuously upwards sloping which would indicate a steadiness in profitability.

To further investigate the robustness of the model figure 14 shows the distribution of changes in PnL. The figure has been rescaled to capture what is important, the unscaled figure has a large spike at zero. Had the profitability of the model been driven by a few movements we would have observed a spike in the positive tail, and then a distribution centered around zero or skewed towards a negative value. It is difficult to make any strong claims about the distribution based on the figure. However, it does not show any signs of spikes in the positive tail, rather it appears to be slightly skewed towards a positive value suggesting that the strategy might be yielding steady returns.



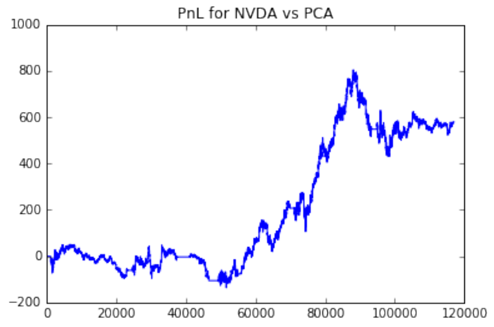


Figure 12: PnL over time for INTC [Left] and NVDA [Right] regressed on a PCA portfolio. Simulated on a test set.

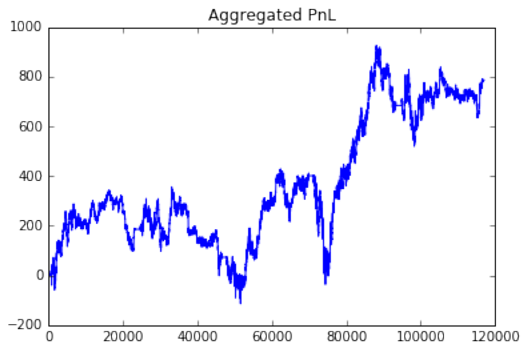


Figure 13: Aggregated PnL over time for six stocks (AAPL, INTC, NVDA, TXN, QCOM and MU) regressed on a PCA portfolio. Simulated on a test set.

5 Future Work

One primary improvement to the methodology would be to test the strategy in a setting more similar to a real world situation, meaning that we account for transaction costs and execute at actual prices instead of using the midprice. These simplifications were made to check whether there is any underlying profitability in the strategy. However, when these simplifications are taken into account the strategies might no longer be profitable. A secondary improvement would be to test the strategy on a longer time horizon and in an environment that is different from the one on which it was trained. Testing it in a high volatility setting would give some indications on how the model performs in extreme scenarios.

We have already touched upon some possible extensions to our strategies. Concerning the statistical arbitrage model, we have seen that the stock dynamics change significantly over time (see discussion on concerns using correlation Section 4.2). The current model setup is relatively static; we do not change what stocks to trade over time, the time length (interval) and time steps to look back (training size) are kept constant, and the weights in the PCA portfolio are not updated over time. Given the ever changing environment of the stock markets we believe that utilizing a more dynamic model in all these dimensions would improve performance and also extend the horizon in which this model is profitable. It could also serve to generalize the model better outside the training sample.

6 Conclusion

In this paper we have explored to approaches to constructing a trading strategy in a high frequency setting. The initial idea was to combine a mathematical model for statistical arbitrage with neural networks to enhance the process of identifying mispricings between two assets. However, we did

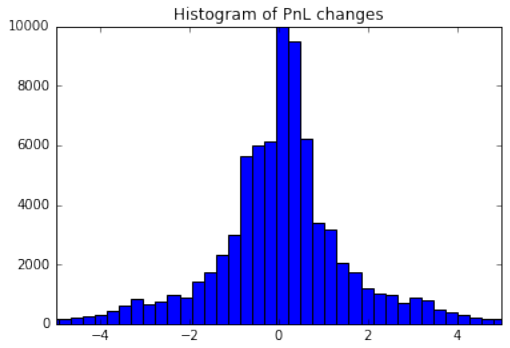


Figure 14: Histogram of PnL changes for the aggregated strategy (considering all six stocks). Simulated on a test set.

not find enough predictability in the residual process to motivate using Deep Learning over more traditional statistical tools. This led us to develop two separate strategies for algorithmic trading in a high frequency setting. We have applied machine learning techniques to find patterns in the limit order book and predict future price movements, executing the strategy if the model is confident enough. The model has provided some encouraging results regarding the profitability of such a strategy. We also developed a mathematical model for implementing statistical arbitrage in high frequency trading. We investigated multiple types of model architecture and found that some models made the strategy deliver profitable results out of sample. However, both strategies rely on heavy simplifications regarding transaction costs and executions. The models would therefore need to be analyzed in a more complex scenario to draw any stronger claims about the profitability of the strategies.

7 Code

The code for this project can be found in the following GitHub repository: `https://github.com/HujiaYuYoyo/448Project`.

References

[1] Avellaneda, M., & Lee, J. H. (2010). Statistical Arbitrage in the US Equities Market. Quantitative Finance, 10(7), p. 761-782.

[2] Cartea, A., Jaimungal, S., and Peñalva, J. (2015). Algorithmic and high frequency trading. Cambridge University Press, chapter 11.

[3] Hochreiter, S. & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9, 1735–1780.

[4] Cho, K., van Merrienboer, B., Glehre, ., Bahdanau, D., Bougares, F., Schwenk, H. & Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation.. In A. Moschitti, B. Pang & W. Daelemans (eds.), EMNLP (p./pp. 1724-1734), : ACL. ISBN: 978-1-937284-96-1

[5] CS231N Lecture 10 (2018). `http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture10.pdf`

[6] Kercheval, A. and Zhang, Y. Modeling high-frequency limit order book dynamics with support vector machines. University of Florida, 2013. http://www.math.fsu.edu/aluffi/archive/paper462.pdf

[7] D. Palguna and I. Pollak. Mid-price prediction in a limit order book. IEEE Journal of Selected Topics in Signal Processing, 10(6):10831092, 2016.

[8] Autocorrelation `https://en.wikipedia.org/wiki/Autocorrelation`

Appendix

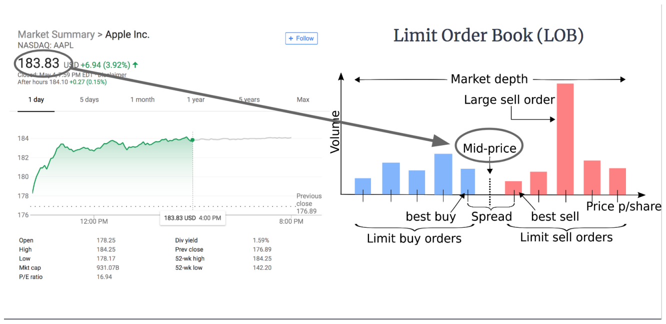


Figure 15: Limit Order Book

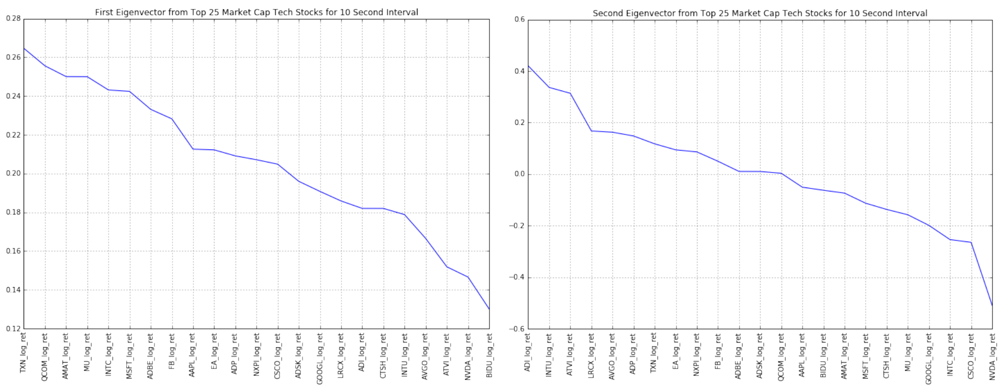


Figure 16: First(left) and second(right) eigenvectors sorted by coefficient size from top 25 market cap tech stocks for 10 seconds interval.

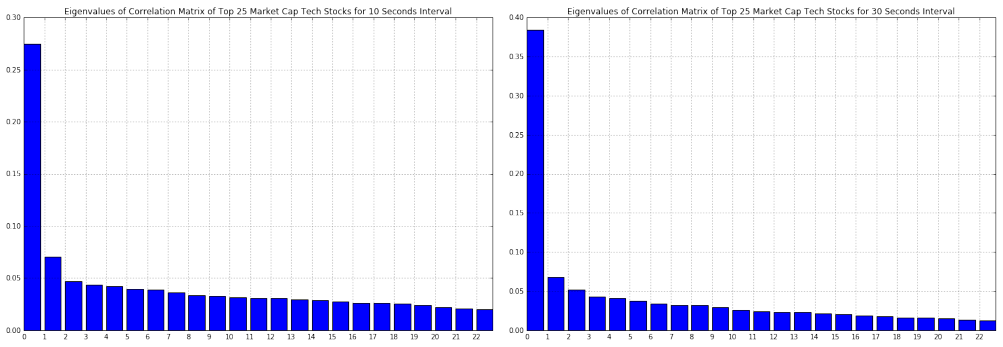


Figure 17: Eigenvalues of top 25 market cap tech stocks for 10 seconds(left) and 30 seconds(right)