Optimize VWAP trading to minimize risk in Market Making

David , Chris, Remy, Kunal, Nikhil, Mengliang*

7th Dec 2012

Abstract

In this project, we first analyze and interpret the data provided by Cantor Fitzgerald to understand the drivers for VWAP slippage, used to capture the efficiency of an opportunistic trading strategy. We then model the relationship between percentage of volume traded and return. Based on this data analysis, we simulate the portfolio evolution, including inventory accumulation, both cash P&L and mark-to-market P&L. In this way, we can build a trading strategy based on a black-box trading algorithm to optimize P&L from market making activities. The data analysis step is of great importance as we want our simulator to be as close to the actual market making activities as possible, which can then be used to facilitate our optimization analysis.

Keywords: VWAP, slippage, spread, percentage of volume traded, P&L, simulation

*David Lambl (dl688), Christopher Nieves (cdn25), Remy Pasco (rp443), Kunal Rajani (kr386), Nikhil Vaidya (nsv5), Mengliang Yu (my288)

Contents

List of Figures

1 Acknowledgement

First and foremost, we would like to thank to our advisor of this project, Dr. Sasha Stoikov for his invaluable guidance and advice. He inspired and motivated us throughout this project. His guidance contributed tremendously to this project. In addition, we would like to extend special thanks to our project sponsor Mr. Jacob Loveless from Cantor Fitzgerald, for providing us with an interesting project and all the trading data needed to help us towards a solution.

We would also like to take this opportunity to thank to Dr. Victoria Averbukh, Ms. Judy Francis and Mr. Jasper Beards at Cornell Financial Engineering Manhattan. Their coordination and hard work provided us with a good environment to work on the project and facilitated weekly meetings to make progress on the project.

Finally, an honorable mention goes to our families and friends for their understanding and support. Without the help of the particulars mentioned above, we would not be able to successfully finish this project.

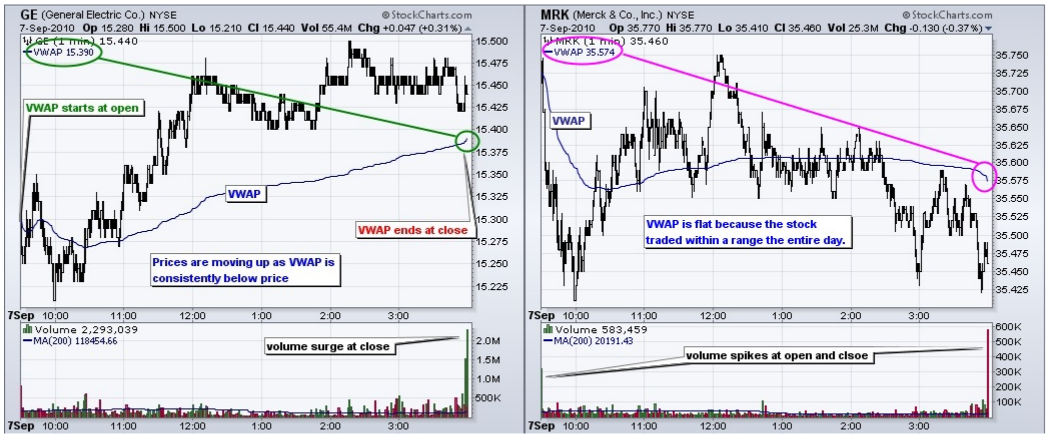


Figure 1: Vwap Price

2 Introduction

Volume weighted average price (VWAP) is calculated by multiplying the volume at each price level by the respective traded price and dividing by the total volume traded during a specific time horizon. It is a measure of the average price a stock traded at over the trading horizon. In general, the more volume traded at a certain price level, the more impact that price has on VWAP. The below formula can be used to calculate VWAP:

$VWAP = \frac{\sum_i P_i * Q_i}{\sum_i Q_i}$

i= each individual trade that takes place over the specific period of time

P_i= price of trade i

Q_i= volume traded of trade i

VWAP has some very interesting characteristics, which can be illustrated using Fig. 1

We see that the black line shows the price movement during the entire day while the blue line is the VWAP during the day. Though price fluctuates a lot, VWAP tends to be much smoother, as it's weighted by trading volumes. However, there are obvious drops in VWAP at the end of the day, which essentially is driven by the huge sell volume spike. Similarly, we can also observe a climb in the VWAP at the beginning of the day as there is a buy volume spike.

The main advantage of VWAP is believed to be its simplicity. VWAP as a quality of execution measurement was first developed by Berkowitz, Logue and Noser[1]. They argued that "a market impact measurement system requires a benchmark price that is an unbiased estimate of prices that could be achieved in any relevant trading period by any randomly selected trader," and then defined VWAP as an appropriate benchmark to satisfy this criterion. The use of VWAP as a measurement tool and execution strategy has soared in recent years. It is quite common nowadays for many institutions to use VWAP as a benchmark to measure the efficiency of a trader or a trading algorithm. Comparison against VWAP is very straightforward. A buy order executed lower than VWAP will be

[1]Berkowitz, S., D. Logue and E. Noser (1988): "The total cost of transactions on the NYSE", Journal of Finance, 41, 97-112.

regarded as a good trade while a sell order executed higher than VWAP is deemed as a good fill.

In addition, VWAP can help institutions to identify liquidity points. Implicit in the use of VWAP trading is the recognition that large orders traded in financial markets may trade at an inferior price compared to smaller orders. This is also known as the liquidity impact cost of trading large orders. Agency trading algorithms attempt to address this liquidity impact cost of large orders. The key is to break up orders with large volumes into many small orders to be executed over the VWAP period. VWAP can then help institutions to determine the liquid and illiquid price points for a specific security.

Though VWAP features its simplicity and has wide popularity to use, it has several shortfalls which should be taken into consideration. For example, VWAP is ineffective as a benchmark for less liquid orders as trading has a greater influence on the VWAP. If a trader is the only buyer/seller of a stock on a trading day, the execution price will simply equal the VWAP, which will then incorrectly indicate that there are no trading costs. Thus VWAP is especially poor when liquidity is a factor since impact is not being measured. In addition, VWAP emphasizes more on value, instead of price. A successful VWAP trade participates in proportion to market volumes across all prices, both high and low, which will bring extra risks.

The paper is organized as follows: Section 3 discusses data analysis. We analyze the data provided by Cantor Fitzgerald to understand the drivers for VWAP slippage, which is used to capture the success of an opportunistic trading strategy and the relationship between percentage of volume traded and return. Section 4 then introduces our simulator. We use real market data to simulate the portfolio evolution, including inventory accumulation, both cash P&L and mark-to-market P&L based on the results from data analysis. Section 4 summarizes and concludes this report. Section 5 explains the optimization of the aggression levels of the black box trading strategy to control inventories and maximize P&L. Section 6 summarizes our findings, and Section 7 is the code for our simulator.

3 Data Analysis

3.1 Sources of Data

TCA Reports.zip Due to the proprietary nature of Cantor Fitzgerald's opportunistic trading algorithm, we are unable to directly observe how the black-box makes its decisions. However, it is possible to develop predictive models for the algorithm's trading using previously executed trades.

The sponsor provided us with a detailed breakdown of all trades relating to the Opportunistic strategy occurring between May 5, 2012 and July 24, 2012. In order to analyze these trades and look for general trends, we made use of the following attributes:

Classification:

- Order ID Number – unique identifier given to each trade
- Symbol – the ticker symbol of the stock being traded
- Side – indicates whether a trade was a buy or sell trade

Times:

- Start Time – the time the order is routed to the marketplace
- End Time – the time when the trade is finished executing

Quantities :

- Cumulative Quantity – the number shares traded for this transaction
- Market Quantity – the total number of shares traded during the trade's duration
- Percentage of Volume = 100*(Cumulative Quantity / Market Quantity)

Prices :

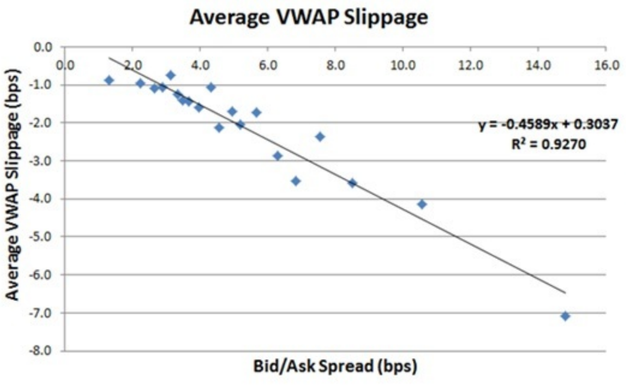
- Strike Price - the price of the security at the time the order was sent
- Average Price - the total cost of this trade divided by the number of shares
- VWAP Price - volume-weighted average price during the duration of the trade

Slippages :

- Strike Slippage = (Average Price / Strike Price) – 1 * negative for sell trades
- VWAP Slippage = (Average Price / VWAP Price) – 1 * negative for sell trades

Unfortunately, all trades in this dataset were made on aggression level 1. Therefore, we will eventually need to make some assumptions regarding how our findings will extrapolate to other aggression levels.

Costs.xlsx The sponsor also provided us with a spreadsheet containing 7969 trades which include information on the bid/ask spread. From these trades, we are able to obtain an average spread for 829 different stocks.



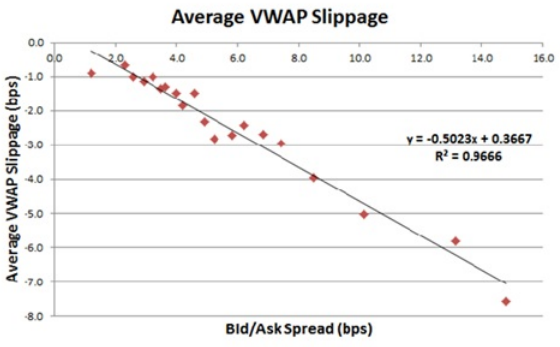


Figure 2: mean VWAP slippage for Buy(left) and sell(right) orders

3.2 Modeling VWAP Slippage

VWAP Slippage is an important way to measure how efficiently the black-box executes our trades, which ultimately impacts the P&L of our trading strategy. Before we can search for an optimal trading strategy through simulation, we will need a convincing model for the slippages.

We used the trade data to examine whether VWAP Slippage showed a dependent relationship on any other field. We explored using several different variables to aid in modeling VWAP Slippage but overwhelmingly the most effective predictor of slippage was the average bid/ask spread.

Linear Regressions For our analysis, we used the VWAP slippage data from the trades contained within the TCA Reports matched with an average bid/ask spread from the Costs spreadsheet. We separated the VWAP slippages into 20 bins based on which percentile range the trade's bid/ask spread falls in. Doing this helped us establish a relationship between the two variables by eliminating the noise due to randomness and other features we couldn't directly observe.

Average VWAP Slippages We model the expected VWAP Slippage of our buy trades by the equation:

$\mu_{buy} = -0.4589 * Spread + 0.3037$

The regression on these conditional averages showed a very strong relationship. Its R2 value of 92.7% suggested the majority of the variation in the average slippage can be explained by the security's spread (in basis points).

We arrive at a very similar model for the expected VWAP Slippages during sell trades

$\mu_{sell} = 0.5023 * Spread + 0.3667$

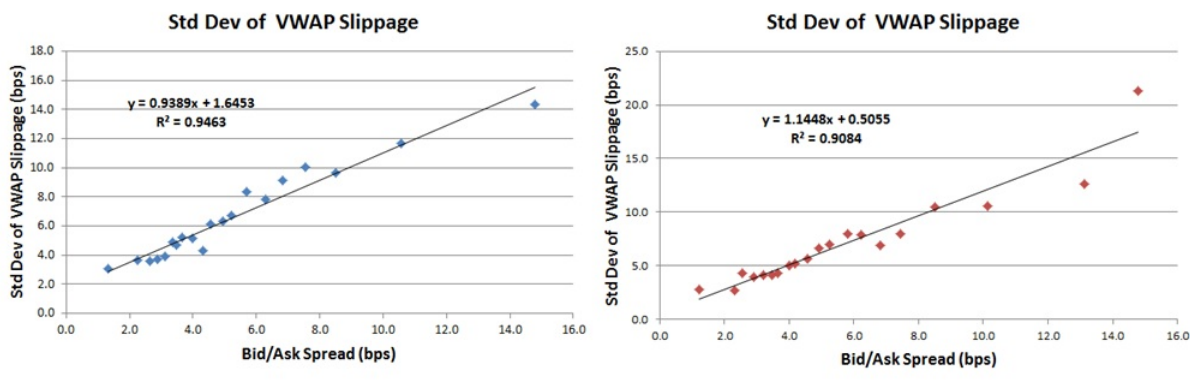


Figure 3: std of VWAP slippage for Buy(left) and sell(right) orders

Again this model explains most of the variability in VWAP slippages for each grouping. The regression of sell trade VWAP slippages in terms of spread has an R2 value of 96.7%.

The regressions can be seen in Fig. 2 on the previous page.

Standard Deviation of VWAP Slippages We also wanted to see if there was a relationship between the standard deviations of the VWAP Slippages and the spread. It turns out that slippages were more volatile when spreads are high.

Our regression model for the standard deviation of VWAP slippages during a buy trade:

$\sigma_{buy} = 0.9389 * Spread + 1.6453$

and sell trade:

$\sigma_{sell} = 1.1448 * Spread + 0.5055$

As can be seen from Fig. 3 both of these models were able to capture the majority of variation seen in the data. The R2 of the model for buy trades had an R-sq of 94.6%. The standard deviation model for sell trades had an R-sq of 90.8%.

Normal Model We initially tried to simulate VWAP Slippages using a normally distributed random variable whose mean and standard deviation are both functions of the bid/ask spread. VWAP Slippage$\sim N(\mu(spread), \sigma(spread))$ To see how this model holds, we generated VWAP slippages and compared them to the observed slippages using their Z scores. After subtracting the overall VWAP slippage mean and then dividing by the overall VWAP slippage standard deviation, we compared the simulated results to the observations: As the two histogram plots in Fig. 4 on the next pageshow, the results from the simulation seem to match up well with the observed results.

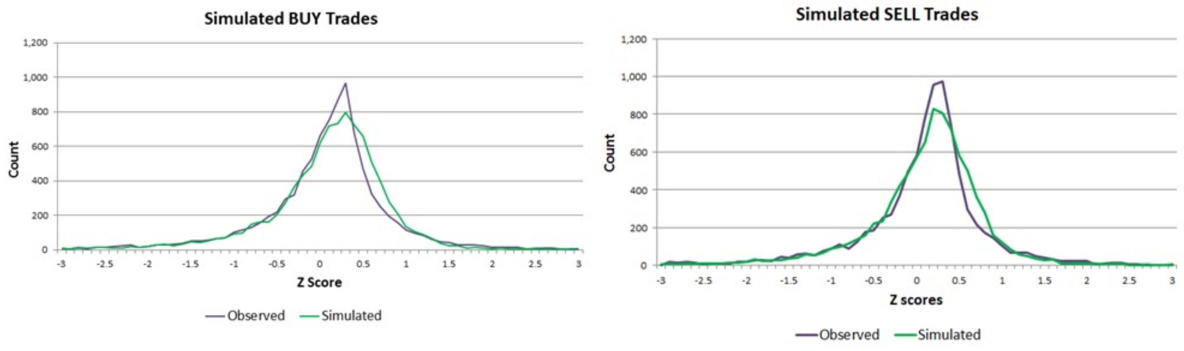


Figure 4: Simulated and Observed slippage

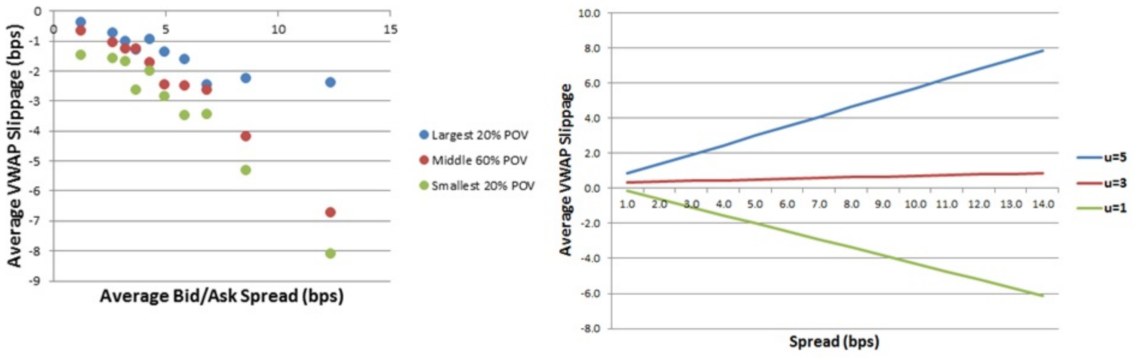


Figure 5: Change in Slippage mean with Aggressiveness

VWAP Slippage & Aggression Level The model above only models trades made by the algorithm at aggression level 1. While we do not have any VWAP slippage observations at other aggression levels, we attempted to make an well-reasoned estimate as to what these slippages may look like.

We know that increasing aggression levels should increase our market participation. We decided to split the existing data into three buckets:

- HIGH participation – trades with Percentage of Volumes in the top 20%
- MEDIUM participation - trades with Percentage of Volumes in the middle 60%
- LOW participation - trades with Percentage of Volumes in the bottom 20%

As we can see from Fig. 5 The VWAP slippages for stocks with wider spreads tended to be affected much more than for stocks with narrower spreads. This makes sense as large

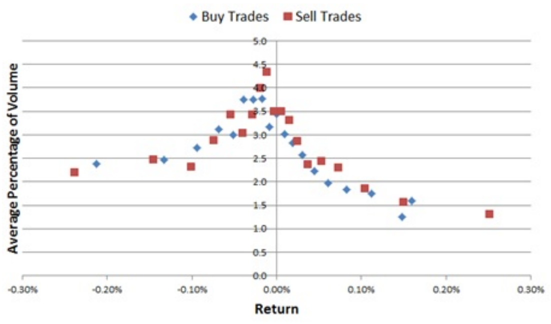


Figure 6: Empirical POV

spreads indicate less liquidity is available and thus our trading has a much more significant market impact. As a result, we assumed the impact of an increasing aggression level was linear with respect to price. We gave this impact a slope of +0.25 so that slippages would be roughly flat at u = 3.

Hence we modified our regression equations to:

$\mu_{buy} = -[0.4589 - 0.25 * (u - 1)] * Spread + 0.3037$

$\mu_{sell} = -[0.5023 - 0.25 * (u - 1)] * Spread + 0.3667$

We decided to keep the standard deviations unchanged by increasing the aggression level, but it may also be reasonable to assume there is more variability with more aggressive trading.

3.3 Modeling Percentage of Volume

Another attribute we wanted to model was the Percentage of Volume (POV). From our analysis, we determined that the change in stock price was the best indicator of our market participation. Note that we excluded trades with POV $>$ 30% as these outliers could skew our results.

For this analysis, we did not have intraday price data so it was not feasible to measure exactly how much the price changed during a trade. However, we decided that the percentage difference between the VWAP price and the strike price was a good substitute.

$Return = (-1)^{1_{\{sell?\}}} * [VWAP_{price}/Strike_{price} - 1]$

Like with VWAP slippage, we initially assumed that buy and sell trades would have different distributions. However, the two both had similar distributions of POV if we negate the returns for sell trades. Therefore, we consider a 10bp price increase when selling to be a return of -0.1% return. Using this definition of return allows us to see the similarity between these trades.

From Fig. 6we can see an inverse relationship between the magnitude of returns and the percentage of volume. During large price moves, there are likely more market participants so our participation decreases. Another important aspect is that large positive price

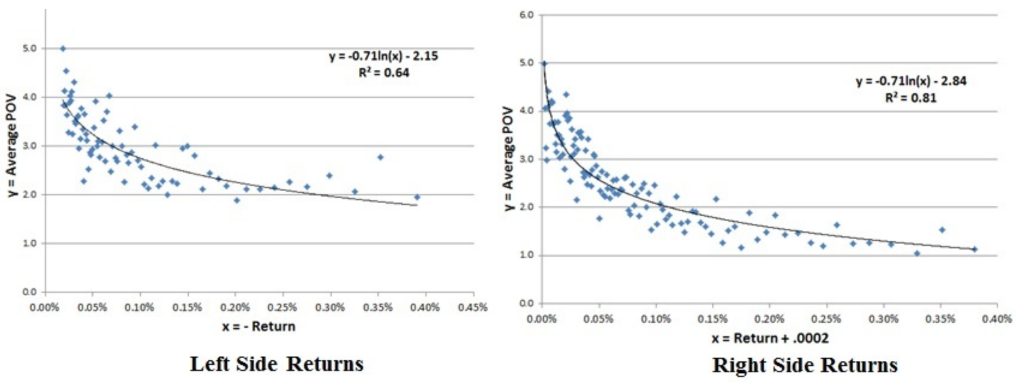


Figure 7: POV Regressions

moves, our trading averages roughly 1% of volume but for large decreases the POV levels off at roughly 2%. As an opportunistic market maker, we stand to benefit more from providing liquidity (selling during rising prices, buying during falling prices).

Regressions Fitting a single curve would be difficult for this data so we decided to break it up into two pieces. The curve above seems to peak right around a -0.02% return so we split the data along that line. We believe that a log regression would be most appropriate in this case. In order to fit the curve to a log regression, we needed to transpose the returns so that returns were all positive. For the left side, all returns were negative, so we define:

$x_{left} = -Return$

We then obtain the following regression:

$\mu_{POV} = -0.71 * ln(-return) - 2.15$ the R-sq of which was 64%.

On the right side, all returns were greater than -0.02%, so we shift the returns up by 0.2%, i.e. $x_{right} = Return + 0.0002$.

The regression yielded the following relationship:

$\mu_{POV} = -0.71 * ln(return + 0.0002) - 2.84$ the R-sq of which was 81%.

We see from Fig. 7 that our regressions fit the observed averages fairly well, but we were concerned that for large price moves the regression equations may result in unrealistic results. In the data, the percentage of volume seems to level off at extreme price swings but the regression approaches 0. We assign two constant values for POV when the price swing is above and below 0.4% to avoid this.

Piecewise Regression Model We then have a piece-wise function for the expected Percentage of Volume with respect to return. Using this measure, we plotted our regression model against the average POV for 200 bins based on the percentile of return, shown in the Fig. 8 on the next page.

$$\mu_{POV} = \begin{cases} = 1.76 & for\ Return < -0.40\% \\ = -0.709 * ln(-Return) - 2.154 & for - 0.40\% < Return < -.02\% \\ = -0.712 * ln(Return + .0002) - 2.837 & for - 0.02\% < Return < 0.40\% \\ = 1.059 & for\ Return > 0.40\% \end{cases}$$

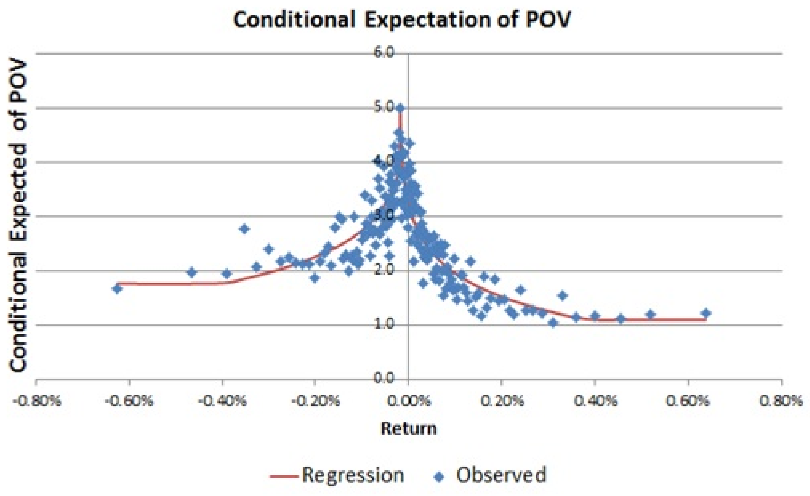


Figure 8: POV Computation

We believe that our regression does a good job in capturing the overall relationship between Return and expected POV.

Exponential Model Percentage of Volume has a lower bound of 0% with all of its outliers occurring on the right side. These features motivated us to select the exponential distribution to simulate our market participation. POV $\sim exp(\lambda)$ Since the expectation of an exponential is the inverse of its rate parameter, we set $\lambda = 1/\mu_{POV}$.

The scatterplots in Fig. 9 on the following page were constructed using the returns from 18,122 trades. This model seems to return results similar to our observations. Similarly, the conditional averages of Percentage of Volume also match up well.

Percentage of Volume & Aggression Level Again we needed to make an assumption about how aggression level impacts our results. We assumed that increasing your aggression level by 1 raises the average POV by 1%.

Therefore we set $\lambda = 1/(\mu_{POV} + (u - 1))$ and generate POV with an exponential random number. This change is depicted in Fig. 10 on the next page

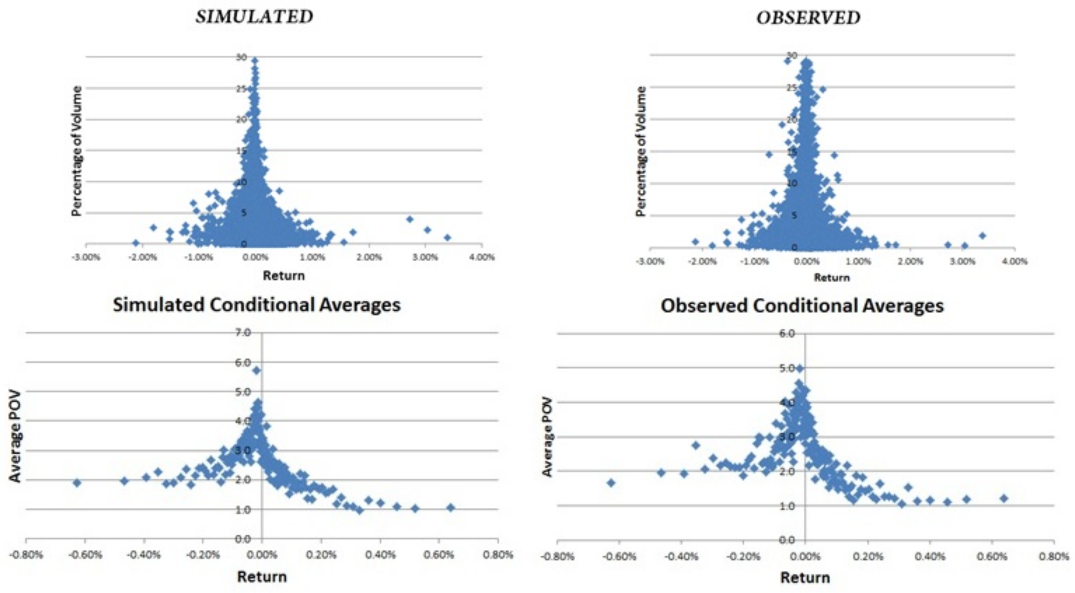


Figure 9: POV scatter plot and conditional averages

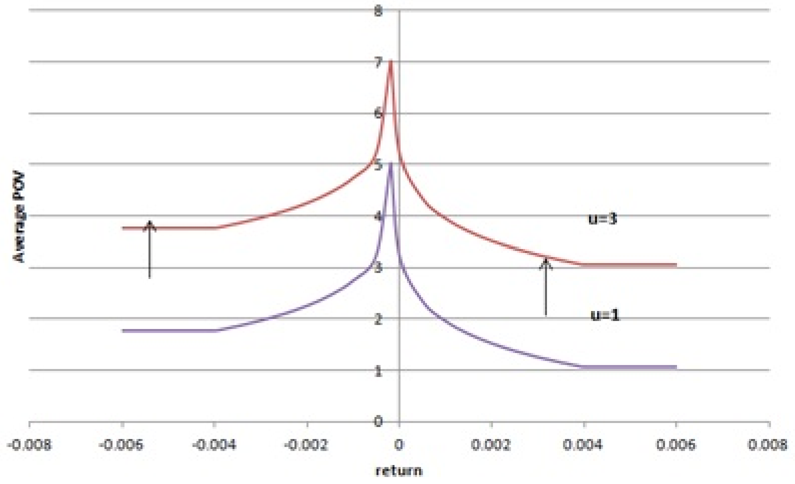


Figure 10: POV with aggressiveness

4 Simulation

The simulation can be broken down into the following steps:

1. We start off with some price, P, and inventory, I, at the beginning of the interval. In that interval we then compute the return of the stock and the spread of the stock. From the data we observed that the spread for a stock is constant and hence we also assume it to be a constant.

2. Based on this Inventory we also compute the aggressiveness of our market making. This aggressiveness is the key component that we try to optimize that is detailed out in the next section.

3. Once we have the return and the inventory we can compute the marked-to-market P&L. M2M P&L is the P&L that arises from holding an inventory as a result of market making and getting exposed to market risk. $I_{t-1} * \triangle P = MtM$ PnL

4. Next we obtain the volume traded based on the return on the stock. We use the relationship between return and percentage of volume (POV) traded as obtained from the data analysis. $\triangle P \rightarrow POV_{buy} \& POV_{sell} \underrightarrow{ADV} V_{t,buy} \& V_{t,sell}$

5. Next, based on the spread we obtain the slippage. Here also we use the relationship we obtained between spread and slippage from the data analysis. Once we have the volume traded and the slippage information we obtain the Cash P&L as highlighted above.

6. Also, after we have obtained the volume traded we compute the change in the inventory. This is the inventory on which we expose ourselves to market risk.

This entire procedure is illustrated in Fig. 11 on the following page.

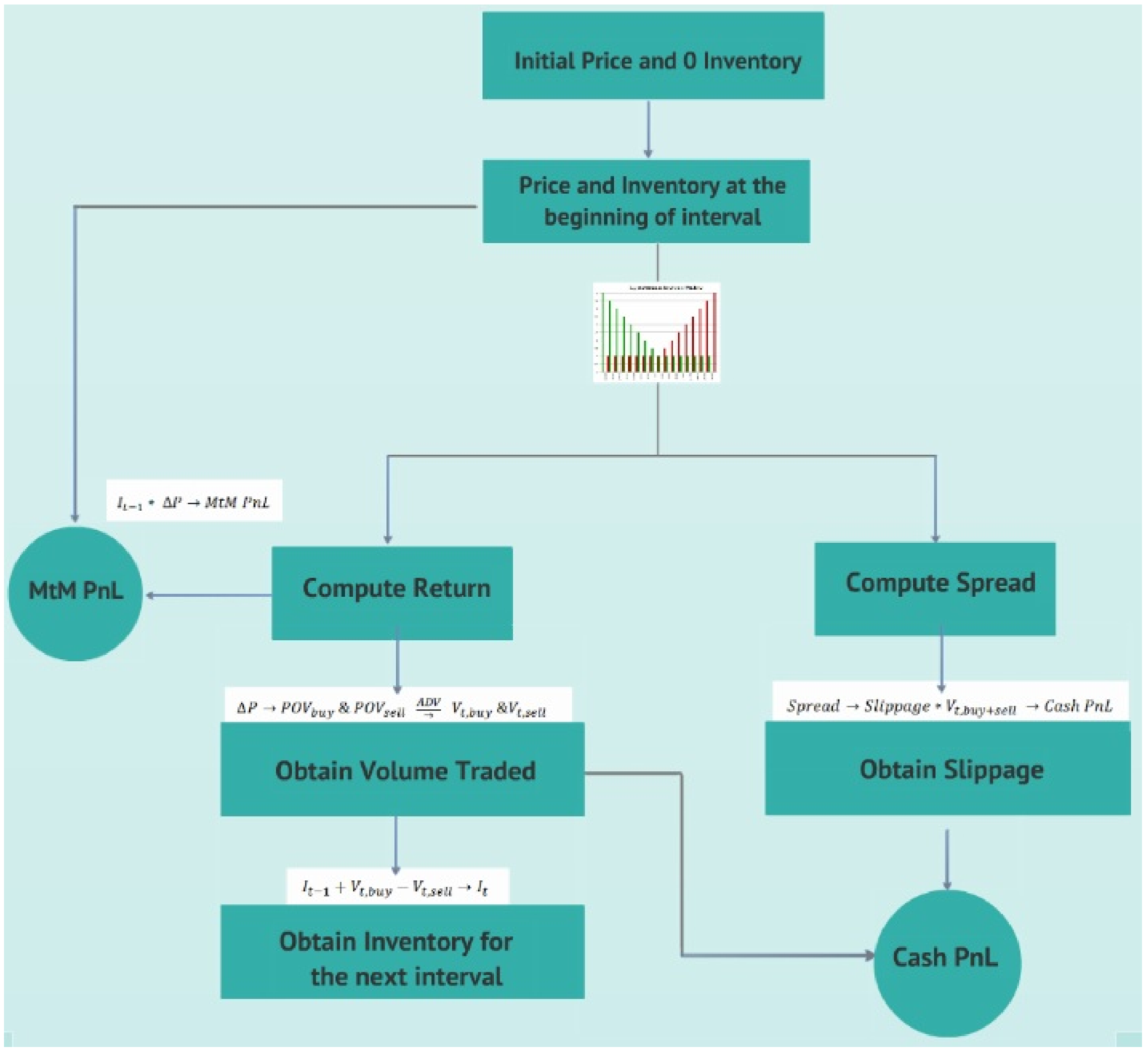

Initial Price and 0 Inventory
Price and Inventory at the beginning of interval
$I_{t-1} * \Delta P \rightarrow MtM\ PnL$
MtM PnL
Compute Return
Compute Spread
$\Delta P \rightarrow POV_{buy}\ \&\ POV_{sell} \xrightarrow{ADV} V_{t,buy}\ \& V_{t,sell}$
Obtain Volume Traded
$Spread \rightarrow Slippage * V_{t,buy+sell} \rightarrow Cash\ PnL$
Obtain Slippage
$I_{t-1} + V_{t,buy} - V_{t,sell} \rightarrow I_t$
Obtain Inventory for the next interval
Cash PnL

Figure 11: Simulation Flowchart

5 Optimization

Changing Aggressiveness : Parameters Optimization The aggressiveness parameters, both on the buy side and on the sell side, can take values between 0 and 5, increasing by increments of 0.5. The objective of the optimization was to find how these aggressiveness parameters should change. Since the objective is to limit our exposure to market risk we have chosen to model our market risk in a simple way with the following indicator function:

I= N*σ. N is the number of shares of the stock we are trading and σ is this stock's volatility. This volatility is computed using a trailing window of 20 days and using the daily closing prices. Once we have computed the current market risk for a stock we compute the corresponding aggressiveness using the formulas in Fig. 12 on the next page:

For a portfolio of stocks we have:

$I = \sqrt{N * Cov * N'_{stocks}}$where $N_{stocks}$is a 1xk vector that contains the number of stocks for each stock and Cov is the covariance matrix based on the last 20 days for each stock. Since I is a postive number that represents the standard deviation of the portfolio the aggressiveness is computed as:

$$u = \begin{array}{ll} 1 & 0 < I < LB \\ a + b * I & LB < I < UB \\ 5 & I > UB \end{array}$$

Hence the stocks for which we hold a positive inventory have their $u_{sell} = u$ and $u_{buy} = 1$ while the stocks for which we hold a negative inventory have their $u_{buy} = u$ and $u_{sell} = 1$.

LB and UB represent the lower and upper bound at which we start modifying aggressiveness and at which maximal aggressiveness is reached. These are the 2 parameters that we look for empirically, they are specific to each stock. Applying these formulas we obtain aggressiveness that have the following patterns in Fig. 12 on the following page (x-axis represents our market risk)

As our market risk increases we sell more aggressively in order to come back to a more neutral portfolio, while we buy more aggressively when our market risk decreases.

To find the pair of optimal bounds (LB, UB) we ran our simulation changing each of the 2 parameters and we looked at these 3 variables:

- Cash made from slippage: the value of our P&L due to slippage at the end of the simulation.
- Mark to market P&L: the value of P&L marked-to-market at the end of our simulation.
- Total P&L: the sum of the 2 previous variables.

The objective here is to maximize the cash made from slippage as well as minimize the mark to market P&L in order to obtain the highest total P&L that we can, no matter how the stock's price evolves. So as to do that we run the simulation for each pair of bounds several times and we take the average value of each of the variables over these different trials. This is only a pseudo Monte Carlo simulation as the computational costs to run our simulation for each pair of bounds only a few times are already high. And given the fact that for each stock we typically try around 10 different values for the lower bound and 10 different values for the upper bound, amounting to a total of 100 different pairs makes it pretty much impossible to run real Monte Carlo simulations. But despite

Sell Aggressiveness:

$$u_{Sell} = \begin{cases} 0, & if\ I < -UB \\ 1, & if - UB < I < LB \\ a + b * I, & if\ LB < I < UB \\ 5, & if\ UB < I \end{cases}$$

Buy Aggressiveness:

$$u_{Buy} = \begin{cases} 5, & if\ I < -UB \\ a - b * I, & if - UB < I < -LB \\ 1, & if - LB < I < UB \\ 0, & UB < I \end{cases}$$

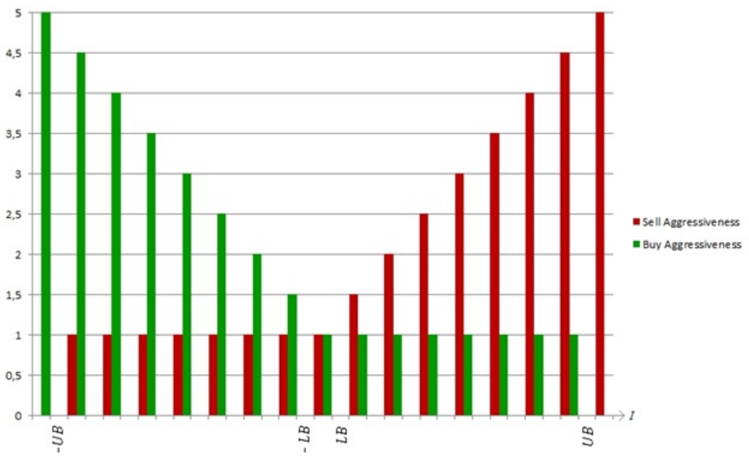


Figure 12: Criteria to change aggressiveness

the fact that intraday prices are randomly generated, closing daily prices are the same for a given stock which reduces the randomness and helps increase the accuracy of our results for only a few simulations. The results in Fig. 13 on the next page are obtained running the simulation with Morgan Stanley's stock over a period of 175 days.

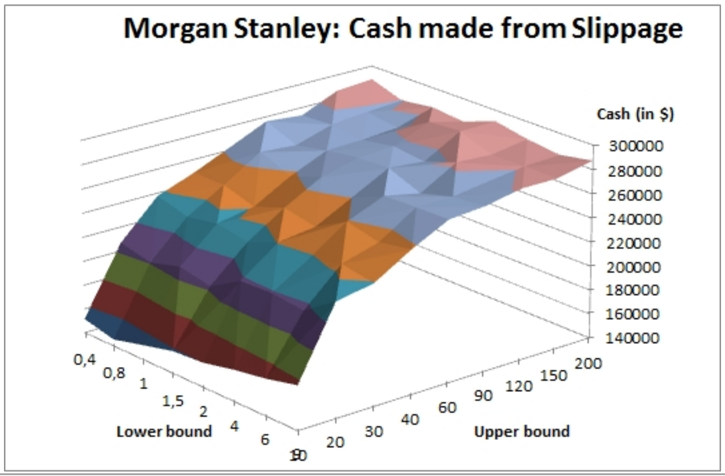
Even though the cash made from slippage keeps increasing as we increase the upper bound and the P&L from mark to market gets more and more negative, we can see that there exists a few pairs of (LB,UB) for which our P&L is higher and there exists a pair that maximizes our P&L. For this stock our results show that the optimal pair of parameters is (400, 40000). Whether we are able or not to find parameters that assure us we are going to end up with a positive total P&L independently of the stock's movements depends on the stock's characteristic. Characteristics that matter are the security's volatility, its average trading volume and its bid/ask spread to price ratio. To visualize this we ran our optimization on 3 other stocks.

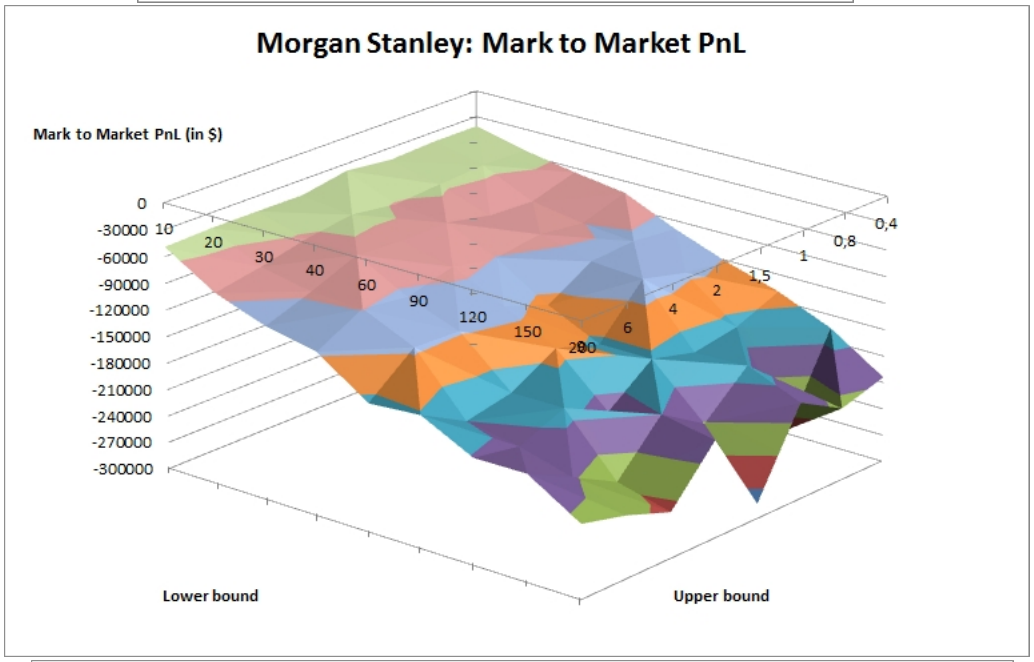
The following graph in Fig. 14 on page 20 corresponds to Citi, a very liquid stock, which makes it profitable. We can notice that the value of the optimal threshold is 3 times as high Morgan Stanley's, this can be linked to the fact that Citi's average trading volume is twice as big.

Now if we look at the results for Bank of America's stock in Fig. 15 on page 20, which is the most active stock in the markets, the optimal bounds are even higher than Citi's. Even though we do not have a closed form solution, this hints to the fact that trading volume is a parameter that weighs in the solution for the optimal bounds.

Finally trying to run our optimization on Goldman Sachs' stock we can see in Fig. 15 on page 20 that we are unable to find parameters that assure us we will profit trading the security. In the best case we end up with a small positive final P&L, but this value is too close to zero to really be significant. This is best explained by the stock's low spread to price ratio which makes it hard to profit while doing market making.

With these optimizations we see that we can find optimal parameters that allow us





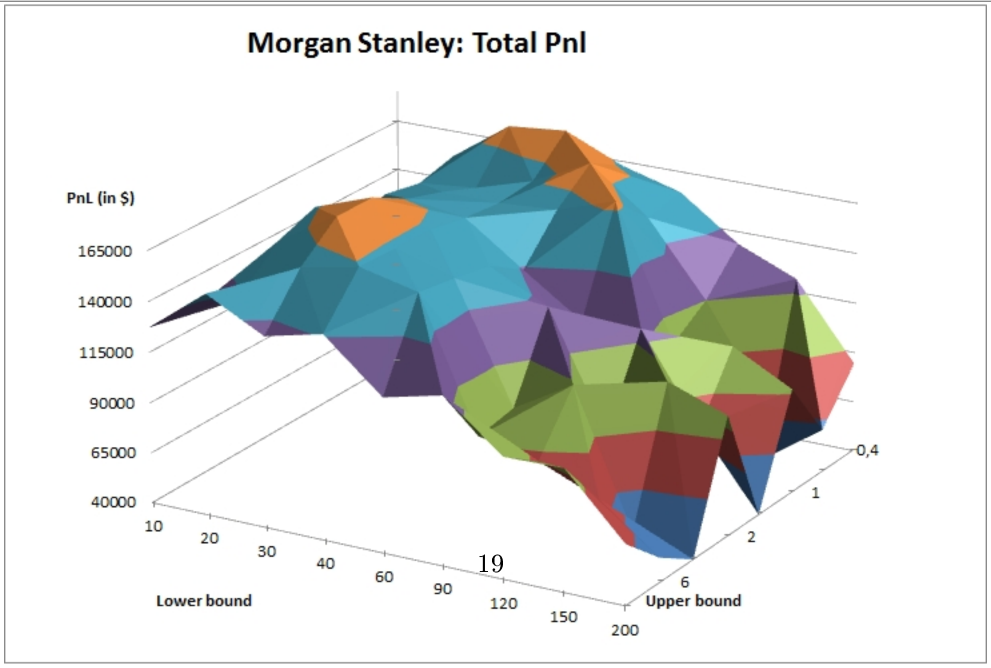


Figure 13: Morgan Stanley: Optimizing PnL to obtain bounds

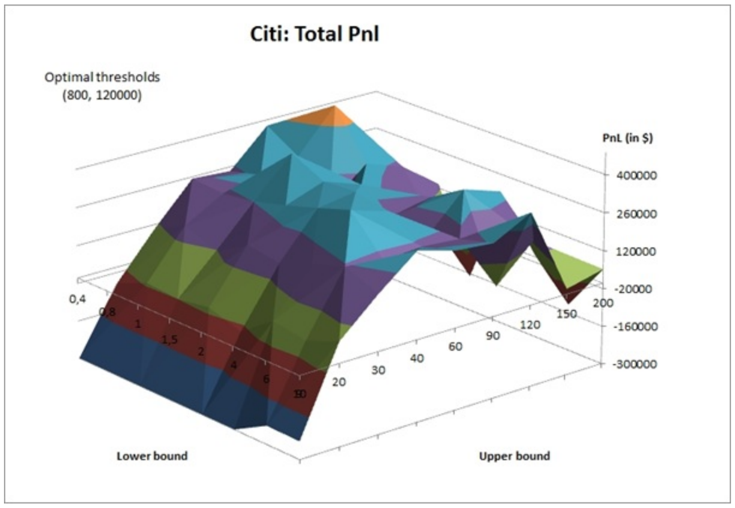
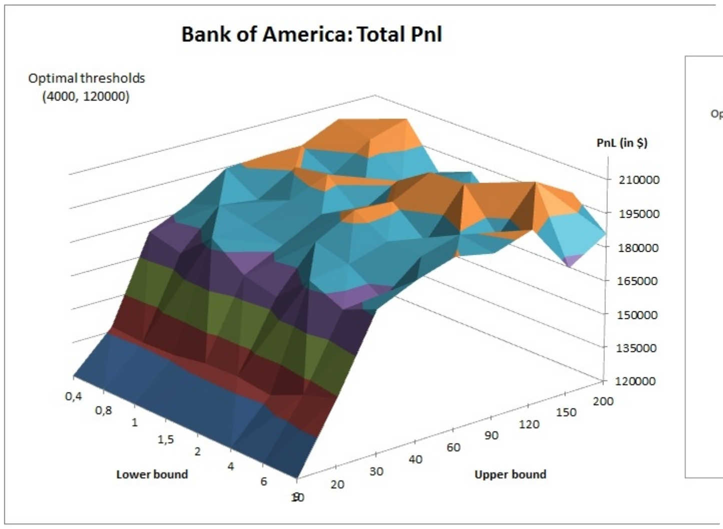


Figure 14: Citi PnL Optimization surface



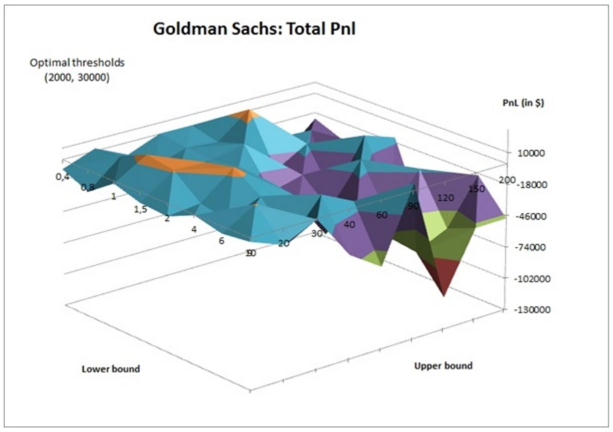


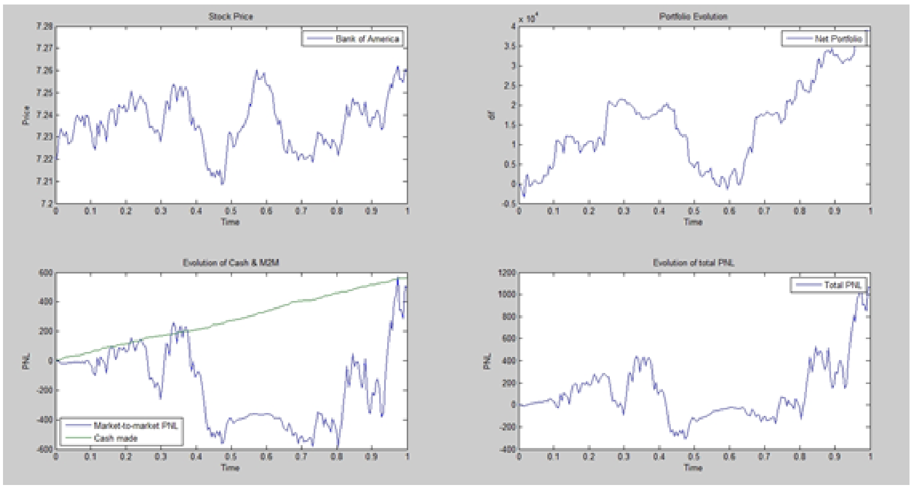
Figure 15: PnL Optimization Surface for BAC and GS

to control our market risk and any price movements thus maximizing our profit. While it is not possible to find optimal bounds that will make our strategy profitable for every stock, when we do find such parameters we can be sure we will profit from the P&L made on slippage without taking on much risk.

6 Results

The figure below represent the results of our simulation after 1 day of trading. In these graphs we ran our simulator using Bank of America data on January 23rd, 2012.

The first graph, in the top left hand corner represents the stock price over the trading day, which we consider to be 6 and a half hours long, from 9:30am to 4pm. To model these prices we use real data points for the start and the end of the day (the end of the day point is actually the start point on the following days), and in between we model the prices using a Brownian bridge. The drift and volatility are computed using historical data.



The 2nd graph, in the top right hand corner represents the evolution of our portfolio throughout the day. At every time of the day we are both buying and selling the stock and this graph simply represents our net position in Bank of America, starting with a neutral position at the beginning of the day. We can see that the evolution of our portfolio is negatively correlated with the changes in price. This is due to the fact that we are market making so we are providing liquidity. Therefore as the market moves up we sell more stock than we buy, and in the same way as the price moves down we buy more than we sell. But as was explained before when there are big moves in the stock price over a short time period we will trade a lesser volume to avoid taking on too much market exposure too fast.

The third and fourth graphs, bottom left hand corner and bottom right hand corner respectively, represent our P&L. The third graph is a decomposition of our P&L as the money that we have made due to slippage(green increasing line) and the marked-to-market value of our portfolio (blue line moving somewhat randomly). The 4th graph is the total value of our portfolio; it is the sum of the cash we have made and the marked-to-market value of our position in the stock.

Slippage PnL Our P&L has two components:

P&L due to slippage(t)=VWAP(t)*Slippage(t) For t = 0, . . . , T-1. The subscript goes only until T-1 because VWAP is calculated between two points (the last one between T-1

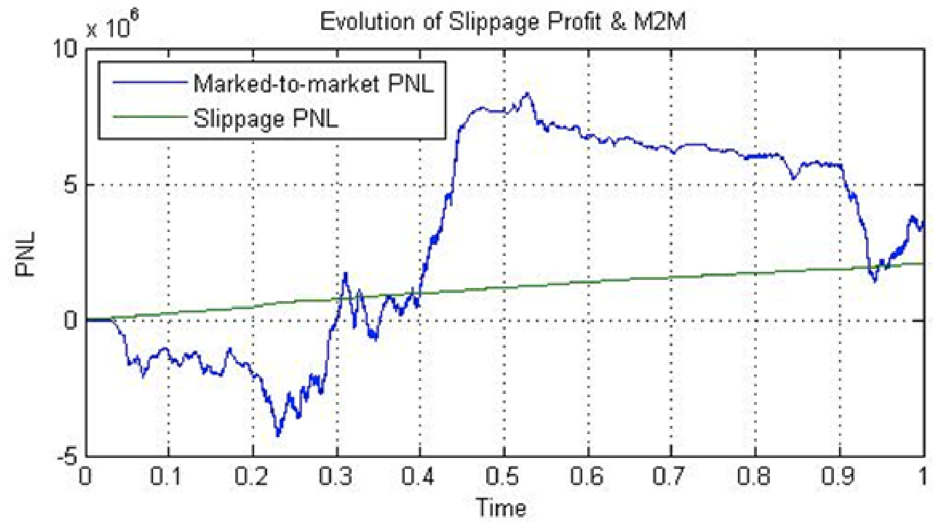


Figure 16: Evolution of PnL for constant u for Citi

and T.

The evolution of P&L slippage is dependent on our aggression level and on the stock spread. Stocks with low spread tend to generate lower slippage P&L (so the slippage P&L line will have lower slope than high-spread stocks). The green line in Fig. 16represents P&L slippage for Citi stock over 175 days while keeping our aggression level at 1. Citi is a stock with relatively high volumes and low spread. We found that this stock is particularly well-behaved and consistent in our simulation, probably due to the high number of high-frequency trading firms making market on this stock.

If we were to change the aggression level for Citi, the green line would be still mostly positive except for occasional kinks that are however invisible over longer periods. However, the P&L slippage in this case will be lower because trading at higher aggression levels will limit our profit. This is shown in Fig. 17 on the following page.

With constant u at 1, our P&L slippage for this simulation was \$2.2 million. With changing u, the profit is only \$1.1 million. In addition, stocks that require more aggressive trading because of their high volatility can have negative P&L slippage for extended periods of time. As a result, the final slippage P&L will be much lower than in the case with constant u at 1. This can be shown in Fig. 18 on the next page representing the slippage P&L for Bank of America (green line).

The slippage P&L in this case is only \$200,000 while if we keep u constant at 1,

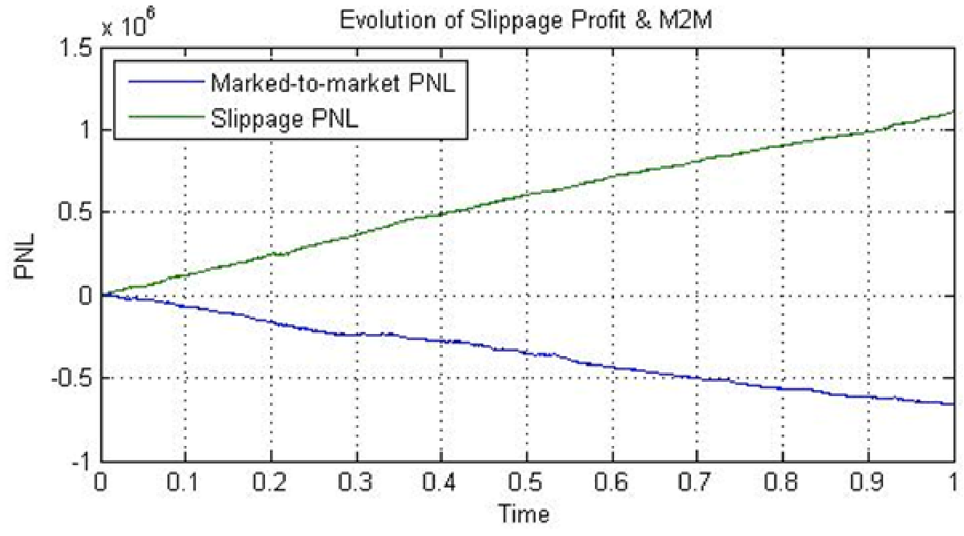


Figure 17: PnL Evolution for changing aggressiveness for Citi

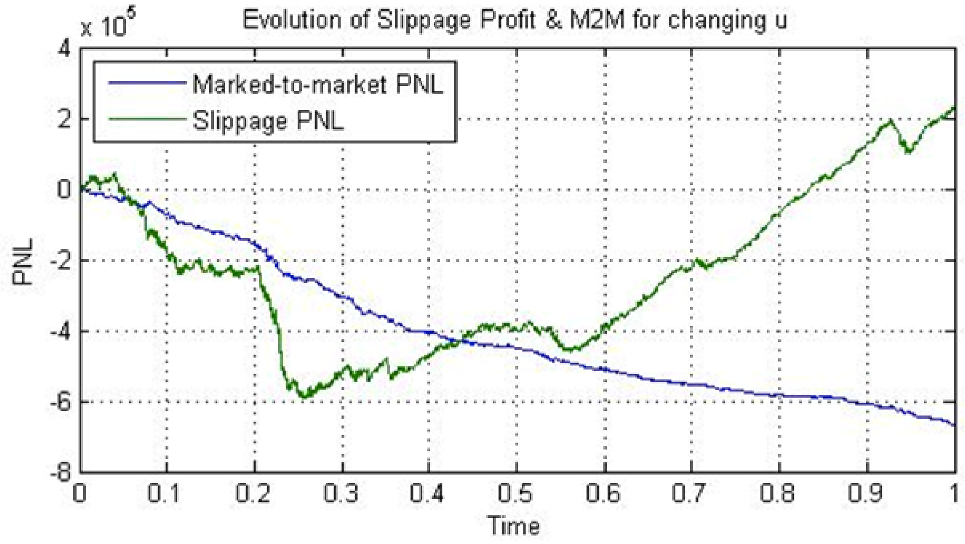


Figure 18: PnL Evolution of BAC for changing u

it would be \$10 million but our market risk would be huge because of our huge stock accumulation.

P&L marked-to-market This can be described by the following equation: marked-to-market P&L(t)=(P(t+1)/P(t) -1)*position(t)

For t = 0, . . . , T-1; P is the stock price; position is our exposure to the stock (can be positive or negative).

If u is kept constant at 1, this P&L component is much more volatile and depends on the price evolution. We would expect the marked-to-market P&L to be positive when the stock price is mean-reverting. This makes intuitive sense because we are market making and buying relatively low and selling relatively high is a good strategy for a mean-reverting stock. Citi is a good choice because high participation by high-frequency traders makes the price evolution more mean reverting (see Fig. 16 on page 23, blue line).

On the other hand, we would expect to lose money when trading a trending stock. The reason being the accumulation of large positions when the price trends. If the price keeps increasing (decreasing), we keep selling (buying) and lose a lot of money on the selling (buying) at the beginning of the trend.

If we are changing u, the marked-to-market P&L is much smoother but decreasing over long periods (see Fig. 17 on the previous page and 18 on the preceding page, blue line). It is smoother because we never accumulate large positions that cause the large swings in P&L. It is decreasing because when we start accumulating large position on the long side (short side), we become an aggressive seller (buyer) at that price. However, when the prices experience trends, we are selling into rising market and buying into a falling market, thus losing money.

Over short time periods marked-to-market PNL can be positive (see the following Fig. 19 on the next page for Citi for 1 day). This is because intraday prices show less momentum than long-term daily prices. However, over long-term trends, we still lose money when our position gets relatively large and we are forced to get aggressive.

Combining slippage P&L and marked-to-market P&L Combining the two components, we see an interesting and desirable result. The Fig. 20 on the following pageshows the results from a Monte Carlo simulation for Citi P&L when u is kept constant at 1 (Monte Carlo simulation is possible because the intraday prices in our simulation are generated by Brownian bridge). The slippage in this case is always positive (histogram at extreme left) but this is particular to Citi, for example Bank of America has also negative slippage P&L. Our marked-to-market P&L is approximately normally distributed around mean, as we would expect (histogram in the middle). Combined we obtain a bell curve that with mean above 0 thanks to the positive slippage (histogram at extreme right). Importantly, note the minimum and maximum values for our final P&L. The minimum is at negative \$15 million and the maximum at over \$20 million. This is a huge range, which makes this strategy unacceptably risky.

The range for our strategy with changing u looks very different. As Fig. 21 on page 27 shows (histogram at extreme right) the total P&L ranges from \$250,000 to \$550,000, that is less than 1% of the previous range. This is also the most important aspect of the changing u strategy and makes it much more suitable for trading than the constant u strategy. Also, not that for Citi, the simulation yields always positive slippage P&L and

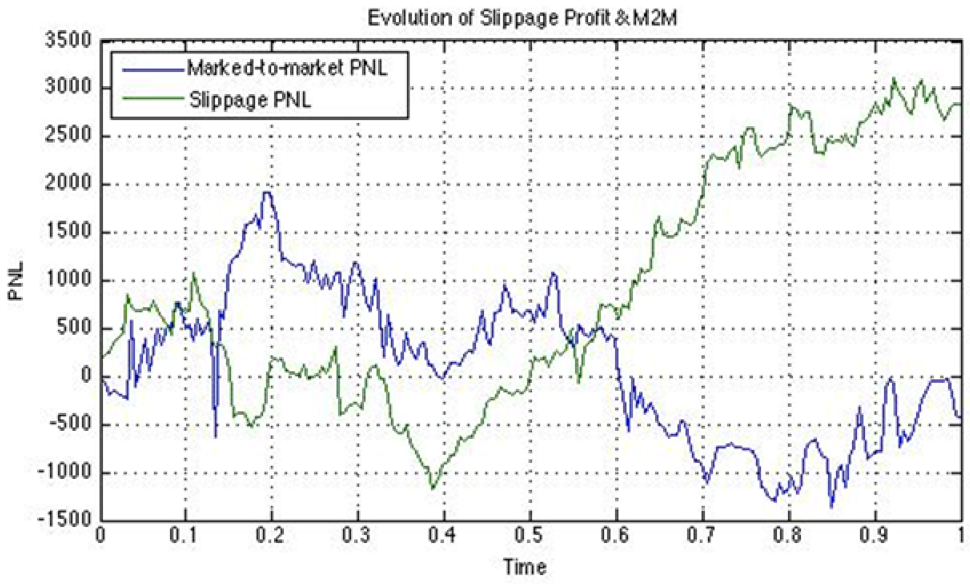


Figure 19: Citi PnL for 1 day

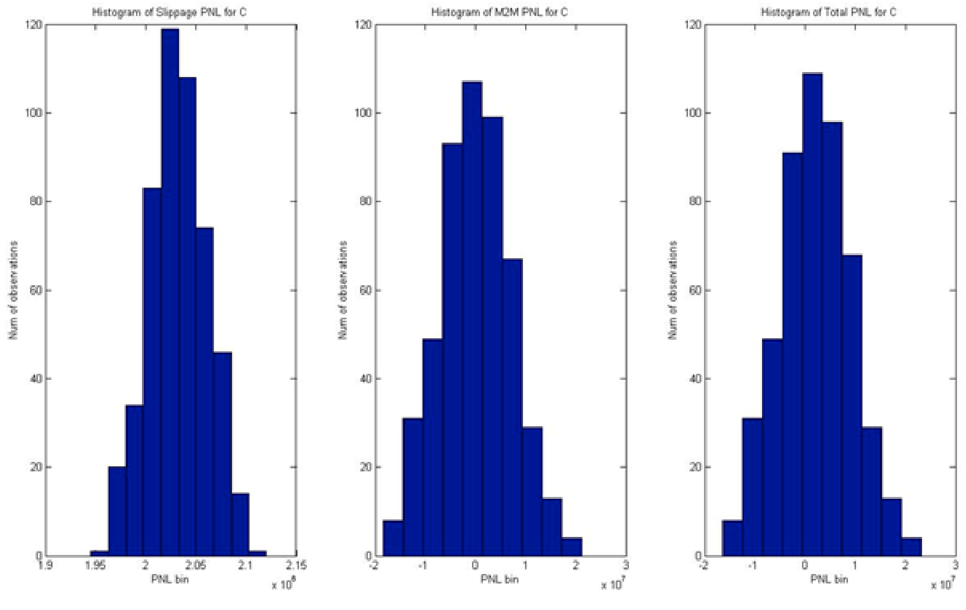


Figure 20: Simulation for a constant u

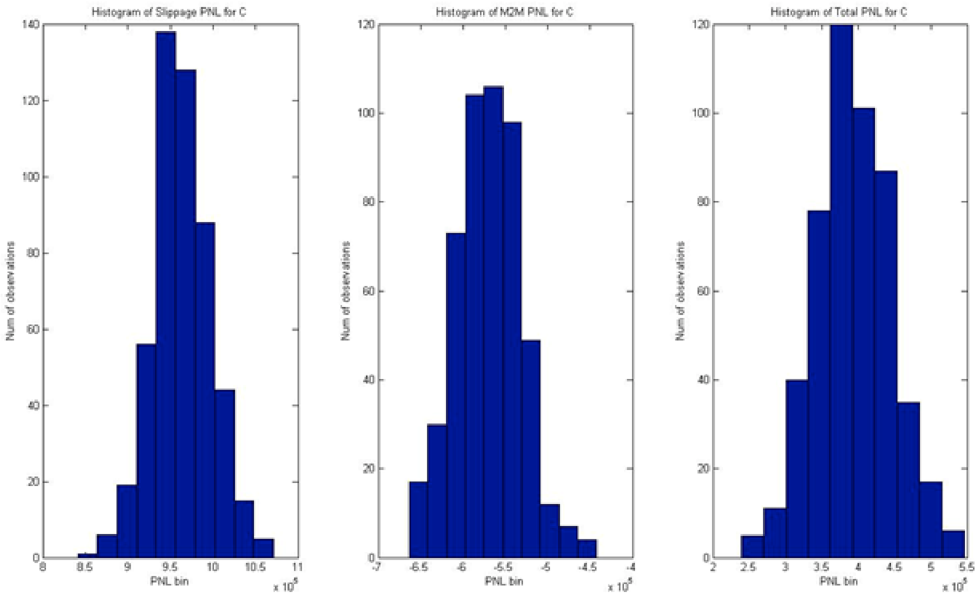


Figure 21: Simulation for changing aggressiveness

always negative marked-to-market P&L. While other stocks (such as Bank of America) always have negative marked-to-market P&L, they often have also negative slippage P&L. This means that even the strategy with changing u is risky and can lose money. The two most important determinants of the P&L are the spread and the volatility.

Multiple stocks Combining multiple stocks and change their u's at the same time based on the overall risk of the portfolio yields more volatile results than trading each stock individually and adding them up. We created a portfolio of Citi, Goldman Sachs, and Morgan Stanley. As Fig. 22 on the following page shows (histogram at extreme right), the range of portfolio P&L generated by a Monte Carlo simulation is between negative \$10 million and positive \$25 million.

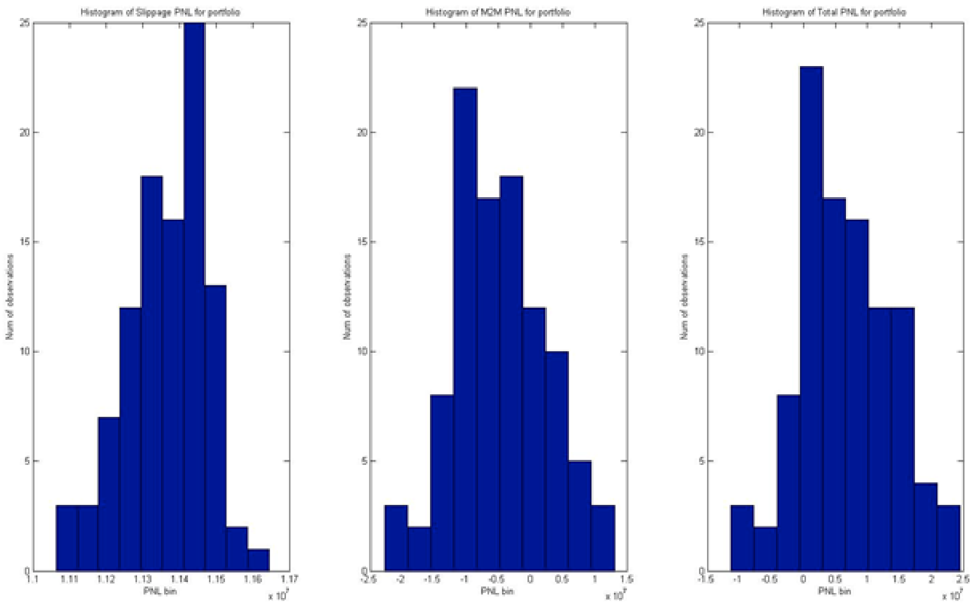


Figure 22: Simulation for Multiple stocks

7 Summary

Our main project goal is to utilize a black-box trading algorithm that can consistently beat the Volume Weighted Average Price by trading opportunistically on the quotes offered to the market on both the bid and the ask, and to capture the bid-ask spread. Our algorithm accumulates short positions in rising markets, and long positions in falling markets. In order to maximize our profit and loss, we strive to maximize our cash revenues from VWAP slippage, while also mitigating market risk, to protect ourselves from marked-to-market losses.

The first goal of our project was to model the drivers of VWAP slippage. With accurate models of the VWAP slippage, we would be able to successfully simulate the cash revenues generated. The trading data provided to us demonstrated that slippage could be driven by a variety of factors: from the percentage of trading volume over the trading period, to the bid-ask spread of the stock that was traded, to the aggression level of the black box.

What we discovered was that most of the variation in VWAP slippage was directly explained by an inverse linear relationship with the bid-ask spread. As spreads widened, a market-making strategy employing the VWAP-beating black box algorithm generated greater revenues. This intuitively makes sense, since a market-making strategy should be able to perform better when spreads widen, since needed liquidity is being provided to the markets.

After modeling the VWAP slippage, we were able to simulate the market-making strategy according to actual changes in price. After back testing several simulations on stocks which actually exist, with their historic volatilities and bid-ask spreads, we found that the market-making strategy generates positive cash revenues, but our profit and loss would be largely determined by the marked-to-market value of our inventory in the stock. Since we are net sellers into rising markets, and net buyers into falling markets, this usually leads to a negative mark-to-market P&L.

We managed the large market risks that we take from large inventories by an optimization process that changes the aggression level on either the bid or the ask based upon inventory accumulation. This allows us to manage market risk, while still trading opportunistically enough to generate worthwhile trading revenues. By expanding our trading algorithm to multiple stocks, we found that due to diversification, market risk can be more cheaply mitigated. When inventories accumulate in highly correlated stocks, the price of unloading this market risk becomes cheaper. When we expanded the simulation to include 3 stocks, we found that the P&L outperformed the market making strategy on any one stock significantly.

Our results demonstrated that the key types of stocks that a market-making strategy works well on are stocks with a small spread and high liquidity. Stocks which are heavily traded, such as Citigroup, did exceedingly well in the simulation, while less liquid stocks like GS did poorly. However, when a universe of stocks is included in the market-making portfolio, we see P&L's outperform any one stock individually. This is because due to diversification, the cost of unloading the market risk becomes cheaper with a large portfolio of stocks.

Appendix: Code

All the code that constitutes our simulation of the algorithm has been written in MatLab. There are quite a few files, and they are commented in such a fashion that allows a reader to understand what the function does.

The way the code is set up one can run the simulation in one of two ways: either by running the main_single_stock file that will run the simulation for 1 stock, or by running the main_multiple_stocks file that will run the simulation for several stocks.

main_single_stock.m

```
% The main
% This file allows us to run our simulator using one stock, and if we
% want to run an optimization to find optimal bounds we simply have to
% enter the desired values of the bounds on lines 62 and 63
%
% There are 5 parts to the main function:
% 1. inputs: keep 390 as the number of points generated per day
% (corresponding to minutes); select your desired number of days
% 2. initialization: prepares all the empty vectors
% 3. prices & volumes (this is the core):
% since we generate the numbers (volumes and prices) for each day
% separately, we need to create a continuous price, volume, and VWAP
% evolution over n days.
% 4. PNL: see PNL.m
% 5. Graphs and save values in Excel files
% ------------------------------

%% INPUTS
clc;
N = 390;                    % 390 for one day
interval = 2;               % in terms of minutes
numDays = 175;              % number of days in the model
horizon = 'minutes';

% select stock to consider, it can be chosen from the list in
% Spreads.xls, these are the stocks for which we have spread data
% Then get the spread of the corresponding stock
tickers = {'MS'};
spread = getSpreads(tickers);

numPointsPerDay = ceil(N/interval)+1;    % number of points in a day
numPoints = numDays * numPointsPerDay;
stockPrices = [];
stockEndPrices = [];
Vbought_Out = [];
Vsold_Out = [];
Vmarket_Out = [];
Vdifference_Out = [];
VWAP_Out = [];
portfolio = 0;
uBuy_Out = [];
uSell_Out = [];
graph_total = 1;
```

```
% if only one stock, we don't graph the sum of the portfolio
if length(tickers) == 1
    graph_total = 0;
end

%% GENERATE PRICES. Change here manually for each stock
% generates 34,301 price points for each of our 4 stocks. This corresponds
% to 196 points for 175 days + 1 point at time = 0
[prices,volumesTraded,dailyPrices] = intraday_prices(N,interval,tickers);
dailySD = computeSD(dailyPrices);

price1 = prices;
volume1 = volumesTraded;
numStocks = size(price1,2);

%different values of bounds for which we want to run our simulation for
%lower and upper bound cannot have a value in common, or code crashes
minModifyU = 2000;%[400 800 1000 1500 2000 4000 6000 9000];
maxModifyU = 30000;%[10000 20000 30000 40000 60000 90000 120000 150000
200000];

CashValues = zeros(length(minModifyU),length(maxModifyU));
LossesValues = zeros(length(minModifyU),length(maxModifyU));
PNLValues = zeros(length(minModifyU),length(maxModifyU));

pseudoMC = 1; %number of times we run our simulation for each pair of
bounds
finalCash = zeros(pseudoMC,1);
finalLosses = zeros(pseudoMC,1);
finalPNL = zeros(pseudoMC,1);

for minIndex = 1:length(minModifyU)
    minIndex
    for maxIndex = 1:length(maxModifyU)
        maxIndex
        for j=1:pseudoMC %pseudo MC loop
            % PRICES & VOLUMES
            for modifyUorNot = 1:1
                % if 1 we modify the u's, otherwise it stays equal to 1
                Vbought_Out = [];
                Vsold_Out = [];
                Vmarket_Out = [];
                Vdifference_Out = [];
                VWAP_Out = [];
                portfolio = 0;
                uBuy_Out = [];
                uSell_Out = [];
                start = 1;
                portfolioEvo = zeros(numDays,1);

                stockPrices = [];
                stockEndPrices = [];
                Vbought_Out = [];
                Vsold_Out = [];
                Vmarket_Out = [];
                Vdifference_Out = [];
                VWAP_Out = [];
                portfolio = 0;
                uBuy_Out = [];
                uSell_Out = [];

                for i=1:numDays
                    currentSD = dailySD(i);

                    if modifyUorNot == 1
                        prices = price1(start:start+numPointsPerDay-1,:);
% generates 196 points for each day
                        stockPrices = [stockPrices; prices];
% appends the 196 to the previous days
                        stockEndPrices = [stockEndPrices; prices(2:end,:)];
% appends 195 prices. This will be used with 195 of volume
                        volumesTraded =
volume1(start:start+numPointsPerDay-2,:);
                    else
                        prices = price1(start:start+numPointsPerDay-1,:);
                        volumesTraded =
volume1(start:start+numPointsPerDay-2,:);
                    end

                    % move by one day into the future
                    start = start + numPointsPerDay - 1;

                    % Vx is 195 points because at t0 we assume volume = 0.
The vector
                    % is of the size 195x1
                    [Vdifference,Vbought,Vsold,Vmarket,uBuy,uSell] =
tradedVolumes(interval,horizon,prices,
portfolio,modifyUorNot,numStocks,volumesTraded,minModifyU(minIndex),maxModi
fyU(maxIndex),currentSD);
                    sumVdiff = sum(Vdifference,1);
                    sumVdiff = sum(sumVdiff,2);
                    portfolio = portfolio + sumVdiff;
                    uBuy_Out = [uBuy_Out; uBuy];
                    uSell_Out = [uSell_Out; uSell];
                    Vbought_Out = [Vbought_Out; Vbought];
                    Vsold_Out = [Vsold_Out; Vsold];
                    Vmarket_Out = [Vmarket_Out; Vmarket];
                    Vdifference_Out = [Vdifference_Out; Vdifference];

                    % for generating VWAP, we need the sizes of price
vector and
                    % Vmarket vector to be the same. For a two minute
interval, the
                    % size should be 196 points
                    Vmarket = [zeros(1,numStocks);Vmarket];

                    VWAP = zeros(size(volumesTraded));
                    for k = 1:numStocks
                        VWAP(:,k) = VWAP_Function(prices, Vmarket);
                    end
                    VWAP_Out = [VWAP_Out; VWAP];

                    % next day, the starting price is the last price of
yesteday
                    initialPrice = stockPrices(end);
                end

                [uBuyValues,uBuyOccurences] = count_unique(uBuy_Out(:,1))
                [uSellValues,uSellOccurences] =
count_unique(uSell_Out(:,1))

                %% PNL & EVOLUTION
                total_PNL = [];
                M2M_PNL = [];
                cash_made = [];

                for i = 1:numStocks
                    [total_PNL_new, M2M_PNL_new, cash_made_new] =
PNL_Function(stockEndPrices(:,i), VWAP_Out(:,i), Vbought_Out(:,i),
Vsold_Out(:,i), uBuy_Out(:,i), uSell_Out(:,i),spread(i));
                    total_PNL = [total_PNL,total_PNL_new];
                    M2M_PNL = [M2M_PNL,M2M_PNL_new];
                    cash_made = [cash_made,cash_made_new];
                end

                cashmadeMean = mean(cash_made,1);

                allStocks_total_PNL = sum(total_PNL,2);
                allStocks_M2M_PNL = sum(M2M_PNL,2);
                allStocks_cash_made = sum(cash_made,2);

                %% GRAPH
                % stockEndPrices has 195 rows
                l = size(stockEndPrices,1);
                t = (0:1:l-1)/(l-1);
                t = t*sqrt(1);

                for jj=1:numStocks
                    figure
                    ticker = tickers{jj};
                    subplot(2,2,1); plot(t,stockEndPrices(:,jj));
                    legend(num2str(ticker))
                    title(['Geometric Brownian motions over '
num2str(numDays) ' days.'])
                    xlabel('Time')
                    ylabel('Price')
                    grid on

                    subplot(2,2,2); plot(t,cumsum(Vdifference_Out(:,jj)));
                    legend('Difference','Location','NorthWest')
                    title(['Evolution of Difference over ' num2str(numDays)
' days.'])
                    xlabel('Time')
                    ylabel('dif')
                    grid on

                    subplot(2,2,3);
                    plot(t,M2M_PNL(:,jj),t,cash_made(:,jj));
                    legend('Marked-to-market PNL','Slippage
PNL','Location','NorthWest');
                    title('Evolution of Slippage Profit & M2M');
                    xlabel('Time');
                    ylabel('PNL');
                    grid on

                    subplot(2,2,4);
                    plot(t,total_PNL(:,jj));
                    legend('total PNL','Location','NorthWest');
                    title('Evolution of total PNL');
                    xlabel('Time');
                    ylabel('PNL');
                    grid on
                end

                %% plotting Cash & Marked-to-Market PNL
                if graph_total == 1

                    figure
                    % ploting the portfolio difference
                    subplot(3,1,1);
plot(t,cumsum(sum(Vdifference_Out(:,j),2)));
                    legend('Difference')
                    title(['Evolution of Difference for GS, MS, C, WFC, and
JPM over ' ...
                        num2str(numDays) ' days.'])
                    xlabel('Time')
                    ylabel('dif')
                    grid on;

                    % ploting components of PNL
                    subplot(3,1,2);
                    plot(t,allStocks_M2M_PNL,t,allStocks_cash_made);
                    legend('Marked-to-market PNL','Slippage PNL');
                    title('Evolution of Slippage Profit & M2M');
                    xlabel('Time');
                    ylabel('PNL');
                    grid on;

                    % ploting total PNL
                    subplot(3,1,3);
                    plot(t,allStocks_total_PNL);
                    legend('total PNL');
                    title('Evolution of total PNL');
                    xlabel('Time');
                    ylabel('PNL');
                    grid on;
                end
                finalCash(j) = cash_made(end);
                finalLosses(j) = M2M_PNL(end);
                finalPNL(j) = total_PNL(end);
            end
        end
        CashValues(minIndex,maxIndex) = mean(finalCash);
        LossesValues(minIndex,maxIndex) = mean(finalLosses);
        PNLValues(minIndex,maxIndex) = mean(finalPNL);
    end
end

CashValues
LossesValues
PNLValues

% xlswrite('CashValuesMatrix',CashValues);
% xlswrite('LossesValuesMatrix',LossesValues);
% xlswrite('PNLValuesMatrix',PNLValues);
```

main_multiple_stocks.m

```
% The main
% This file allows us to run our simulator using several stocks, and if we
% want to run an optimization to find optimal bounds we simply have to
% enter the desired values of the bounds on lines 62 and 63
%
% There are 5 parts to the main function:
% 1. inputs: keep 390 as the number of points generated per day
% (corresponding to minutes); select your desired number of days
% 2. initialization: prepares all the empty vectors
% 3. prices & volumes (this is the core):
% since we generate the numbers (volumes and prices) for each day
% separately, we need to create a continuous price, volume, and VWAP
% evolution over n days.
% 4. PNL: see PNL.m
% 5. Graphs and save values in Excel files
% ------------------------------

%% INPUTS
clc;
N = 390;                % 390 for one day
interval = 1;           % in terms of minutes
numDays = 100;          % number of days in the model
horizon = 'minutes';

% select stocks to consider, they can be chosen from the list in
% Spreads.xls, these are the stocks for which we have spread data
% Then get the spreads of the corresponding stocks
tickers = {'GS','MS','C'};
spread = getSpreads(tickers);

numPointsPerDay = ceil(N/interval)+1;   % number of points in a day
numPoints = numDays * numPointsPerDay;
stockPrices = [];
stockEndPrices = [];
Vbought_Out = [];
Vsold_Out = [];
Vmarket_Out = [];
Vdifference_Out = [];
VWAP_Out = [];
portfolio = 0;
uBuy_Out = [];
uSell_Out = [];
graph_total = 1;

% if only one stock, we don't graph the sum of the portfolio
if length(tickers) == 1
    graph_total = 0;
end

%% GENERATE PRICES. Change here manually for each stock
% generates 34,301 price points for each of our 4 stocks. This corresponds
% to 196 points for 175 days + 1 point at time = 0
[prices,volumesTraded,dailyPrices] = intraday_prices(N,interval,tickers);
dailySD = computeCov(dailyPrices);

price1 = prices;
volume1 = repmat(sum(volumesTraded,2)/length(tickers),1,length(tickers));
numStocks = size(price1,2);
```

```
%different values of bounds for which we want to run our simulation for
%lower and upper bound cannot have a value in common, or code crashes
minModifyU = 1000;%[200 400 800 1000 1500 2000 3999 5999 7999 11999];
maxModifyU = 80000;%[5000 10000 15000 20000 30000 40000 50000];

CashValues = zeros(length(minModifyU),length(maxModifyU));
LossesValues = zeros(length(minModifyU),length(maxModifyU));
PNLValues = zeros(length(minModifyU),length(maxModifyU));

pseudoMC = 1; %number of times we run our simulation for each pair of
bounds
finalCash = zeros(pseudoMC,1);
finalLosses = zeros(pseudoMC,1);
finalPNL = zeros(pseudoMC,1);

for minIndex = 1:length(minModifyU)
    minIndex
    for maxIndex = 1:length(maxModifyU)
        maxIndex
        for j=1:pseudoMC %pseudo MC loop
            % PRICES & VOLUMES
            for modifyUorNot = 1:1
                % if 1 we modify the u's, otherwise it stays equal to 1
                Vbought_Out = [];
                Vsold_Out = [];
                Vmarket_Out = [];
                Vdifference_Out = [];
                VWAP_Out = [];
                portfolio = 0;
                uBuy_Out = [];
                uSell_Out = [];
                start = 1;
                portfolioEvo = zeros(numDays,1);

                stockPrices = [];
                stockEndPrices = [];
                Vbought_Out = [];
                Vsold_Out = [];
                Vmarket_Out = [];
                Vdifference_Out = [];
                VWAP_Out = [];
                portfolio = zeros(numStocks,1);
                uBuy_Out = [];
                uSell_Out = [];

                for i=1:numDays
                    currentSD = dailySD(:,:,i);

                    if modifyUorNot == 1
                        prices = price1(start:start+numPointsPerDay-1,:);
% generates 196 points for each day
                        stockPrices = [stockPrices; prices];
% appends the 196 to the previous days
                        stockEndPrices = [stockEndPrices; prices(2:end,:)];
% appends 195 prices. This will be used with 195 of volume
                        volumesTraded =
volume1(start:start+numPointsPerDay-2,:);
                    else
                        prices = price1(start:start+numPointsPerDay-1,:);
                        volumesTraded = 
volume1(start:start+numPointsPerDay-2,:);
                    end
                    
                    % move by one day into the future
                    start = start + numPointsPerDay - 1;
                    
                    % Vx is 195 points because at t0 we assume volume = 0. 
The vector
                    % is of the size 195x1
                    [Vdifference,Vbought,Vsold,Vmarket,uBuy,uSell] = 
tradedVolumesMultiple(interval,horizon,prices, 
portfolio,modifyUorNot,numStocks,volumesTraded,minModifyU(minIndex),maxModi
fyU(maxIndex),currentSD);
                    sumVdiff = sum(Vdifference,1);
                    sumVdiff = sum(sumVdiff,2);
                    
                    VdiffTot = sum(Vdifference,1);
                    portfolio = portfolio + VdiffTot';
                    uBuy_Out = [uBuy_Out; uBuy];
                    uSell_Out = [uSell_Out; uSell];
                    Vbought_Out = [Vbought_Out; Vbought];
                    Vsold_Out = [Vsold_Out; Vsold];
                    Vmarket_Out = [Vmarket_Out; Vmarket];
                    Vdifference_Out = [Vdifference_Out; Vdifference];
                    
                    % for generating VWAP, we need the sizes of price 
vector and
                    % Vmarket vector to be the same. For a two minute 
interval, the
                    % size should be 196 points
                    Vmarket = [zeros(1,numStocks);Vmarket];
                    
                    VWAP = zeros(size(volumesTraded));
                    for k = 1:numStocks
                        VWAP(:,k) = VWAP_Function(prices, Vmarket);
                    end
                    VWAP_Out = [VWAP_Out; VWAP];
                    
                    % next day, the starting price is the last price of 
yesterday
                    initialPrice = stockPrices(end);
                end
                
                %displays which values of aggressiveness occur, and how 
many times
                [uBuyValues,uBuyOccurences] = count_unique(uBuy_Out(:,1));
                [uSellValues,uSellOccurences] = 
count_unique(uSell_Out(:,1));
                
                %% PNL & EVOLUTION
                total_PNL = [];
                M2M_PNL = [];
                cash_made = [];
                
                for i = 1:numStocks
                    [total_PNL_new, M2M_PNL_new, cash_made_new] = 
PNL_Function(stockEndPrices(:,i), VWAP_Out(:,i), Vbought_Out(:,i), 
Vsold_Out(:,i), uBuy_Out(:,i), uSell_Out(:,i),spread(i));
                    total_PNL = [total_PNL,total_PNL_new];
                    M2M_PNL = [M2M_PNL,M2M_PNL_new];
                    cash_made = [cash_made,cash_made_new];
                end

                allStocks_total_PNL = sum(total_PNL,2);
                allStocks_M2M_PNL = sum(M2M_PNL,2);
                allStocks_cash_made = sum(cash_made,2);

                %% GRAPH
                % stockEndPrices has 195 rows
                l = size(stockEndPrices,1);
                t = (0:1:l-1)/(l-1);
                t = t*sqrt(1);

                for jj=1:numStocks
                    figure
                    ticker = tickers{jj};
                    subplot(2,2,1); plot(t,stockEndPrices(:,jj));
                    legend(num2str(ticker))
                    title(['Geometric Brownian motions over '
num2str(numDays) ' days.'])
                    xlabel('Time')
                    ylabel('Price')
                    grid on

                    subplot(2,2,2); plot(t,cumsum(Vdifference_Out(:,jj)));
                    legend('Difference','Location','NorthWest')
                    title(['Evolution of Difference over ' num2str(numDays)
' days.'])
                    xlabel('Time')
                    ylabel('dif')
                    grid on

                    subplot(2,2,3);
                    plot(t,M2M_PNL(:,jj),t,cash_made(:,jj));
                    legend('Marked-to-market PNL','Slippage
PNL','Location','NorthWest');
                    title('Evolution of Slippage Profit & M2M');
                    xlabel('Time');
                    ylabel('PNL');
                    grid on

                    subplot(2,2,4);
                    plot(t,total_PNL(:,jj));
                    legend('total PNL','Location','NorthWest');
                    title('Evolution of total PNL');
                    xlabel('Time');
                    ylabel('PNL');
                    grid on
                end

                %% plotting Cash & Marked-to-Market PNL
                if graph_total == 1
                    figure
                    % ploting the portfolio difference
                    subplot(3,1,1);
plot(t,cumsum(sum(Vdifference_Out(:,j),2)));
                    legend('Difference')
                    title(['Evolution of Difference for GS, MS, C over'
num2str(numDays) ' days.'])
```

```
                    xlabel('Time')
                    ylabel('dif')
                    grid on;

                    % ploting components of PNL
                    subplot(3,1,2);
                    plot(t,allStocks_M2M_PNL,t,allStocks_cash_made);
                    legend('Marked-to-market PNL','Slippage PNL');
                    title('Evolution of Slippage Profit & M2M');
                    xlabel('Time');
                    ylabel('PNL');
                    grid on;

                    % ploting total PNL
                    subplot(3,1,3);
                    plot(t,allStocks_total_PNL);
                    legend('total PNL');
                    title('Evolution of total PNL');
                    xlabel('Time');
                    ylabel('PNL');
                    grid on;
                end

                %compute final values of variables of interest
                finalCash(j) = cash_made(end);
                finalLosses(j) = M2M_PNL(end);
                finalPNL(j) = total_PNL(end);
            end
        end
        %take average of the variables over the several simulations
        CashValues(minIndex,maxIndex) = mean(finalCash);
        LossesValues(minIndex,maxIndex) = mean(finalLosses);
        PNLValues(minIndex,maxIndex) = mean(finalPNL);
    end
end

CashValues
LossesValues
PNLValues

% %save values in Excel files
% xlswrite('CashValuesMatrixMultiple',CashValues);
% xlswrite('LossesValuesMatrixMultiple',LossesValues);
% xlswrite('PNLValuesMatrixMultiple',PNLValues);
```

<u>intraday_prices.m</u>

```
%%%%%%%%%%%%%%%%%%%%%%%%
% The code below first obtains the daily prices and volume of the stocks
% listed for the days specified and then interpolates the intraday path
% based on the BM between the two consecutive closing prices.
%%%%%%%%%%%%%%%%%%%%%%%%

function [price,volume,dailyPrices] =
intraday_prices(NumMinDay,interval,tics)
```

```
% Getting the daily data
%%%%tics = {'C' 'GE' 'ROP' 'WMT'};
%tics = {'C','GS','MS','BAC'};
% BAC:high spread and volume, GE: high volume, ROP: low volume, WMT: low
spread
startdt = '01-23-2012';    % in  format
enddt = '09-30-2012';      % in  format
StocksData = hist_stock_data(startdt,enddt,tics);

% Specifying simulation parameters
% interval = 5 ;  % this means interval is 2 minutes and there are 390
trading minutes in a day
% NumMinDay = 390;

%%%%% ----------------------------------------------------
% Computation of number of days
if(mod(NumMinDay,interval)~=0)    % needs to be a factor of 390;
    error('Error: The number of trading intervals should be a whole number
for a day. Check the interval size');
end
temp = zeros(1,length(tics));
for j = 1:length(tics)
    temp(j) = length(StocksData(j).Date);
end
if(all(temp~=temp(1)))
    error('Error: not all stocks have data on the dates chosen. Choose
different dates');
else
    NumDays = temp(1);
end
NumIntervalsDay = NumMinDay/interval+1;
DaysIndex = 1+NumIntervalsDay:NumIntervalsDay:(NumDays+1)*NumIntervalsDay;

% creating the arrays
price = zeros(NumDays*NumIntervalsDay,length(tics));
volume = zeros(NumDays*NumIntervalsDay,length(tics));
ADV = zeros(NumDays,length(tics));

%creating the data for each interval
for i = 1:length(tics)
    price(1,i) = StocksData(i).Open(1);
    price(DaysIndex,i) = StocksData(i).AdjClose;
    ADV(:,i) = StocksData(i).Volume;
end

dailyPrices = StocksData(1).AdjClose;
dailyPrices = dailyPrices(end:-1:1);

% Creating intraday prices using BM. The factor is used to reduce std if
% the prices are too small
for i = 1:NumDays
    EndDayPrice = price(1+NumIntervalsDay*i,:);
    for j = 2:NumIntervalsDay
        factor = ones(1,length(tics));
        PastPrice = price( (i-1)*NumIntervalsDay+(j-1) ,:);
        PriceMean = ((NumIntervalsDay - j).*PastPrice + EndDayPrice )./
(NumIntervalsDay-j+1);
        factor = 10*NumIntervalsDay./PastPrice;
        PriceStd = (NumIntervalsDay - j)/(NumIntervalsDay-j+1)./factor;
        price( (i-1)*NumIntervalsDay + j,:) = normrnd(PriceMean,PriceStd);
    end
    volume( (i-1)*NumIntervalsDay+1:i*NumIntervalsDay,:) =
repmat(ADV(i,:)./NumIntervalsDay,NumIntervalsDay,1);
end

price = price(2:end,:); % remove the first value? Why is there +1 point?
price = price(end:-1:1,:);
volume = volume(end:-1:1,:);
```

hist_stock_data.m

```
function stocks = hist_stock_data(start_date, end_date, tickers,varargin)

stocks = struct([]);        % initialize data structure

% split up beginning date into day, month, and year.  The month is
% subracted is subtracted by 1 since that is the format that Yahoo! uses
bd = start_date(4:5);       % beginning day
bm = sprintf('%02d',str2double(start_date(1:2))-1); % beginning month
by = start_date(7:10);       % beginning year

% split up ending date into day, month, and year.  The month is subracted
% by 1 since that is the format that Yahoo! uses
ed = end_date(4:5);         % ending day
em = sprintf('%02d',str2double(end_date(1:2))-1);   % ending month
ey = end_date(7:10);         % ending year

% determine if user specified frequency
temp = find(strcmp(varargin,'frequency') == 1); % search for frequency
if isempty(temp)                                % if not given
    freq = 'd';                                 % default is daily
else                                            % if user supplies frequency
    freq = varargin{temp+1};                    % assign to user input
    varargin(temp:temp+1) = [];                 % remove from varargin
end
clear temp

h = waitbar(0, 'Please Wait...');               % create waitbar
idx = 1;                                        % idx for current stock data

% cycle through each ticker symbol and retrieve historical data
for i = 1:length(tickers)

    % update waitbar to display current ticker
    waitbar((i-1)/length(tickers),h,sprintf('%s %s %s%0.2f%s', ...
       'Retrieving stock data for',tickers{i},'(',(i-
1)*100/length(tickers),'%)'))

% download historical data using the Yahoo! Finance website
    [temp, status] =
urlread(strcat('http://ichart.finance.yahoo.com/table.csv?s='...
        ,tickers{i},'&a=',bm,'&b=',bd,'&c=',by,'&d=',em,'&e=',ed,'&f=',...
        ey,'&g=',freq,'&ignore=.csv'));

    if status
        % organize data by using the comma delimiter
        [date, op, high, low, cl, volume, adj_close] = ...
            strread(temp(43:end),'%s%s%s%s%s%s%s','delimiter',',');
        stocks(idx).Ticker = tickers{i};        % obtain ticker symbol
        stocks(idx).Date = date;                % save date data
        stocks(idx).Open = str2double(op);      % save opening price data
        stocks(idx).High = str2double(high);    % save high price data
        stocks(idx).Low = str2double(low);      % save low price data
        stocks(idx).Close = str2double(cl);     % save closing price data
        stocks(idx).Volume = str2double(volume);      % save volume data
        stocks(idx).AdjClose = str2double(adj_close); % save adjustied
close data

        idx = idx + 1;                          % increment stock index
    end

    % clear variables made in for loop for next iteration
    clear date op high low cl volume adj_close temp status

    % update waitbar
    waitbar(i/length(tickers),h)
end

close(h)    % close waitbar
```

<u>PNL_Function.m</u>

```
% needs a price vector of size n
% bought, sold vectors of size n
% VWAP of size n
% uBid and uAsk are scalars
%
% We separate the PNL into two parts:
% 1. what we have made on edge: edge from buying + edge from selling
% 2. market-to-market PNL which is computed as follows:
% - the difference between the cumulative cost of buying stocks and the
% market value
% - the difference between the cumulative "cost" of selling stocks and the
% market value
%
% At the end, we plot separately the cash we made and the market PNL
% note that the cash is much bigger than the sum of our buying & selling
% PNL
% -----------------------------------------

function [total_PNL, M2M_PNL, cash_made] = PNL_Function(prices, VWAP,
bought, sold, uBid, uAsk, spread)
%% rebuilding the slippage for buy and sell based on the direction of the
% market. We are assuming that the percentage change of the default values
% for raising/falling market is the same for each u

n = length(bought);
slippageBid = zeros(n,1);           % must have the same number of elements
as there is number of trades
slippageAsk = zeros(n,1);

for i=1:(n-1)
    price_change = prices(i+1)/prices(i)-1;
    if (price_change>=0)
        slippageBid(i) = generate_VWAP(spread,1,uBid(i));
        slippageAsk(i) = generate_VWAP(spread,1,uAsk(i));
    else
        slippageBid(i) = generate_VWAP(spread,-1,uBid(i));
        slippageAsk(i) = generate_VWAP(spread,-1,uAsk(i));
    end
end

%% PNL calculation
cashSavingB = cumsum((VWAP.*slippageBid).*bought);
cashSavingS = cumsum((VWAP.*slippageAsk).*sold);
cash_made = cashSavingB + cashSavingS;

netExposure = bought - sold;
cumNetExposure = cumsum(netExposure);
purchaseCost = VWAP.*(bought - sold);
cumPurchaseCost = cumsum(purchaseCost);
marketValue = prices.*cumNetExposure;
M2M_PNL = marketValue - cumPurchaseCost;

total_PNL = M2M_PNL + cash_made;
```

tradedVolumes.m

```
function [Vdifference,Vbought,Vsold,Vmarket,uBuy,uSell] =
tradedVolumes(n,horizon,prices,portfolio,modifyUorNot,numStocks,volumesTrad
ed,minOpt,maxOpt,currentSD)
%%%%%INPUTS
%n is the interval on which each volume is computed (eg n=2 the day is
divided in 195 2-minute intervals)
%prices is the vector of prices based on the same intervals as the ones the
volumes are computed
%portfolio is the number of shares of the portfolio we are currently
holding (positive or negative)
%modifyUorNot indicates whether we change aggressiveness or not (1 yes,2
no)
%numStocks is the number of stocks in the portfolio]
%volumesTraded is the number of stocks traded in each interval for each
stock
%minOpt and maxOpt correspond to the values of the bounds for which
aggressiveness will change
%currentSD is the volatility of the stock considered for that day
%%%%%OUTPUTS
%Vmarket: vector containing the number of shares traded in the whole market
on a single stock during the intervals we are interested in
%Vbought, Vsold: vectors containing the number of shares we bought and sold
during the intervals
%Vdifference: vector being the difference between the number of stocks
bought and sold, during each interval
%uBuy and uSell: vectors representing the values of u (buy and sell) during
each of the intervals throughout the day

if strcmp(horizon,'days')
    split = 1;
elseif strcmp(horizon,'minutes')
    split = 6.5*60;
elseif strcmp(horizon,'seconds')
    split = 6.5*60*60;
end
```

```
[mu,sigma,lambda] = getVolumeInputs();

%in case we cannot fit an integer number of n in our split we compute the
length of the last interval (rest)
%eg for 390 minutes, if n = 45 minutes we have 8 intervals of 45 minutes
and one of only 30 minutes
divider = split/n;
intervals = floor(divider);
rest = divider - intervals;

mu_vector = volumesTraded;
sigma_vector = repmat(sigma*sqrt(n),intervals,numStocks);

% % if rest ~= 0
% %     intervals = intervals + 1;
% %     mu_vector = [mu_vector; mu*rest*n];
% %     sigma_vector = [sigma_vector;sigma*sqrt(rest*n)];
% % end

Vmarket = mu_vector.*normrnd(1,sigma_vector);

returns = zeros(intervals,numStocks);

for i=1:intervals
    returns(i,:)=prices(i+1,:)/prices(i,:)-ones(1,numStocks); %value of
0.02 means a return of 2%
end

Vbought = zeros(intervals,numStocks);
Vsold = zeros(intervals,numStocks);
uBuy = zeros(intervals,numStocks);
uSell = zeros(intervals,numStocks);

for i = 1:intervals
    if modifyUorNot == 1
        [uBuy(i,:),uSell(i,:)] =
modifyU(portfolio,numStocks,minOpt,maxOpt,currentSD);
    else
        uBuy(i,:) = ones(1,numStocks);
        uSell(i,:) = ones(1,numStocks);
    end
    for j = 1:numStocks
        Vbought(i,j) =
0.01*generate_POV(returns(i,j),1,uBuy(i,j))*Vmarket(i,j);
        Vsold(i,j) = 0.01*generate_POV(returns(i,j),-
1,uSell(i,j))*Vmarket(i,j);
    end
    VboughtTot = sum(Vbought(i,:));
    VsoldTot = sum(Vsold(i,:));
    portfolio = portfolio + VboughtTot - VsoldTot;
end

Vdifference = Vbought-Vsold;
```

tradedVolumesMultiple.m

```
function [Vdifference,Vbought,Vsold,Vmarket,uBuy,uSell] =
tradedVolumesMultiple(n,horizon,prices,portfolio,modifyUorNot,numStocks,vol
umesTraded,minOpt,maxOpt,currentSD)
%%%%%INPUTS
%n is the interval on which each volume is computed (eg n=2 the day is
divided in 195 2-minute intervals)
%prices is the vector of prices based on the same intervals as the ones the
volumes are computed
%portfolio is the number of shares of the portfolio we are currently
holding (positive or negative)
%modifyUorNot indicates whether we change aggressiveness or not (1 yes,2
no)
%numStocks is the number of stocks in the portfolio]
%volumesTraded is the number of stocks traded in each interval for each
stock
%minOpt and maxOpt correspond to the values of the bounds for which
aggressiveness will change
%currentSD is the covariance matrix of the stocks considered for that day
%%%%%OUTPUTS
%Vmarket: vector containing the number of shares traded in the whole market
on a single stock during the intervals we are interested in
%Vbought, Vsold: vectors containing the number of shares we bought and sold
during the intervals
%Vdifference: vector being the difference between the number of stocks
bought and sold, during each interval
%uBuy and uSell: vectors representing the values of u (buy and sell) during
each of the intervals throughout the day

if strcmp(horizon,'days')
    split = 1;
elseif strcmp(horizon,'minutes')
    split = 6.5*60;
elseif strcmp(horizon,'seconds')
    split = 6.5*60*60;
end

[mu,sigma,lambda] = getVolumeInputs();

%in case we cannot fit an integer number of n in our split we compute the
length of the last interval (rest)
%eg for 390 minutes, if n = 45 minutes we have 8 intervals of 45 minutes
and one of only 30 minutes
divider = split/n;
intervals = floor(divider);
rest = divider - intervals;

mu_vector = volumesTraded;
sigma_vector = repmat(sigma*sqrt(n),intervals,numStocks);

% % if rest ~= 0
% %     intervals = intervals + 1;
% %     mu_vector = [mu_vector; mu*rest*n];
% %     sigma_vector = [sigma_vector;sigma*sqrt(rest*n)];
% % end

Vmarket = mu_vector.*normrnd(1,sigma_vector);

returns = zeros(intervals,numStocks);

for i=1:intervals
```

```
    returns(i,:)=prices(i+1,:)/prices(i,:)-ones(1,numStocks); %value of
0.02 means a return of 2%
end

Vbought = zeros(intervals,numStocks);
Vsold = zeros(intervals,numStocks);
uBuy = zeros(intervals,numStocks);
uSell = zeros(intervals,numStocks);

for i = 1:intervals
    if modifyUorNot == 1
        [uBuy(i,:),uSell(i,:)] =
modifyUMultiple(portfolio,numStocks,minOpt,maxOpt,currentSD);
    else
        uBuy(i,:) = ones(1,numStocks);
        uSell(i,:) = ones(1,numStocks);
    end
    for j = 1:numStocks
        Vbought(i,j) =
0.01*generate_POV(returns(i,j),1,uBuy(i,j))*Vmarket(i,j);
        Vsold(i,j) = 0.01*generate_POV(returns(i,j),-
1,uSell(i,j))*Vmarket(i,j);
    end
    portfolio = portfolio + Vbought(i,:)' - Vsold(i,:)';
end

Vdifference = Vbought-Vsold;
```

<u>modifyU.m</u>

```
function [uBuy,uSell] =
modifyU(portfolio,numStocks,minOpt,maxOpt,currentSD)

portfolio = portfolio*currentSD; %risk adjusted portfolio
min = minOpt;
max = maxOpt;
uBuy = 1;
uSell = 1;
step = 0.5;
indices = 1:step:5;
if mod(5,step) ~= 0 %to make sure 5 is actually in the vector
    indices = [indices 5];
end
l=length(indices);

diff = max - min;
inc = diff/(l-2);
threshold = min:inc:max;

k=0;
for i=(l-1):(-1):1
    if abs(portfolio)>threshold(i)
        k=i;
        break
    end
end
```

```
if (k~=0 & portfolio>0)
    if k==1
        uBuy = 0;
    end
    uSell = indices(k);
end

if (k~=0 & portfolio<0)
    if k==1
        uSell = 0;
    end
    uBuy = indices(k);
end

uBuy = repmat(uBuy,1,numStocks);
uSell = repmat(uSell,1,numStocks);
```

modifyUMultiple.m

```
function [uBuy,uSell] =
modifyUMultiple(portfolio,numStocks,minOpt,maxOpt,currentSD)

Inv = portfolio';
portfolio = sqrt(portfolio'*currentSD*portfolio); %risk adjusted portfolio
min = minOpt;
max = maxOpt;
uBuy = 1;
uSell = 1;
step = 0.5;
indices = 1:step:5;
if mod(5,step) ~= 0 %to make sure 5 is actually in the vector
    indices = [indices 5];
end
l=length(indices);

diff = max - min;
inc = diff/(l-2);
threshold = min:inc:max;

k=0;
for i=(l-1):(-1):1
    if abs(portfolio)>threshold(i)
        k=i;
        break
    end
end


if (k~=0 & portfolio>0)
    if k==1
        uBuy = 0;
    end
    uSell = indices(k);
end

if (k~=0 & portfolio<0)
```

```
    if k==1
        uSell = 0;
    end
    uBuy = indices(k);
end

ub = uBuy;
us = uSell;
uBuy = ones(1,numStocks);
uSell = ones(1,numStocks);

for i = 1:numStocks
    if(isnan(Inv(i)) == 0 && Inv(i) ~=0)
        if(Inv(i) < 0)
            uBuy(i) = us;
            uSell(i) = 1;
        else
            uBuy(i) = 1;
            uSell(i) = us;
        end
    else
        uBuy(i) = 1;
        uSell(i) = 1;
    end
end
```

generate_POV.m

```
% Generates a Percentage of Volume that depends on the price change,
% trade type, and stated aggression level.  The numbers used were chosen
% based on our analysis of the trade data.
%
% Inputs:
%     price change - the amount the price rose or fell during this trade
%     trade type - (+1) for buying / (-1) for selling
%     u - aggression level
function POV = generate_POV(price_change, trade_type, u)

if u == 0
    POV = 0;
else
    lambda = 1.0;
    x = trade_type*price_change;

    % Sets lambda in the case of an favorable price move
    if (x > 0)
        lambda = 9.756294*(-0.54604*log(x) - 1.885964);
    % Sets lambda in the case of an unfavorable price move
    elseif (x < 0)
        lambda = 4.424209*(-0.624*log(abs(x)) - 1.5152);
    end

    % Generates an exponential random variable to simulate
    % X := sqrt(price_change)*POV    where  X ~ exp(lambda)
    e1 = ( -log(rand()) )/( lambda );

    % The simulated percentage of volume.
    %    For now, if no price change then choose a POV ~ uniform(2,20)
    if (price_change == 0)
        POV = unifrnd(2,20);
    else
        POV = ( e1 ) / ( sqrt(abs(x)) );
    end

    if POV > 15
        POV = 15;
    end

    % Adjusted the percentage of volume based on aggression level.
    %    For now, we assume each aggression level we move up adds a
percentage
    %    point to POV
    POV = POV + (u-1);
end
```

generate_VWAP.m

```
% Generates a VWAP slippage that depends on the spread, trade type, and
% stated aggression level.  The numbers used were chosen based on our
% analysis of the trade data.
%
% Inputs:
%    spread: vector of the bid/ask spreads for the stocks traded (in bps)
%    type  : vector indicating whether each trade was a BUY (+1) or SELL (-
1)
%    u     : the aggression levels, vector of integers from 1 to 5
%
% Outputs:
%    VWAP_slippages : the vector of simulated VWAP slippages
%
function VWAP_slippages = generate_VWAP(spread, type, u)

n = length(spread);
VWAP_slippages = zeros(n,1);
mu = zeros(n,1);
sigma = zeros(n,1);

for i = 1:n
    % Set parameters if a BUY Trade
    if (type == 1)
        mu(i) = (-0.458944 + .25*(u(i)-1))*spread(i) + 0.00003;
        sigma(i) = 0.7*(0.938859*spread(i) + 0.000164);
    % Set parameters if a SELL Trade
    elseif (type == -1)
        mu(i) = (-0.502295 + .25*(u(i)-1))*spread(i) + 0.0000367;
        sigma(i) = 0.7*(1.144760*spread(i) + 0.000050);
    end

    VWAP_slippages(i) = normrnd(mu(i),sigma(i));
end

VWAP_slippages = -VWAP_slippages;
```

VWAP_Function.m

```
% generates VWAP
% needs a vector of prices and market volumes of the same length
% the first number of the volumes vector should be 0 (since we don't trade
% at time 0)
% -----------------------------------

function VWAP = VWAP_Function(prices, vol)

l = length(prices)-1;
VWAP = zeros(l,1);

for i=1:l
    price_t = prices(i);
    price_t1 = prices(i+1);
    vol_t = vol(i);
    vol_t1 = vol(i+1);
    VWAP(i) = (price_t*vol_t + price_t1*vol_t1)/(vol_t + vol_t1);
end
```

getSpreads.m

```
function spreads = getSpreads(tickers)

n = length(tickers);

spreads = nan(n,1);

[num,txt,raw] = xlsread('Spreads');
txt = txt(2:end); %gets rid of the header

for i = 1:n
    x = strmatch(tickers(i),txt,'exact');
    spreads(i) = num(x);
end
```

computeSD.m

```
function sd = computeSD(dailyPrices)

window = 20; %length of trailing window

sd = zeros(length(dailyPrices),1);

lastPrices = zeros(window);

differentSD = length(dailyPrices)-window-1;
for i = 1:differentSD
    lastPrices = dailyPrices(i:i+window-1);
    sd(i+window) = std(lastPrices);
end
```

```
sd(1:window) = sd(window+1);
sd(end) = sd(end-1);
```

computeCov.m

```
function covMat = computeCov(dailyPrices)

numStocks = size(dailyPrices,2);
window = 20; %length of trailing window
l = size(dailyPrices,1);

covMat = zeros(numStocks,numStocks,l);

lastPrices = zeros(window,numStocks);

differentSD = l-window-1;
for i = 1:differentSD
    lastPrices = dailyPrices(i:i+window-1,:);
    covMat(:,:,i+window) = cov(lastPrices);
end

mat = covMat(:,:,window+1);
for j = 1:window
    covMat(:,:,j) = mat;
end
covMat(:,:,end) = covMat(:,:,end-1);
```

getVolumeInputs.m

```
function [mu,sigma,lambda] = getVolumeInputs()
%output is average and standard deviation of volume traded per minute

mu = 10^8;
sigma = 0.8; %standard deviation of traded volume
lambda = 10; %correlation with stock return

numberTradingHours = 6.5; %trading is from 9:30am to 4pm
n = numberTradingHours*60;

mu = mu/n;
sigma = sigma/sqrt(n);
```

count_unique.m

```
function [uniques,numUnique] = count_unique(x,option)
%COUNT_UNIQUE  Determines unique values, and counts occurrences
%   [uniques,numUnique] = count_unique(x)
%
%   This function determines unique values of an array, and also counts the
%   number of instances of those values.
%
%   This uses the MATLAB builtin function accumarray, and is faster than
%   MATLAB's unique function for intermediate to large sizes of arrays for
integer values.
%   Unlike 'unique' it cannot be used to determine if rows are unique or
%   operate on cell arrays.
%
```

```
%   If float values are passed, it uses MATLAB's logic builtin unique
function to
%   determine unique values, and then to count instances.
%
%   Descriptions of Input Variables:
%   x:  Input vector or matrix, N-D.  Must be a type acceptable to
%       accumarray, numeric, logical, char, scalar, or cell array of
%       strings.
%   option: Acceptable values currently only 'float'.  If 'float' is
%           specified, the input x vector will be treated as containing
%           decimal values, regardless of whether it is a float array type.
%
%   Descriptions of Output Variables:
%   uniques:    sorted unique values
%   numUnique:  number of instances of each unique value
%
%   Example(s):
%   >> [uniques] = count_unique(largeArray);
%   >> [uniques,numUnique] = count_unique(largeArray);
%
%   See also: unique, accumarray

% Author: Anthony Kendall
% Contact: anthony [dot] kendall [at] gmail [dot] com
% Created: 2009-03-17

testFloat = false;
if nargin == 2 && strcmpi(option,'float')
    testFloat = true;
end

nOut = nargout;
if testFloat
    if nOut < 2
        [uniques] = float_cell_unique(x,nOut);
    else
        [uniques,numUnique] = float_cell_unique(x,nOut);
    end
else
    try %this will fail if the array is float or cell
        if nOut < 2
            [uniques] = int_log_unique(x,nOut);
        else
            [uniques,numUnique] = int_log_unique(x,nOut);
        end
    catch %default to standard approach
        if nOut < 2
            [uniques] = float_cell_unique(x,nOut);
        else
            [uniques,numUnique] = float_cell_unique(x,nOut);
        end
    end
end

end

function [uniques,numUnique] = int_log_unique(x,nOut)
%First, determine the offset for negative values
minVal = min(x(:));
```

```
%Check to see if accumarray is appropriate for this function
maxIndex = max(x(:)) - minVal + 1;
if maxIndex / numel(x) > 1000
    error('Accumarray is inefficient for arrays when ind values are >> than
the number of elements')
end

%Now, offset to get the index
index = x(:) - minVal + 1;

%Count the occurrences of each index value
numUnique = accumarray(index,1);

%Get the values which occur more than once
uniqueInd = (1:length(numUnique))';
uniques = uniqueInd(numUnique>0) + minVal - 1;

if nOut == 2
    %Trim the numUnique array
    numUnique = numUnique(numUnique>0);
end
end

function [uniques,numUnique] = float_cell_unique(x,nOut)

if ~iscell(x)
    %First, sort the input vector
    x = sort(x(:));
    numelX = numel(x);

    %Check to see if the array type needs to be converted to double
    currClass = class(x);
    isdouble = strcmp(currClass,'double');

    if ~isdouble
        x = double(x);
    end

    %Check to see if there are any NaNs or Infs, sort returns these either
at
    %the beginning or end of an array
    if isnan(x(1)) || isinf(x(1)) || isnan(x(numelX)) || isinf(x(numelX))
        %Check to see if the array contains nans or infs
        xnan = isnan(x);
        xinf = isinf(x);
        testRep = xnan | xinf;

        %Remove all of these from the array
        x = x(~testRep);
    end

    %Determine break locations of unique values
    uniqueLocs = [true;diff(x) ~= 0];
else
    isdouble = true; %just to avoid conversion on finish

    %Sort the rows of the cell array
    x = sort(x(:));
    %Determine unique location values
    uniqueLocs = [true;~strcmp(x(1:end-1),x(2:end)) ~= 0] ;
end

%Determine the unique values
uniques = x(uniqueLocs);

if ~isdouble
    x = feval(currClass,x);
end

%Count the number of duplicate values
if nOut == 2
    numUnique = diff([find(uniqueLocs);length(x)+1]);
end
end
```