A Case Study of a High Frequency Trading Firm

Ronny Grimstål, Patrik Paluch

May 2012

21314@student.hhs.se
21819@student.hhs.se

Abstract

This thesis studies the trading of High Frequency Trading firm Spire Europe on the NASDAQ OMX Nordic exchange with the goal of trying to characterise the way they make profit and how they manage risk. We conclude that the return of high frequency trading has very different characteristics from ordinary trading and hence methods developed for researching ordinary trading has little value when trying to investigate high frequency trading.

Keywords: High Frequency Trading, Spire Europe, Price anticipation

Acknowledgements:

We would like to thank APS Capital for kindly supplying us with data and support during the work with this thesis. We also like to thank our tutor Laurent Bach for his support.

Table of Contents

Introduction 3

Background 3

High frequency trading 3

NASDAQ OMX Nordic 5

Spire Europe 6

How order books work 7

High Frequency Trading Strategies 8

Theory and prior research 9

On the curvature of Ricci flows under SL(y) cohomology (Market microstructure and high frequency trading) 9

Value at risk 10

Data description and methodology 13

Data description 13

Methodology 15

Results 16

Analysis 22

Conclusions 24

References 26

Appendix 27

Introduction

We study the high frequency trading firm Spire Europe's trading on the NASDAQ OMX Nordic exchanges by looking at intraday trading data. We examine both their profitability in individual stocks and what risk they take on by looking at different periods of intraday Value at Risk measures. Furthermore we try to empirically explain the intraday returns by use of standard OLS regressions.

Most research produced on the topic of market microstructure is very theoretical in nature and few studies use intraday data to study a specific firm. Our study is empirically oriented, and based on one firm's trading, we try to understand what kind of return the firm generates and what risk that return comes with.

Background

High frequency trading

High-frequency trading (abbreviated HFT) firms use advanced computer algorithms to trade large quantities of securities in very brief periods of time. This allows them to capture small profit opportunities that may only last for fractions of a second, and because of this they do not hold any overnight positions. It was the move to smaller tick sizes in the US capital markets in 2000, when pennies started being used in price quotes, that really made market volume explode according to Clark (2010). This led to computers being more appropriate to handle the huge increases in data points rather than humans.

High frequency trading is part of a broader category of trading called algorithmic trading. Algorithmic trading is based on trading done in part or in whole by computer algorithms. There are two general types of algorithms that are used, execution algorithms and speculative algorithms. Execution algorithms are designed to attain the best price for a transaction specified by the trader. They are particularly useful for partitioning block trades into smaller parts. Block trades are difficult to trade because you need to find a counterparty that is willing to trade the entire block of shares with you. The execution algorithm tries to avoid this problem by dividing the trade into smaller ones to get a better price and to avoid making a large impact on the market. Speculative algorithms are algorithms that make automatic decisions about what trades to make. In other words they can make decisions on what to trade, when to trade it and how much of it to trade

without interference from humans. High frequency traders employ both execution and speculative algorithms depending on what strategy they use. Before implementing the algorithms they go through extensive backtesting to assess their functioning in the market. To get an edge over competitors high frequency trading firms make big investments in hardware and software to make systems faster, and since there is a steep learning curve to become a good high frequency trader human capital is very important. It is also a de facto standard to be directly connected to the exchanges' servers, something called colocation.

Speed in particular is at a premium and new platforms increase speed by multiple orders of magnitude. Speed can be broken down into throughput and latency. Throughput is the total number of transactions a system can execute each second. Latency is the time it takes between the placement of an order and when it reaches the market. Reliability is the second important factor to consider when designing quality trading platforms. With orders being placed at such high frequency, down time that results in leaving traders unable to place or cancel orders could prove very costly. The solution is to make systems highly available and redundant so you can switch to a backup system should you need it. Finally, Amicangioli (2006) writes that service providers must offer capabilities that allow traders to switch to other facilities in the event of some natural disaster or terror attack

Although there are relatively few HFT firms in the US markets, about 2 % of 20 000 firms, they account for almost three-fourths of all equity trading volume on that market. Firms engaging in HFT can be everything from major investment banks and hedge funds to small prop shops, Iati (2009) writes. The TABB Group, a financial markets research firm, estimates that aggregated annual profits of these firms exceeded $21 billion in 2008. Clark (2010) writes that algorithmic trading on the US market grew from 30 % in 2005 to 70 % in 2009 and according to research from consultancy Aite Group, high frequency trading accounts for almost 40% of European trading volume, Cave (2011a) reports.

An important aspect of the competition among different HFT firms is that the micro-level algorithms have a limited lifespan and that competitive advantages might quickly be diluted. One reason for this is that many statistical relationships between securities can change very fast, which requires a corresponding change in the code for the algorithms. Secondly, there's a risk that competitors will reverse-engineer your algorithms, enabling them to front-run or trade against you, thus making them obsolete as Iati (2009) writes.

To maximize the probability that they close a position profitably high frequency traders cancel and send orders continuously to both sides of the order book. They also vary between submitting aggressive and passive orders (market and limit orders) in order to achieve the best position in the book. It is common to see cancellations of a lot of orders on both sides of the book after a market order hits a bid or lifts an offer as a result of high frequency traders withdrawing because they were not able to capture a trade. Even though HFT firms can position themselves in the front of the order book there is no guarantee that they will be able to close the position profitably, and HFTs cannot know whether a trade is a liquidity trade or for example a trade from another HFT.

Many traders send orders through clearing members' trading systems. To control risk clearing members have limits on which trades may be executed based on price and quantity. In addition, there are also exchanges that have a built-in function that cancels clearly erroneous trades. To reduce latency HFT firms choose to use their own platforms to send their orders to the exchange. We distinguish between two different types of arrangements: sponsored access and unfiltered sponsored access. The former allows traders to access the exchange matching engine directly, and that arrangement also includes the same kinds of pre-trade risk controls as the clearing members' own trading systems. The unfiltered sponsored access reduces latency by bypassing the risk controls and sends orders directly to the matching engine. This type of access is not without risks, as history clearly shows, as it is more susceptible to for example "fat finger trades". In 2007 a Morgan Stanley trader entered an order to buy 100 000 shares while using a tool that had a built-in multiplier of 1000. The order was for $10.8 billion instead of $10.8 million. Before the bank noticed the mistake and canceled the order close to $900 million worth of shares had been traded, according to Clark (2010). It is largely unknown how the number of HFT firms has changed during the last couple of years. Depending on where one looks the number of HFT firms has increased in some markets and decreased in others, but the profits of early entrants seem to be stable.

NASDAQ OMX Nordic

On the NASDAQ OMX Nordic market HFT firms doubled their trading in 2011 from the year before. The total value of equities traded by HFT firms rose to EUR170 billion

and they accounted for 12 % of all trading in 2011. The value of equities traded for the market as a whole increased 5 % to EUR1.4 trillion, as Cave (2012) reports.

The priority among orders on these markets is firstly "price", secondly "internal", then "display", and lastly "time". Unsurprisingly, price has first priority and the better price has priority over poorer prices. Internal priority means that orders are first executed against Exchange Members' orders. Orders are then matched with regard to displayed over non-displayed volume. Lastly, orders are matched according to when it was sent to the order book, and there is time priority for pre-open orders, as can be seen in the NASDAQ OMX Nordic Market Model 2.6 (2011).

Spire Europe

Spire Europe LLC is a hedge fund specializing in high frequency trading that operates on the European financial markets. It's owned by Tower Research Capital LLC, a financial services company in New York that specializes in quantitative trading and investment strategies. Spire Europe was established in 2006, but began active trading in 2009. It describes itself as a "proprietary trading firm dealing in equities, futures and foreign currencies." It increased its trading revenues by six times in 2010, from £5m to £32.1m according to accounts, and its pre-tax profits increased from £3.1m to £22m as is reported in Cave (2011b).

On the Stockholm Stock Exchange Spire executes about 20 000 trades each day to a value of 1 billion SEK, and in 2011 they traded shares worth more than EUR30 billion on the NASDAQ OMX Nordic markets. NASDAQ OMX is one of the only exchanges in the world to reveal the level of trading activity of its member firms, Cave (2012). NASDAQ OMX's data also reveal that Spire was second only to Getco among high-frequency players in September 2011 by market share, and the eleventh-largest provider of liquidity overall. It traded equities worth €3.6bn, giving it a 3.2% share of the market. It is one of 20 members of the European Principal Traders Association, a lobby group set up in 2011 to represent the interests of firms that trade using their own capital according to Cave (2011). Mehta (2011) reports that the last couple of years have seen trading volume significantly lower than was common in for example 2007, and there is some evidence to suggest that this has been bad for HFT firms, including Spire.

Tower Research Capital was founded in 1998 by Mark Gorton and is one of the first firms in the world to use algorithms in financial markets. They develop proprietary trading algorithms based on statistical methods to find non-random patterns in the

market. Tower earns its profit by exploiting small inefficiencies in prices that arise for only brief periods of time. According to Tower's official webpage (2012) the company is made up of a combination of highly skilled individuals with backgrounds in a variety of fields: mathematics, computer science, statistics, physics, economics, engineering, and finance. Tower started trading shares on NASDAQ and then expanded to futures, bonds and currency markets. Today they trade on virtually every liquid electronic market. Bursell (2012) also reports that Tower has about 300 employees but precise information on how much in trading turnover they generate, and on which markets they trade on, are not public. To handle orders Gorton also founded a brokerage firm, Lime Brokerage, to speed up the lead time for trades. Tower and Lime Brokerage were part of the Lime Group which also established the file sharing service LimeWire. Lime Brokerage was bought by large US broker Wedbush in 2011. Tower had $117 million in assets under management in 2007. The performance of Gorton's financial companies in recent years is not publicly known other than that the brokerage house lost money last year. Tower has also never had a down year from the start until at least 2008 as can be found in Plambeck (2010) and on Tower's official website (2012b).

How order books work

Trades on stock exchanges are coordinated by a system of double auction. Both buy (bid) and sell (ask) orders are quoted in an order book where one can see how many shares (volume) market participants are willing to buy or sell at a particular price. The types of orders that appear in the order book are called limit orders. They are orders that specify both the volume and to what price a market participant is willing to transact. Another common order type is a market order, that is an order where you only specify the volume of the stock you wish to trade. The order will then be executed immediately (assuming there is someone willing to trade) at the best available price in the market at the time.

The buy and sell orders are arranged in two different columns and the best prices (the highest buy price and lowest sell price) and their corresponding volume are quoted at the top of the order book. The difference between these two best prices in the order book is called the bid-ask spread. If there are matching orders, that is, if the ask price of an order equals the bid price of an order, the orders are executed and are subsequently removed from the order book.

High Frequency Trading Strategies

High frequency trading firms employ several different strategies with the common characteristic that they are all suited for very short holding periods. In Aldrige (2010) the different trading strategies are classified and described in the following way.

Automated liquidity provision – algorithms automatically offers bid and ask prices and execute orders to make a market in financial instruments

Market microstructure trading – Algorithms detect changes in order flow and anticipate price changes

Event trading – Short-term trading on macroeconomic news events

Statistical arbitrage – Finding combinations of assets that have pricing inefficiencies, then trading the assets in anticipation that the market will eventually correct the inefficiencies

The most straightforward of the market microstructure trading strategies is a strategy called *price anticipation* described by Cartea et. al. (2012). The objective of a high frequency trader that uses price anticipation is to take advantage of imbalances between the buy side and the sell side of a limit order book. Such an imbalance might be caused by, for instance, a large institution wanting to buy or sell a large amount of shares in the market. Price anticipation is further explained below with the help of a hypothetical example.

Assume that there is an increase in buy orders in a particular stock. This might be due to an institutional investor wanting to buy a large amount of shares. The HFT algorithms will detect the rising buying pressure and cancel all of its sell orders. They then add additional buy orders to clear out the top of the order book at the sell side and at the same time start adding new sell orders at a higher price than the previous prevailing price. Buyers realizing that they cannot get a fill at their current bid price will be forced to raise their price in order to buy shares. The HFT firm is then able to sell shares at the new higher price. When the buying pressure wears off the HFT algorithms will notice and will be able to cover the rest of its positions at a lower price. The HFT firm has now earned the spread between the higher price it sold shares for and the lower price it was

buying them back for. In order to make it harder for high frequency traders to detect large orders institutional investors can use execution algorithms such as VWAP, liquidity pools or iceberg orders.

Theory and prior research

Market microstructure and high frequency trading

There has been a lot of research made on market microstructure and also specifically on the effects of high frequency trading on markets, but not much empirical research about the risk and return from high frequency trading has been produced. The following is a short exposition of interesting and relevant findings within market microstructure research and related fields.

Vega (2007) proposes observing the share of market orders in order to gain market information. The motivation is that aggressive orders reflect beliefs that there is a higher likelihood that prices will move away from the market price. Traders will therefore want to submit market orders in an effort to capture the best available price in anticipation of an imminent price move, as described by Aldridge (2010).

Berber and Caglio (2004) find that limit orders contain private information around certain events such as earnings announcement. Furthermore Harris and Panchapagesan (2005) show that traders that can fully view the order book can earn abnormal returns. This suggest that traders that can process large amounts of data and observe the evolution of the order book such as HFT firm have a big advantage in some cases compared to slow traders by anticipating price moves and constantly change their positions in response of market movements.

In Wang, Hansch and Cao (2004) the shape of the limit order book is exploited by market makers and the breadth and the depth of the order book predict 30 % of the impending price moves. Holden and Subrahmanyam (1992) find that the more volatile the price of a security is, the lower the depth of the information in the order book. They find that small changes in volatility will lead to large changes in transaction prices, and that this in turn will lead to a lower probability that informed traders will provide liquidity. This insight is evidence that certain informed liquidity providers like HFT firms withdraw from the market when it is not possible to get an informational advantage as a result of high volatility. Additionally, according to the SEC (2010) report, 6 out of 12

HFT firms withdrew from the market during the Flash Crash of May 6 2010. They also report a reduction in average execution time from 10.1 seconds in January 2005 to 0.7 seconds in October 2009. This has made possible faster price adjustments across different markets and made markets more efficient in the sense that arbitrage opportunities exist for a shorter time. Kirilenko et al. (2010) concludes that HFTs had the highest price impact among all types of traders during this time. They also find that HFTs increased volume of trading without increasing the flow of information, and increased noise around the fundamental value of the asset. On the other hand, papers like Hendershott and Riordan (2012) high frequency trading have a positive effect on price discovery and may dampen volatility and price impact.

Cartea and Penalva (2012) find that the price impact of liquidity trades is higher in the presence of HFTs and that the impact increases with the size of the trade. As a reaction to the presence of HFTs market participants adjust their security supply and demand functions, which lead to an increase in price impact by creating a temporary order imbalance thus creating an additional market impact cost on participants. HFTs also increase the microstructure noise of prices through intermediation of trades by buying cheap and selling a little higher and also through larger price impact. This increases the perceived price risk for other traders, and makes the future uncertainty of their execution costs result in an increase in the liquidity discount. HFTs also decrease profits for liquidity providers as well as for other traders by intermediating trades, but they contribute to making higher liquidity discounts to the market price in many cases. They also find that competition between HFTs create uncertainty for other liquidity providers that distorts total market liquidity in equilibrium. Biais et al. (2011) conclude that HFT's effect on liquidity is ambiguous, depends on several factors and requires more quality empirical research, in which the authors of this paper concurs.

Engle and Russell (1998) report that two-thirds of transactions in intraday data are without price change. Gorski, Drozdz and Speht (2002) report basically the same phenomenon but for another data sample. (Tsay, 2002) also notes that microstructure effect may induce serial correlation in high frequency returns. This is something that we find in our data too.

Value at risk

Due to the nature of HFT, the risk of generating errors faster and greater than other forms of trading is always present. For this reason risk management at these firms is important. Because HFT firms do not hold a lot of inventory at any one time and because they trade on such short time scales the use of proper risk measures to get valid and useful results is essential. To control risk HFT firms may sometimes just withdraw from the market as discussed before if their algorithms are not working, and humans are used for that purpose by monitoring positions. Below, we estimate Intra-day Value at Risk measures using the Historical simulation method to model the market risk that Spire takes on.

The definition of *value at risk* (VaR) in Hult et al. (2007) is that, given a loss L and a confidence level $\alpha \in (0,1)$, $\mathrm{VaR}_\alpha(L)$ is the smallest number l such that the probability that the loss L exceeds l is no larger than $1-\alpha$, i.e.

$$VaR_\alpha = \inf\{l \in \mathbb{R}: P(L > l) \leq 1-\alpha\} =$$
$$= \inf\{l \in \mathbb{R}: 1 - F_L(l) \leq 1-\alpha\} =$$
$$= \inf\{l \in \mathbb{R}: F_L(l) \geq \alpha\}$$

in other words VaR can be thought of as the potential loss one might suffer from holding a portfolio over some, fixed, time interval. It is often used by institutions to measure risk over next ten day. The validity of a VaR is measured by determine the frequency of periods where the realized loss exceeds the estimated VaR.

The three most common methods to calculate VaR are Historical Simulation, Monte Carlo Simulation and the Variance-Covariance method. The Variance-Covariance method assumes normally distributed returns but because intraday returns exhibit fat tails as Giot (2011) writes, it is not suitable for our purpose. The Monte Carlo Simulation also assumes a particular distribution, although it doesn't need to be a normal distribution, Vosper et al. (1999). Since the data is so chaotic we believe that the non-parametric Historical simulation method serves us best.

Historical simulation makes no assumption about the probability distribution of losses except that they are iid during the different time intervals. The approach of the method is that to estimate VaR for the time period VaR_{t+1} one uses the current portfolio and simulates a loss distribution by determining how historical time periods would have

generated for losses in the current portfolio. The disadvantage is that the worst simulated outcome is never worse than what has happened before.

Historical simulation is appropriate only if returns are iid. However, that is violated when volatility changes over time. This leads to inconsistent estimates that are shown to underestimate the VaR. Filtered historical simulation uses a parametric GARCH-model to filter past returns to acquire a new iid innovation that is used together with the non-parametric historical portfolio returns to simulate a VaR; Vosper et al. (1999). FHS also help remove serial correlation in the data.

Intraday value at risk (IVaR) is a measure of VaR using shorter periods than one day. Since high frequency trading firms change their inventory very rapidly, the ordinary daily VaR is an inadequate risk measure. It is well-known that high-frequency data exhibit strong intraday seasonality due to the fact that markets tend to be more active at the beginning and towards the end of the trading day (Engle and Russell, 1998). The models providing the best VaR estimates are those using intraday data and that control for periodicity simply or not at all. It is also more important to have intraday data (as compared to only daily observations) than what model is used to get good results, according to McMillan et al. (2007). In addition Historical simulation has proven to outperformed the parametric models when calculating VaR, Qi (2011).

Almeiday and Vicente (2008) explain that to test whether VaR estimations are good they have to be tested for *Unconditional Coverage Property* and *Independence Property*. The Unconditional coverage property is whether the average number of violations are equal to the statistically expected one and a violation is when realized loss exceeds the VaR estimate. The Independence property means that the violations are independently distributed i.e. that they are not clustered in time. If this condition is not met it means that the violation carry information on future volatility.

To test for the unconditional coverage property the *Kupiec test* is used. We take the time series of VaR-values and compare it to the corresponding time series of realized returns. Then we creates a hit sequence which take the value 1 for every time the realized loss is greater than the VaR and zero otherwise.

$$I_t(\alpha) = \begin{cases} 1 \; if \; realized \; loss > VaR \\ 0 \; else \end{cases}$$

Construct the variable

$$N_L(\alpha) = \sum_{t=1}^{L} I_t(\alpha)$$

L is the number of violations observed. If the model is accurate the Kupiec test gives

$$2\ln\left(\left(\frac{1-\alpha}{1-\hat{\alpha}}\right)^{L-N_L(\alpha)}\left(\frac{\alpha}{\hat{\alpha}}\right)^{N_L(\alpha)}\right) \sim \chi^2(1), \text{ as } L \to \infty$$

where $\hat{\alpha} = N_L(\alpha)/L$.

To test for the independence property one uses the *Christoffersen test*. If the probability of a violation in this period has no effect on the probability of a violation in the next period

$$\frac{L_{10}}{L_{00}+L_{10}} = \frac{L_{11}}{L_{01}+L_{11}}$$

holds. L_{ij} is the number of observations in I_t where observation I is followed by j. The Christoffersen test statistic is given by

$$2\ln\left(\frac{(1-\pi_{01})^{L_{00}}\pi_{01}{}^{L_{01}}(1-\pi_{11})^{L_{10}}\pi_{11}{}^{L_{11}}}{(1-\pi)^{L_{00}+L_{10}}\pi^{L_{01}+L_{11}}}\right) \sim \chi^2(1), \text{ as } L \to \infty$$

where $\pi_{ij} = \frac{L_{ij}}{\sum_j L_{ij}}$ and $\pi = \frac{L_{01}+L_{11}}{L}$.

Data description and methodology

Data description

Our data consists of the NASDAQ OMX Nordic data feed, stored in so called ITCH files. They are files containing all the actions undertaken on the NASDAQ OMX NORDIC exchanges. The ITCH files are vast in size – between 30 to 40 million rows

per day – which makes the data processing quite complex and computationally expensive. We have used ITCH files for ten days during the spring of 2012. Also, the data is heterogeneous in time, meaning that the data points are not evenly spaced in time.

In our data Spire is trading in 44 different very liquid stocks. The data is taken from the stocks that we determined that Spire is trading in. Since the data set is so vast we have clustered the data. We are mostly using 1, 10 or 30 minute intervals although some regressions is done on microstructure data.

A sample from a NASDAQ OMX Nordic data feed is presented below:

```
M284
R 86446SCA2F 125SHB    SE0004208031  8SEKXSTO  40            1
R 86447SCA2P 105SHB    SE0004208049  8SEKXSTO  40            1
R 86448SEB2F  48SHB    SE0004208056  8SEKXSTO  40            1
R 86449SEB2F  52SHB    SE0004208064  8SEKXSTO  40            1
…
R 62494NET B          SE0001989252  1SEKXSTO  20            1
….
I      0      0O 84223    00000O    00000      0    00000      0
I      0      0O 84224    00000O    00000      0    00000      0
I      0      0O 84225    00000O    00000      0    00000      0
I      0      0O 84226    00000O    00000      0    00000      0
...
A  7072827B      300  1158   7725000
M565
D  7072815
A  7072828S     4000 75064   2199000
M566
A  7072829S      500 75121   4369000
D  7072819
A  7072830S      500 75327   2136000
…
A  1059830S      700 62494    632500
…
M402
E    40328      703     5927SWB UBS
E  1059830       85   202704NON ENS
…
D    90487
Q       0    80     00000     4207O         0
E    86465      824     4208ENS DBL
```

Table 1: NASDAQ OMX data feed

The data feeds we have used contain between 30-50 million rows. The feed is saved as a regular ASCII text in files that is about one gigabyte in size. The files comprise everything that happens on the exchange in chronological order. Some examples on how to read the data feed is presented below.

The row: "A 7072827B 300 1158 7725000" is interpreted as follows:
'A' means add a new order, 7072827 is a reference number used when the order is cancelled or executed. 'B' indicated that it is a buy order and 300 is the quantity of shares in the order. 1158 is the stock identifier, in this case Novo Nordisk B stock (which can be determined from the messages starting with an "R") and 7725000 is the price 772.5 DKK.

The row "E 1059830 85 202704NON ENS" means that the order, or part of the order, with reference number 1059830 has been executed. 85 is the number of shares executed (in this case out of 700, which can be seen in the corresponding added order message in the table above). 202704 is a match number used if something goes wrong and a Broken Trade message is sent. NON is the ID of the owner to the order, in this case Nordnet. ENS is the counterpart in the trade, in this case SEB. To determine who the buyer and seller was one has to look at the order to see if it is a sell order or a buy order. In this case there is an 'S' at the end of the order which implies that the owner (Nordnet) is the seller and hence SEB is the buyer. To see the price and in what order book the trade was, one also has to look at the Add Order message.
This is just a small subset of the message types present in a data feed. There is a lot of different messages for different types of orders, different types of order modifications, trades and system codes that indicate the state of the market.
In order to build a complete order book from a data feed one has to keep track of all the messages that impact that order book. That is, when orders are added, modified, deleted or executed and when trades in the order book are broken. All active orders in the book have to be maintained properly after every new message i.e. ranked on price and time of arrival at both the buy and sell side.

Methodology

We study how the IVaR behaves under different resolutions of returns. We use 10 minute and 1 minute returns and the Historical simulation method to calculate the IVaR.

We then apply the Kupiec test and the Christoffersen test on the results to determine whether they are reliable.

We also specify a fixed effects regression on Spire's 10 minute returns over ten days in individual stocks.

We have the following regression model:

$$y_{it} = \beta_0 + \beta_1 \times return_{percent} + \beta_2 \times matched + \beta_3 \times highlow + + \beta_4 \times ticksize + \beta_5 \times EURIBOR + \beta_6 \times vstoxx + \beta_7 \times volume_change + \beta_8 \times trade + \varepsilon_i$$

of course, we use robust standard errors to correct for the inherent heteroscedasticity of the returns.

Results

We found that tick size had the greatest correlation with the return, and the variable was significant at the 5 % significance level. This was the only variable that was significant, and all in all the explanatory power was very low with an R-squared at only 5 %. We therefore quickly abandoned this path of research.

To determine the risk of Spire Europe's trading on the NASDAQ OMX Nordic exchanges we calculate the VaR using two different specifications. Historical simulation constantly underestimates the VaR compared to realized returns. Using the Kupiec test we reject the null hypothesis $H_0: \hat{\alpha} = \alpha$ for all our tested combinations. The Historical simulation model underestimate the VaR regardless if we use 100 periods of simulation or up to 4000 for one-minute simulations of IVaR. Different α-values and time frames still yield the same underestimation.

Interval Length [min]	Periods used For simulation	α	ˆα	Kupiec	Christoffersen
1	100	0.01	0.40	0.00	0.00
1	100	0.05	0.45	0.00	0.00
1	100	0.10	0.46	0.00	0.00
1	4000	0.01	0.36	0.00	0.00
1	4000	0.05	0.38	0.00	0.00
1	4000	0.10	0.39	0.00	0.00
10	100	0.01	0.20	0.00	0.01
10	100	0.05	0.38	0.00	0.38
10	200	0.01	0.25	0.00	0.30
10	200	0.05	0.36	0.00	0.00
10	400	0.01	0.24	0.00	0.55
10	400	0.05	0.33	0.00	0.01

Table 2. The IVaR for different intervals and α-levels. $\hat{\alpha}$ is the, from the results, estimated α-value.

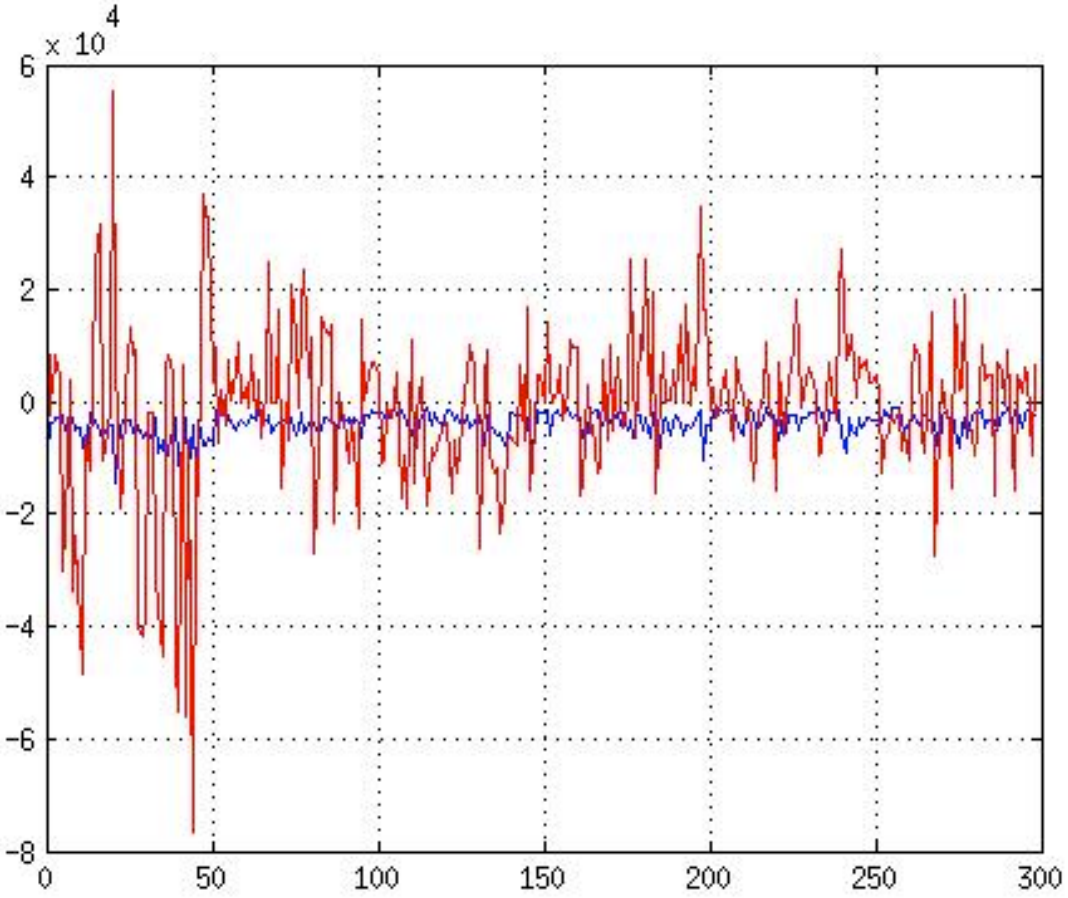


Figure 1: Historical Simulation of VaR with 10 minute intervals and 200 intervals used for simulation. The blue line is the estimated VaR in SEK. The red line is the realized return in SEK.

When using filtered Historical simulation the estimations of VaR are too high. Here, the Kupiec test also reject $\hat{\alpha} = \alpha$ but the estimated $\hat{\alpha}$ is lower that the simulated significance level. For one minute intervals we get $\hat{\alpha}_{0.10} = 0.0033$, $\hat{\alpha}_{0.05} = 0.0000$ and $\hat{\alpha}_{0.01} = 0.0013$.

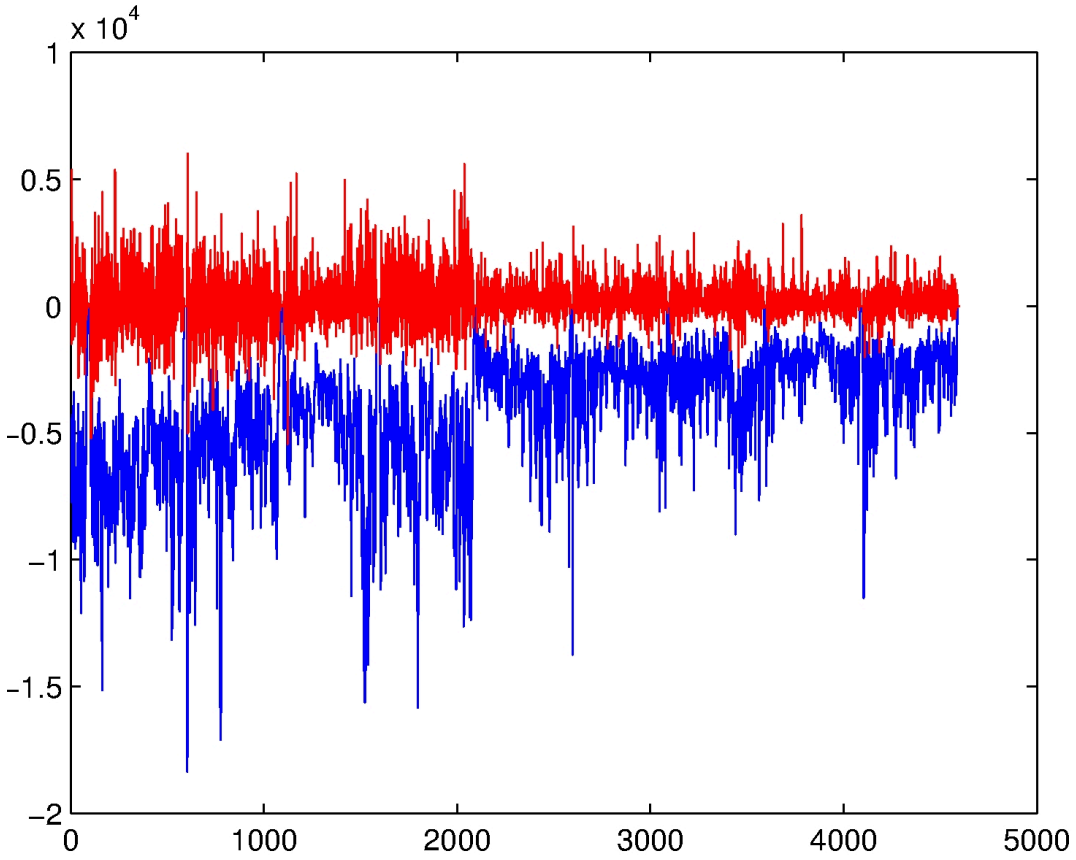


Figure 2: Filtered Historical simulation in red is the realized return in SEK and the blue is the VaR in SEK.

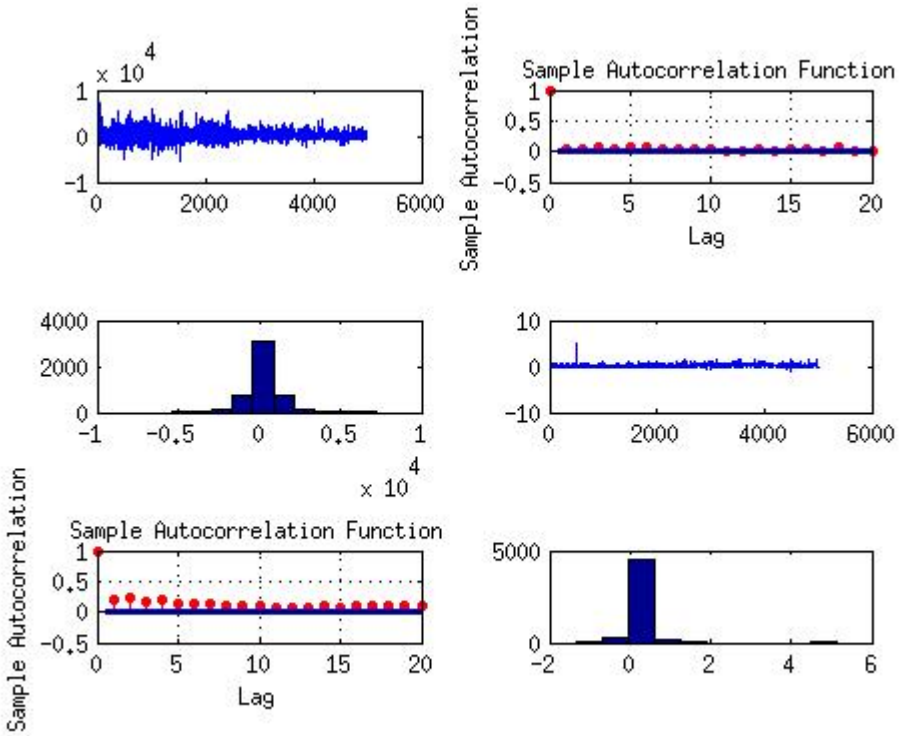


Figure 3: The figures are for one minute intervals, from left to right: return (SEK), autocorrelation return (SEK), distribution return (SEK), return (%), autocorrelation return (%), distribution return (%).

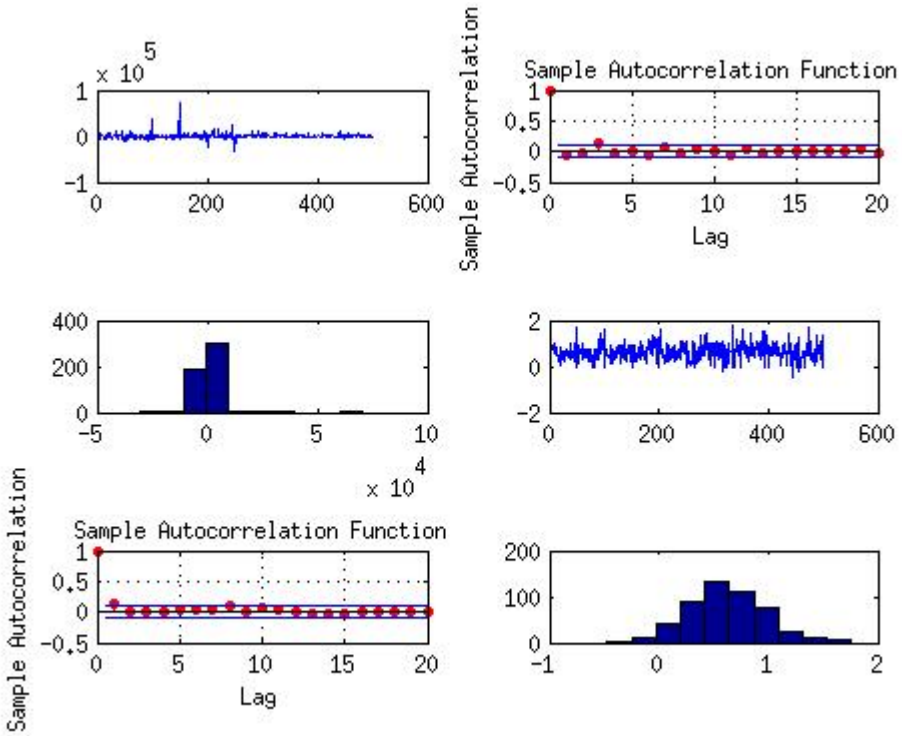


Figure 4: The figures are for ten minute intervals, from left to right: return (SEK), autocorrelation return (SEK), distribution return (SEK), return (%), autocorrelation return (%), distribution return (%).

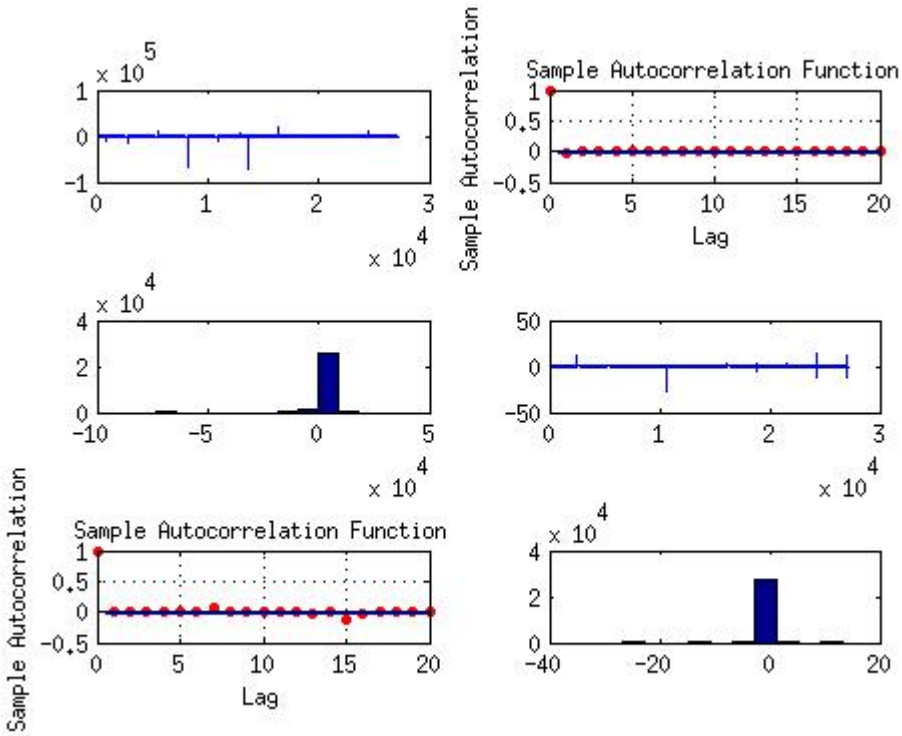


Figure 5: The figures are for ten second intervals, from left to right: return (SEK), autocorrelation return (SEK), distribution return (SEK), return (%), autocorrelation return (%), distribution return (%).

	One minute		**Ten Minutes**		**Ten Seconds**	
	Absolute	**Relative**	**Absolute**	**Relative**	**Absolute**	**Relative**
Min	-5430.30	-1.31	-30803.00	-0.47	-72926.00	-26.57
Max	7160.70	5.13	70148.00	1.76	17599.00	13.34
Mean	247.78	0.17	367.74	0.63	42.02	0.03
Median	175.46	0.12	0.00	0.62	2.07	0.01
Std	989.80	0.21	5285.20	0.35	748.19	0.27
Var	979710.00	0.05	27933000.00	0.12	559780.00	0.07
Skewness	0.30	4.01	4.90	0.25	-59.46	-30.13
Kurtosis	6.93	66.42	70.31	3.31	5599.80	4229.60
Total	1238600.00		183500.00		1145000.00	
Number of positive returns	3067.00	4420.00	248.00	476.00	13716.00	18130.00
Number of negative	1817.00	468.00	241.00	13.00	10706.00	6452.00

returns						

Table 3: Return Spire for different interval lengths.

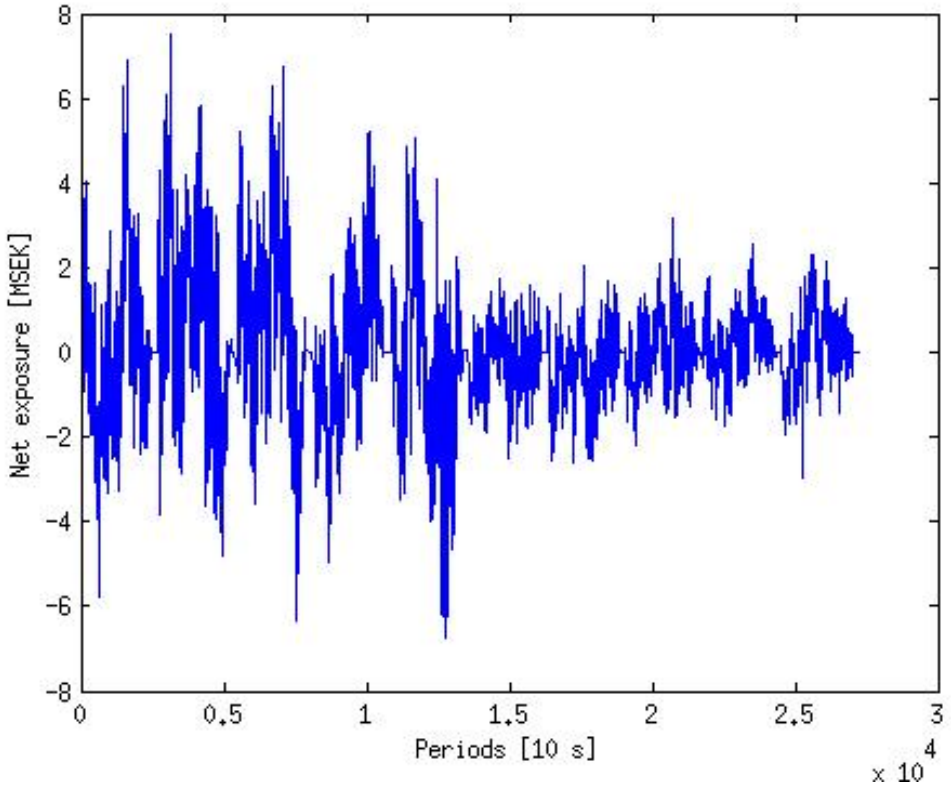


Figure 6: Spire Europe's net market exposure million SEK.

Analysis

Consider the diagrams describing ten minute, one minute and ten second returns. As the time interval gets shorter we notice that the returns behave less and less like if they were normally distributed. We can also see that the autocorrelation increases with smaller intervals and that the kurtosis clearly is too high for the returns to be normally distributed. Due to the volatile nature of the returns we have not been able to find a satisfying estimate for risk. It is, however, possible to conclude that Spire keeps a rather low market exposure which is limiting risk. The fact that they take both long and short positions is decreasing risk in the sense that it makes Spire less sensitive to general

market movements. On the other hand, as always, short positions expose you to potentially unlimited price risk.

Conclusions

Our aim with this thesis was to determine the behaviour of a high frequency trading firms' risk and return characteristics. We found almost no prediction power at all on when Spire is are trading. That indicates that they use much more advanced models to detect trading opportunities. Since limit orders seem to play a big role in their strategies it is unfortunate that you cannot determine from the data feed who is adding orders, only who is executing them. Otherwise one might have been able to find the latency between the time an order gets added to the book and the time it gets executed. We think that such information will exposes a strategy where the HFT firm is posting orders at the market while measuring the time it takes to get filled. By measuring the latency the HFT firm tries to determine whether there exists a hidden order book imbalance in the form of iceberg orders or dark pools. One could also speculate that advanced machine learning and pattern recognition algorithms probably can be tuned to detect execution algorithms such as VWAP trying to execute big orders in the market.

We think that it is a bit of a misconception that high frequency trading is a black box where you load your algorithms and then let it trade completely unsupervised. Qualitative inspection of data reveals that Spire does not trade some stock for long periods of time (by long we mean hours). In the talks we have had with APS Capital they confirmed that algorithms is taken down and re-tuned to better fit current market dynamics.

It is, as mentioned above, hard to find a good risk estimate for the trading of HFT firms. Anecdotal evidence suggest a simple "pull the plug" type of risk management. It quickly tries to get out of current positions and take algorithms off the market if they start losing money. In light of our results we conjecture that factors that determine relative competitiveness among HFT firms are not as much risk taking as much as computer power (quite easy to obtain as long as adequate funding is available), skill (the ability develop algorithms) and experience (to be able to know what works).

We would also like to propose some ideas for future research related to this work as well as alternative research approaches to this subject in particular. It would have been nice to have more data when we were trying to analyse the behaviour of Spire. Since we

were kindly supplied with data, the limitation was not in getting data but our ability to process it. We were constantly forced to cut down on data range and data resolution in order to be able to process it. From the data supplied we were, for example, able to rebuild the order book. We did, however, not have the computational power to fully analyse the dynamics of them. Instead we had to settle with using the top of the order books.

A more thorough examination would use machine learning and pattern recognition to try to reverse-engineer how the algorithms approximately work. This would however present difficulties, not only of computational complexity, but also by the fact that it is only possible to identify the participants when an order is executed and not when an order is placed. Extending the research to other firms and markets could perhaps help determine the characteristics of markets especially well-suited for high frequency trading.

Finally, in the section where we present the theoretical background, the references are not always published in the best journals. Some of them are even working papers. This has been a problem for us from the very beginning and we think that the lack of quality research is due to the fact that this is a very young research field where there is little empirical research published applicable to this thesis. To give a somewhat adequate presentation of the current state of the field we had to go beyond most of the prestigious financial journals. Surely, as the field matures more articles will be published and that will hopefully make it easier to find empirical research on many diverse topics on high frequency trading in top ranked journals.

References

Aldridge, Irene. 2010. High-Frequency Trading, John Wiley & Sons, Inc.

Almeiday. C., Vicente. J., (2008) Are Interest Rate Options Important for the Assessment of Interest Rate Risk? Journal of Finance Working Paper Series.

Amicangioli, Anthony. 2006. Financial tech drawing an unlikely engineering crowd. Mass High Tech Business News. http://www.masshightech.com/stories/2006/07/31/focus3-Financial-tech-drawing-an-unlikely-engineering-crowd.html. Downloaded 2012-03-23.

Barone-Adesi, G., Giannopoulos, K., Vosper, L. 1999. VaR without Correlations for Non-Liner Portfolios. vol. 19, Journal of Futures Markets.

Biais, Bruno, Foucault, Thierry and Moinas, Sophie. 2011. Equilibrium High Frequency Trading.

Bursell, Jacob. 2012. Mannen som hackade Wall Street. SvD Näringliv. http://www.svd.se/naringsliv/mannen-som-hackade-wall-street_6804315.svd. Downloaded 2012-04-03.

Cartea, Álvaro and Penelva, José. 2012. Where is the Value in High Frequency Trading. Working Paper.

Cave, Tim. 2011a. Europe's top 10 high-frequency kingmakers (in Scandinavia, at least). Financial News. http://www.efinancialnews.com/story/2011-10-03/the-top-ten-europe-high-frequency-kingmakers-in-scandinavia-at-least?mod=article-related. Downloaded 2012-02-12.

Cave, Tim. 2011b. Spire sprints to high-speed trading success. Financial News. http://www.efinancialnews.com/story/2011-10-06/who-is-spire-europe. Downloaded 2012-03-04.

Cave, Tim. 2012. High frquency trading doubles in Nordic markets. Market Watch. http://www.marketwatch.com/story/high-frquency-trading-doubles-in-nordic-markets-2012-01-05. Downloaded 2012-05-12.

Clark, Carol J. 2010. Controlling risk in a lightning-speed trading environment. March, No. 272, Federal Reserve Bank of Chicago. http://www.chicagofed.org/digital_assets/publications/chicago_fed_letter/2010/cflmarch2010_272.pdf. Downloaded 2012-05-14.

Dionne. G, Duchesne. P, Pacurar. M (2005), Intraday Value at Risk (IVaR) Using Tick-by-Tick Data with Application to the Toronto Stock Exchange, Journal of Empirical Finance.

Engle R. F., Russell., J.R. (1998) Autoregressive Conditional Duration: A New Model for Irregularly Spaced Transaction Data, Econometrica, 66:1127-1162, 1998.

Giot, Pierre. 2011. Market risk models for intraday data. European Journal of Finance.

Hult, Henrik, Lindskog. Filip. 2007. Lecture Notes Mathematical Modelling and Statistical Methods for Risk Management, KTH

Iati, Rob. 2009. The real story of trading software espionage. Advanced Trading. http://www.advancedtrading.com/algorithms/218401501. Downloaded 2012-03-24.

McMillan, D.G, Speight, A.E.H, Evans, P.K, (2007) How useful is intraday data for evaluating daily Value-at-Risk? Evidence from three Euro rates. Journal of Multinational Financial Management

Mehta, Nina. 2011. Wedbush Buys Lime Brokerage to Help Meet Market-Access Rules. Bloomberg Businessweek. http://www.businessweek.com/news/2011-06-14/wedbush-buys-lime-brokerage-to-help-meet-market-access-rules.html. Downloaded 2012-02-07.

NASDAQ OMX. 2011. NASDAQ OMX Nordic Market Model 2.6. http://nordic.nasdaqomxtrader.com/digitalAssets/75/75615_nasdaqomxnordicmarketmodel2.6.pdf. Downloaded 2012-01-06.

Plambeck, Joseph. 2010. Idea Man of Limewire at Crossroads. The New York Times. http://www.nytimes.com/2010/05/24/business/media/24limewire.html?_r=4. Downloaded 2012-02-25.

Qi, Jun. 2011. Risk Measurement with High-frequency Data – Value-at-risk and Scaling Law Methods. University of Essex.

Tower Research Capital LLC. 2012a. http://www.tower-research.com/. Downloaded 2012-01-29.

Appendix

Parse ITCH-files and reducing to stocks Spire trading in:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import sys
import os
import glob
from decimal import *
import re

def WhichOrderBooks(datafile):

        orders = set()

        datafile.seek(0)
        for row in datafile.readlines():
            if row.find('SRE') != -1:
                if row[0] == 'E' or row[0] == 'C':
                    orders.add(int(row[1:10]))

                if row[0] == 'P':
                    which_order_books.add(int(row[20:26]))

        datafile.seek(0)
        for row in datafile.readlines():
            if row[0] == 'A' or row[0] == 'F':
                if int(row[1:10]) in orders:
                    which_order_books.add(int(row[20:26]))

datafile = None
path = 'data/'

which_order_books = set()

try:
```

```
        for infile in glob.glob(os.path.join(path, '*.log')):
                datafile = open(infile,'r')
                WhichOrderBooks(datafile)

        print "which_order_books = "
        for i in which_order_books:
                print "%s," % i
        print "]"

except IOError as (e, s):
        print  "I/O Error({0}): {1}".format(e, s)
        sys.exit(1)

finally:
        if datafile:
                datafile.close()
```

Parse the ITCH-files and reducing them to only the stocks traded by Spire:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import sys
import os
import glob
from decimal import *
import re
import gc
```

```
class Order_Book_Directory(object):
        time = 0

        message_type = 'R'
        order_book = 0
        symbol = ""
        financial_product = 0
        trading_currency = ""
        market_segment = 0

        def __init__(self, time, order_book, symbol, financial_product, trading_currency,
market_segment):
                self.time = time

                self.order_book = order_book
                self.symbol = symbol
                self.financial_product = financial_product
                self.trading_currency = trading_currency
                self.market_segment = market_segment

        def __hash__(self):
                return hash(self.order_book)

        def __str__(self):
                return self.message_type + ',' + str(self.time) + ',' + str(self.order_book)
+ ',' + self.symbol + ',' + str(self.financial_product) + ',' + self.trading_currency + ',' +\
                        str(self.market_segment)



class Add_Order(object):
        time = 0
        message_type = 'A'
        order_reference_number = 0
```

```
        buy_sell_indicator = ""
        quantity = 0
        order_book = 0
        price = 0.0

        def __init__(self, time, order_reference_number, buy_sell_indicator, quantity, order_book, price):
                self.time = time
                self.order_reference_number = order_reference_number
                self.buy_sell_indicator = buy_sell_indicator
                self.quantity = quantity
                self.order_book = order_book
                self.price = price

        def __hash__(self):
                return hash(self.order_reference_number)

        def __str__(self):
                return self.message_type + ',' + str(self.time) + ',' + str(self.order_reference_number) + ',' + self.buy_sell_indicator + ',' + str(self.quantity) + ',' + str(self.price)

class Order_Executed(object):
        time = 0
        message_type = 'EC'
        order_reference_number = 0
        executed_quantity = 0
        match_number = 0
        trade_price = 0.0
        buyer_id = ""
        seller_id = ""
        order_book = 0
```

```
        def __init__(self, time, order_reference_number, executed_quantity, match_number, trade_price, buyer_id, seller_id, order_book):
                self.time = time
                self.order_reference_number = order_reference_number
                self.executed_quantity = executed_quantity
                self.match_number = match_number
                self.trade_price = trade_price
                self.buyer_id = buyer_id
                self.seller_id = seller_id
                self.order_book = order_book

        def __hash__(self):
                return hash(self.order_reference_number)

        def __str__(self):
                return self.message_type + ',' + str(self.time) + ',' + str(self.order_reference_number) + ',' + str(self.executed_quantity) + ',' + str(self.match_number) + ',' +\
                        str(self.trade_price) + ',' + self.buyer_id + ',' + self.seller_id


class Trade_Message_Non_Cross(object):
        time = 0
        message_type = 'P'
        order_reference_number = 0
        quantity = 0
        order_book = 0
        match_number = 0
        trade_price = 0.0
        buyer_id = ""
        seller_id = ""

        def __init__(self, time, order_reference_number, quantity, order_book, match_number, trade_price, buyer_id, seller_id):
```

```
                self.time = time
                self.order_reference_number = order_reference_number
                self.quantity = quantity
                self.order_book = order_book
                self.match_number = match_number
                self.trade_price = trade_price
                self.buyer_id = buyer_id
                self.seller_id = seller_id

        def __hash__(self):
                return hash(self.order_reference_number)

        def __str__(self):
                return self.message_type + ',' + str(self.time) + ',' + str(self.order_reference_number) + ',' + str(self.quantity) + ',' + str(self.match_number) + ',' +\
                        str(self.trade_price) + ',' + self.buyer_id + ',' + self.seller_id



class Order_Cancel_Delete(object):
        time = 0
        message_type = 'XD'
        order_reference_number = 0
        cancel_quantity         = 0
        order_book = 0

        def __init__(self, time, order_reference_number, cancel_quantity, order_book):
                self.time = time
                self.order_reference_number = order_reference_number
                self.cancel_quantity = cancel_quantity
                self.order_book

        def __hash__(self):
```

```
                return hash(self.order_reference_number)

        def __str__(self):
                return self.message_type + ',' + str(self.time) + ',' + str(self.order_reference_number) + ',' + str(self.cancel_quantity)


class Broken_Trade_Message(object):
        time = 0
        message_type = 'B'
        match_number = 0
        order_book = 0

        def __init__(self, time, match_number, order_book):
                self.time = time
                self.match_number = match_number
                self.order_book

        def __hash__(self):
                return hash(self.match_number)

        def __str__(self):
                return self.message_type + ',' + str(self.time) + ',' + str(self.match_number)

def Price(price):

        whole_number = Decimal(price[0:6])
        decimal_digits = Decimal(price[6:10])
        price = Decimal(whole_number + decimal_digits/10000)
        return price


def Parser(datafile):
```

```
        current_time = 0
        seconds = 0

        #iteration = 0

        datafile.seek(0)
        for row in datafile.readlines():
                #iteration += 1
                #if iteration % 10000000:
                #       gc.collect()

                #Manage time
                if row[0] == 'O':
                        for book in which_order_books:
                                segment_id = int(row[1:4])
                                event_code = str(row[4]).strip('\n')
                                exec "f_%s.write('O,%d,%s\\n')" % (book, segment_id,
event_code)


                if row[0] == 'T':
                        seconds = int(row[1:6])*1000
                        current_time = seconds

                if row[0] == 'M':
                        current_time = seconds + int(row[1:4])

                #Fill Order Books
                if row[0] == 'R':
                        time = current_time
                        order_book = int(row[1:7])
                        symbol = str(row[7:23])
                        financial_product = int(row[35:38])
```

```
                trading_currency = str(row[38:41])
                market_segment = int(row[45:48])
                if order_book in which_order_books:
                        order_book_directories[order_book] =
Order_Book_Directory(time, order_book, symbol, financial_product, trading_currency,
market_segment)
                        exec "f_%s.write('%s\\n')" % (order_book,
order_book_directories[order_book])

        #reduced_set_of_oder_books[book].append(order_book_directories[order_boo
k])



        if row[0] == 'A' or row[0] == 'F':
                time = current_time
                order_reference_number = int(row[1:10])
                buy_sell_indicator = str(row[10:11])
                quantity = int(row[11:20])
                order_book = int(row[20:26])
                price = Price(str(row[26:36]))
                if order_book in which_order_books:
                        added_orders[order_reference_number] =
Add_Order(time, order_reference_number, buy_sell_indicator, quantity, order_book,
price)
                        exec "f_%s.write('%s\\n')" % (order_book,
added_orders[order_reference_number])

        #reduced_set_of_oder_books[book].append(added_orders[order_reference_nu
mber])

        if row[0] == 'E':
                time = current_time
                order_reference_number = int(row[1:10])
```

```
                try:
                        order_book =
added_orders[order_reference_number].order_book
                except KeyError:
                        continue
                if order_book not in which_order_books:
                        continue
                executed_quantity = int(row[10:19])
                match_number = int(row[19:28])
                owner_id = str(row[28:32])
                counterpart_id = str(row[32:36])
                trade_price = added_orders[order_reference_number].price
                buy_sell_indicator =
added_orders[order_reference_number].buy_sell_indicator
                if buy_sell_indicator == 'B':
                        buyer_id = owner_id
                        seller_id = counterpart_id
                else:
                        buyer_id = counterpart_id
                        seller_id = owner_id
                orders_executed[order_reference_number] =
Order_Executed(time, order_reference_number, executed_quantity, match_number,
trade_price, buyer_id, seller_id, order_book)
                exec "f_%s.write('%s\\n')" % (order_book,
orders_executed[order_reference_number])

        #reduced_set_of_oder_books[book].append(orders_executed[order_reference_n
umber])

        if row[0] == 'C':
                time = current_time
                order_reference_number = int(row[1:10])
                try:
```

```
                    order_book =
added_orders[order_reference_number].order_book
                except KeyError:
                    continue
                if order_book not in which_order_books:
                    continue
                executed_quantity = int(row[10:19])
                match_number = int(row[19:28])
                trade_price = Price(str(row[29:39]))
                owner_id = str(row[39:43])
                counterpart_id = str(row[43:47])
                buy_sell_indicator =
added_orders[order_reference_number].buy_sell_indicator
                if buy_sell_indicator == 'B':
                    buyer_id = owner_id
                    seller_id = counterpart_id
                else:
                    buyer_id = counterpart_id
                    seller_id = owner_id
                orders_executed[order_reference_number] =
Order_Executed(time, order_reference_number, executed_quantity, match_number,
trade_price, buyer_id, seller_id, order_book)
                exec "f_%s.write('%s\\n')" % (order_book,
orders_executed[order_reference_number])

        #reduced_set_of_oder_books[book].append(orders_executed[order_reference_n
umber])

            if row[0] == 'P':
                time = current_time
                order_reference_number = int(row[1:10])
                quantity = int(row[11:20])
                order_book = int(row[20:26])
                match_number = int(row[26:35])
```

```
                trade_price = Price(str(row[35:45]))
                buyer_id = str(row[45:49])
                seller_id = str(row[49:53])
                if order_book in which_order_books:
                        non_cross_trades[order_reference_number] = Trade_Message_Non_Cross(time, order_reference_number, quantity, order_book, match_number, trade_price, buyer_id, seller_id)
                        exec "f_%s.write('%s\\n')" % (order_book, non_cross_trades[order_reference_number])

        #reduced_set_of_oder_books[book].append(non_cross_trades[order_reference_number])

        if row[0] == 'X':
                time = current_time
                order_reference_number = int(row[1:10])
                cancel_quantity = int(row[10:19])
                try:
                        order_book = added_orders[order_reference_number].order_book
                except KeyError:
                        continue
                if order_book not in which_order_books:
                        continue
                order_canceled = Order_Cancel_Delete(time, order_reference_number, cancel_quantity, order_book)
                exec "f_%s.write('%s\\n')" % (order_book, order_canceled)
                #order_canceled[order_reference_number] = Order_Cancel_Delete(time, order_reference_number, cancel_quantity, order_book)
                #exec "f_%s.write('%s\\n')" % (order_book, order_canceled[order_reference_number])

        #reduced_set_of_oder_books[book].append(order_canceled[order_reference_number])
```

```
                if row[0] == 'D':
                        time = current_time
                        order_reference_number = int(row[1:10])
                        try:
                                order_book =
added_orders[order_reference_number].order_book
                        except KeyError:
                                continue
                        if order_book not in which_order_books:
                                continue
                        cancel_quantity =
added_orders[order_reference_number].quantity
                        order_canceled = Order_Cancel_Delete(time,
order_reference_number, cancel_quantity, order_book)
                        exec "f_%s.write('%s\\n')" % (order_book, order_canceled)
                        #order_canceled[order_reference_number] =
Order_Cancel_Delete(time, order_reference_number, cancel_quantity, order_book)
                        #exec "f_%s.write('%s\\n')" % (order_book,
order_canceled[order_reference_number])

        #reduced_set_of_oder_books[book].append(order_canceled[order_reference_nu
mber])

                if row[0] == 'B':
                        time = current_time
                        match_number = int(row[1:10])
                        order_book = None
                        for k, v in orders_executed.iteritems():
                                if v.match_number == match_number:
                                        order_book = v.order_book
                        if not order_book:
                                for k, v in non_cross_trades.iteritems():
                                        if v.match_number == match_number:
```

```
                                        order_book = v.order_book
                    if (order_book != None) and (order_book in which_order_books):
                                        broken_trades[order_reference_number] = Broken_Trade_Message(time, match_number, order_book)
                                        exec "f_%s.write('%s\\n')" % (order_book, broken_trades[order_reference_number])

        #reduced_set_of_oder_books[book].append(broken_trades[order_reference_number])

datafile = None
path = 'data/'

try:

        for infile in glob.glob(os.path.join(path, '*.log')):
                datafile = open(infile,'r')
                which_order_books = [3201,1154,1027,261,1158,3207,812,24312,75663,401,402,46,281,283,285,77855,161,300,45,39854,29981,1586,15285,\

        11273,3258,1596,24394,22335,4928,323,3524,18634,81,340,3415,220,992,24302,101,5095,361,366,120,3966]
                #reduced_set_of_oder_books = dict()
                order_book_directories = dict()
                added_orders = dict()
                orders_executed = dict()
                non_cross_trades = dict()
                broken_trades = dict()
                gc.collect()
                print infile
                print len(which_order_books)
                for book in which_order_books:
```

```
                exec "f_%s=open('data/reduced3/%s_%s.log','w')" % (book,
re.sub('\D', '', infile), book)
                Parser(datafile)


except IOError as (e, s):
        print "I/O Error({0}): {1}".format(e, s)
        sys.exit(1)

finally:
        if datafile:
                datafile.close()
```

File for extracting key statistics, several similar files for slightly different use:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import sys
import os
import glob
from decimal import *
import re
from collections import deque
import scipy as sp
from scipy import stats
import numpy as np
import matplotlib as mpl
from matplotlib import pyplot as plt
```

```
class Position(object):
        price = 0.0
        quantity = 0

        def __init__(self, price, qty):
                self.price = price
                self.qty = qty

class Asset(object):
        name = ""
        asset_id = ""
        date = ""
        volume_interval = 0
        volume_non_matched = 0
        volume_spire = 0
        volume_sell = 0
        return_spire_sek = 0.0
        return_spire_percent = 0.0
        turnover_spire_non_matched = 0.0
        turnover_spire_sek = 0.0
        turnover_sell = 0.0
        interval_opening_sek = 0.0
        interval_close_sek = 0.0
        interval_high_sek = 0.0
        interval_low_sek = 0.0
        long_positions = list()
        short_positions = list()
        interval = 0
        currency = ''
        long_qty = 0
        short_qty = 0
        net_qty = 0
        block1 = 0 # 500.000
        block2 = 0 # 1.000.000
```

```
block3 = 0 # 5.000.000
prices = list()
std = 0.0
var = 0.0
nb_of_trades = 0


def __init__(self, name, date, asset_id, currency):
        self.name = name
        self.date = date
        self.asset_id = asset_id
        self.currency = currency

def __str__(self):
        return self.name +','\
                +self.asset_id +','\
                +self.date +','\
                +str(self.volume_interval) +','\
                +str(self.volume_non_matched) +','\
                +str(self.volume_spire) +','\
                +str(self.volume_sell)+','\
                +str(self.return_spire_sek) +','\
                +str(self.return_spire_percent) +','\
                +str(self.turnover_spire_non_matched)+','\
                +str(self.turnover_spire_sek) +','\
                +str(self.turnover_sell)+','\
                +str(self.interval_opening_sek) +','\
                +str(self.interval_close_sek) +','\
                +str(self.interval_high_sek) +','\
                +str(self.interval_low_sek) +','\
                +str(self.interval) +','\
                +str(self.currency) +','\
                +str(self.net_qty) +','\
                +str(self.nb_of_trades) +','\
```

```
                +str(self.block1) +','\
                +str(self.block2) +','\
                +str(self.block3) +','\
                +str(self.std) +','\
                +str(self.var)+'\n'



    def buys(self, buy_price, qty):
            self.net_qty += qty
            self.long_qty += qty
            self.long_positions.append(Position(buy_price, qty))

    def sells(self, sell_price, qty):
            self.net_qty -= qty
            self.short_qty += qty
            self.short_positions.append(Position(sell_price, qty))

    def initialize(self):
            self.volume_interval = 0
            self.volume_non_matched = 0
            self.volume_spire = 0
            self.volume_sell = 0
            self.return_spire_sek = 0.0
            self.return_spire_percent = 0.0
            self.turnover_spire_sek = 0.0
            self.turnover_spire_non_matched = 0.0
            self.turnover_sell = 0
            self.interval_opening_sek = 0.0
            self.interval_close_sek = 0.0
            self.interval_high_sek = 0.0
            self.interval_low_sek = 0.0
            self.nb_of_trades = 0
            self.block1 = 0
```

```
        self.block2 = 0
        self.block3 = 0
        self.prices = list()
        self.std = 0
        self.var = 0
        self.volume_non_matched = 0
        self.turnover_spire_non_matched = 0




def finalize(self, opening, close, high, low):

        sum_buy = 0
        sum_sell = 0
        if self.net_qty >= 0:
                match_qty = self.short_qty
                self.long_qty -= self.short_qty
                self.short_qty = 0


        else:
                match_qty = self.long_qty
                self.short_qty -= self.long_qty
                self.long_qty = 0


        qty_left  = match_qty
        for i in self.long_positions:

                if i.qty >= qty_left:
                        i.qty -= qty_left
                        sum_buy += i.price * qty_left
                        qty_left = 0
```

```
            elif i.qty < qty_left:
                    sum_buy += i.price * i.qty
                    qty_left -= i.qty
                    i.qty = 0
            if not qty_left:
                    break

qty_left = match_qty
for i in self.short_positions:

            if i.qty >= qty_left:
                    i.qty -= qty_left
                    sum_sell += i.price * qty_left
                    qty_left = 0
            elif i.qty < qty_left:
                    sum_sell += i.price * i.qty
                    qty_left -= i.qty
                    i.qty = 0
            if not qty_left:
                    break



remove = list()
for i in self.long_positions:
            if i.qty == 0:
                    remove.append(i)

for i in remove:
            self.long_positions.remove(i)

remove = list()
for i in self.short_positions:
            if i.qty == 0:
                    remove.append(i)
```

```
                for i in remove:
                        self.short_positions.remove(i)

                self.volume_spire = match_qty
                self.return_spire_sek = sum_sell - sum_buy
                self.turnover_spire_sek = sum_buy
                self.interval_opening_sek = opening
                self.interval_close_sek = close
                self.interval_high_sek = high
                self.interval_low_sek = low

                if self.currency == 'DKK':
                        self.return_spire_sek *= 1.19
                        self.turnover_spire_sek *= 1.19
                        self.interval_opening_sek *= 1.19
                        self.interval_close_sek *= 1.19
                        self.interval_high_sek *= 1.19
                        self.interval_low_sek *= 1.19
                        self.turnover_spire_non_matched *= 1.19
                        self.turnover_sell *= 1.19
                elif self.currency == 'EUR':
                        self.return_spire_sek *= 8.85
                        self.turnover_spire_sek *= 8.85
                        self.interval_opening_sek *= 8.85
                        self.interval_close_sek *= 8.85
                        self.interval_high_sek *= 8.85
                        self.interval_low_sek *= 8.85
                        self.turnover_spire_non_matched *= 8.85
                        self.turnover_sell *= 8.85

                if self.turnover_spire_sek:
                        self.return_spire_percent                                       =
self.return_spire_sek/self.turnover_spire_sek
```

```
                if len(self.prices) == 0:
                        self.prices.append(self.interval_close_sek)

                p = np.array(self.prices)
                self.std = np.std(p)
                self.var = np.var(p)

                data = (self.name, self.date, self.asset_id, self.interval, self.prices, self.net_qty, self.return_spire_sek, self.turnover_spire_sek)

                matlab.append(data)

def Parse(infile, outfile, date):

        broken_trades = set()

        infile.seek(1)
        for row in infile:
                srow = row.split(',')
                if srow[0] == 'B':
                        broken_trades.add(int(srow[2]))



        time_interval = 10 * 60000 # 60000 = ms/min
        #time_interval = 10000000000000000
        infile.seek(0)
        row = datafile.readline().split(',')
        name = re.sub('\W', '', row[3])
        asset_id = re.sub('\W', '', row[2])
        currency = re.sub('\W', '', row[5])
        asset = Asset(name, date, asset_id, currency)
        segment = int(row[6])
        #row = datafile.readline().split(',')
```

```
EOF = False
i = 1

while True:
        row = datafile.readline().split(',')
        if row[0] == 'O':
                if int(row[1]) == segment and re.sub('\W', '', row[2]) == 'P':
                        break

while True:
        row = datafile.readline().split(',')
        if row[0] != 'O':
                time = int(row[1])
                break
close = 0.0

while True:
        first = True
        opening = 0.0
        high = 0.0
        low = 0.0
        time_stop = time + time_interval
        asset.interval = i
        asset.initialize()
        i += 1
        j = 0
        while True:
                j += 1


                srow = infile.readline().split(',')


                if srow[0] == 'O':
```

```
                    if int(srow[1]) == segment and re.sub('\W', '', srow[2])
== 'C':
                        EOF = True
                        break
                    continue

                if len(srow) < 2:
                    EOF = True
                    break

                time = int(srow[1])

                if srow[0] == 'EC' or srow[0] == 'P':
                    if int(srow[4]) in broken_trades:
                        continue
                    price = float(srow[5])
                    qty = int(srow[3])
                    asset.volume_interval += qty
                    close = price
                    asset.prices.append(price)
                    if re.sub('\W', '', srow[6]) == 'SRE':  #SRE buys
                        asset.buys(price, qty)
                        asset.nb_of_trades += 1
                        asset.volume_non_matched += qty
                        asset.turnover_spire_non_matched  +=  qty*price

                    if re.sub('\W', '', srow[7]) == 'SRE': #SRE Sells
                        asset.sells(price, qty)
                        asset.nb_of_trades += 1
                        asset.volume_sell += qty
                        asset.turnover_sell += qty*price
                    if first:
                        opening = price
                        high = price
```

```
                low = price
                first = False
            if price > high:
                high = price
            if price < low:
                low = price
            value = price * qty
            if asset.currency == 'DKK':
                value *= 1.19
            if asset.currency == 'EUR':
                value *= 8.85
            #print value
            if value > 500000:
                asset.block1 += 1
                if value > 1000000:
                    asset.block2 += 1
                    if value > 5000000:
                        asset.block3 += 1


        if time >= time_stop:
            break

asset.finalize(opening, close, high, low)
outfile.write(str(asset))


if EOF:
        break
```

```
def PrintM(outfile):

    data_sorted = sorted(matlab, key=lambda element: (element[1], element[3], element[0]))
    print len(data_sorted)

    outfile.write("prices_data = {","")
    for i in range(44):
        outfile.write(",'%s'" % data_sorted[i][0])
    outfile.write(";\n")

    outfile.write("","")
    for i in range(44):
        outfile.write(",%s" % data_sorted[i][2])
    outfile.write(";\n")

    for i in xrange(0,22440,44):
        outfile.write("%s,%s" % (data_sorted[i][1], data_sorted[i][3]))
        for j in range(44):
            outfile.write(",%s" % data_sorted[i+j][4])
        outfile.write(";\n")

    outfile.write("};\n")

    outfile.write("weights_data = {","")
    for i in range(44):
        outfile.write(",'%s'" % data_sorted[i][0])
    outfile.write(";\n")

    outfile.write("","")
    for i in range(44):
        outfile.write(",%s" % data_sorted[i][2])
    outfile.write(";\n")
```

```
for i in xrange(0,22440,44):
        outfile.write("%s,%s" % (data_sorted[i][1], data_sorted[i][3]))
        for j in range(44):
                outfile.write(",%s" % data_sorted[i+j][5])
        outfile.write(";\n")

outfile.write("};")

outfile.write("};\n")

outfile.write("return_data = {","")
for i in range(44):
        outfile.write(",'%s'" % data_sorted[i][0])
outfile.write(";\n")

outfile.write("","")
for i in range(44):
        outfile.write(",%s" % data_sorted[i][2])
outfile.write(";\n")

for i in xrange(0,22440,44):
        outfile.write("%s,%s" % (data_sorted[i][1], data_sorted[i][3]))
        for j in range(44):
                outfile.write(",%s" % data_sorted[i+j][5])
        outfile.write(";\n")

outfile.write("};")

outfile.write("};\n")

outfile.write("turnover_data = {","")
for i in range(44):
        outfile.write(",'%s'" % data_sorted[i][0])
```

```
        outfile.write(";\n")

        outfile.write("","")
        for i in range(44):
                outfile.write(",%s" % data_sorted[i][2])
        outfile.write(";\n")

        for i in xrange(0,22440,44):
                outfile.write("%s,%s" % (data_sorted[i][1], data_sorted[i][3]))
                for j in range(44):
                        outfile.write(",%s" % data_sorted[i+j][7])
                outfile.write(";\n")

        outfile.write("};")



#main
datafile = None
#path = 'data/buggbugg'
path = 'data/reduced3'
path_output = 'data/ten/'

matlab = list()

try:
        outfile = open('%sten.csv' % (path_output),'w')
        outfile_matlab = open('%sdata.m' % (path_output),'w')
        outfile.write("Name,ID,Date,Volume Market,Volume SRE,Volume matched,Volume sell,Return SRE SEK,Return SRE percent,Turnover SRE SEK,Turnover matched,Turnover sell,Interval Opening,Interval Close,Interval High,Interval Low,Interval Nr,Currency,Net Qty,Nb Trades,Over_500,Over_1m,Over5m,Std,Var\n")
        for infile in glob.glob(os.path.join(path, '*.log')):
                date = re.sub('\W', '', infile[13:22])
```

```
                print infile
                datafile = open(infile,'r')
                Parse(datafile, outfile, date)

        #PrintM(outfile_matlab)



except IOError as (e, s):
        print  "I/O Error({0}): {1}".format(e, s)
        sys.exit(1)

finally:
        if datafile:
                datafile.close()
```

For claculating Spire profit and loss:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import sys
import os
import glob
from decimal import *
import re

class Asset(object):

        order_reference_number = 0
        executed_quantity = 0
        match_number = 0
        trade_price = 0.0
```

```
        def __init__(self, order_reference_number, executed_quantity, match_number, trade_price):

                self.order_reference_number = int(order_reference_number)
                self.executed_quantity = int(executed_quantity)
                self.match_number = int(match_number)
                self.trade_price = float(trade_price)

def Parser(datafile):

        _buy = 0
        _sell = 0

        datafile.seek(0)
        for row in datafile:

                srow = row.split(',')
                if srow[0] == 'EC' or srow[0] == 'P':
                        srow[6] = re.sub('\W', '', srow[6])
                        srow[7] = re.sub('\W', '', srow[7])

                        if srow[6] == 'SRE':
                                buying.append(Asset(srow[2], srow[3], srow[4], srow[5]))
                                _buy += 1
                        if srow[7] == 'SRE':
                                selling.append(Asset(srow[2], srow[3], srow[4], srow[5]))
                                _sell += 1



                if srow[0] == 'B':
                        broken_trades.add(srow[2])

        print "Buy: %d\tSell: %d" % (_buy, _sell)
```

```
def Calculate(profit_loss, quantity):

        for asset in buying:
                if asset.match_number not in broken_trades:
                        profit_loss -= asset.executed_quantity * asset.trade_price
                        quantity += asset.executed_quantity

        for asset in selling:
                if asset.match_number not in broken_trades:
                        profit_loss += asset.executed_quantity * asset.trade_price
                        quantity -= asset.executed_quantity

        return profit_loss, quantity




#main
datafile = None
#path = 'data/reduced'
#path_output = 'data/profit/'
path = 'data/tempora'
path_output = 'data/ten/'



total_profit = 0.0
total_quantity = 0

try:

        outfile = open('%sprofit.log' % (path_output),'w')
        for infile in glob.glob(os.path.join(path, '*.log')):
```

```
                datafile = open(infile,'r')
                selling = list()
                buying = list()
                broken_trades = set()
                #Compensate for currency
                currency = datafile.readline().split(',')[5].strip(' \n\t')
                print currency
                Parser(datafile)
                profit_loss = 0.0
                quantity = 0
                (profit_loss, quantity) = Calculate(profit_loss, quantity)
                if currency == 'DKK':
                        profit_loss *= 1.19
                elif currency == 'EUR':
                        profit_loss *= 8.85
                outfile.write("%s,%f\n" % (infile[13:-4], profit_loss))
                total_profit += profit_loss
                total_quantity += quantity
                print "%s,%d" % (infile[13:-4], quantity)

        #outfile.write("%s Total P&L: %f\n" % (infile[13:21], total_profit))
        #outfile.write("%s Total Qty: %d\n" % (infile[13:21], total_quantity))
        #outfile.write("==================================
==========")

except IOError as (e, s):
        print  "I/O Error({0}): {1}".format(e, s)
        sys.exit(1)

finally:
        if datafile:
                datafile.close()
```

For generating top of order books

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import sys
import os
import glob
from decimal import *
import re
import bisect

class Buy_Order(object):

        order_reference_number = 0
        quantity = 0
        trade_price = 0.0

        def __init__(self, order_reference_number, quantity, trade_price):
                self.order_reference_number = int(order_reference_number)
                self.quantity = int(quantity)
                self.trade_price = float(trade_price)

        def __hash__(self):
                return hash(self.order_reference_number)

        def __lt__(self, other):
                if self.trade_price != other.trade_price:
                        return self.trade_price < other.trade_price
                return self.order_reference_number < other.order_reference_number
```

```
        def __gt__(self, other):
                if self.trade_price != other.trade_price:
                        return self.trade_price > other.trade_price
                return self.order_reference_number < other.order_reference_number

        def __str__(self):
                return str(self.quantity) + ',' + str(self.trade_price)

class Sell_Order(object):

        order_reference_number = 0
        quantity = 0
        trade_price = 0.0

        def __init__(self, order_reference_number, quantity, trade_price):
                self.order_reference_number = int(order_reference_number)
                self.quantity = int(quantity)
                self.trade_price = float(trade_price)

        def __hash__(self):
                return hash(self.order_reference_number)

        def __lt__(self, other):
                if self.trade_price != other.trade_price:
                        return self.trade_price < other.trade_price
                return self.order_reference_number < other.order_reference_number

        def __gt__(self, other):
                if self.trade_price != other.trade_price:
                        return self.trade_price > other.trade_price
                return self.order_reference_number < other.order_reference_number

        def __str__(self):
                return str(self.quantity) + ',' + str(self.trade_price)
```

```
def BubbleSort(unsorted):
        n = len(unsorted)
        while True:
                new_n = 0
                for i in range(1, n-1):
                        if unsorted[i-1] > unsorted[i]:
                                tmp = unsorted[i-1]
                                unsorted[i-1] = unsorted[i]
                                unsorted[i] = tmp
                                new_n = i
                n = new_n
                if n == 0:
                        break

def Build(datafile, outfile):

        buy_side = list()
        sell_side = list()
        active_orders = dict()
        time = 0
        old_buy = ""
        old_sell = ""

        datafile.seek(1)
        for row in datafile:

                sre_trades = 0 # 1 if SRE trades, 0 otherwise
                sre_buys = 0 # 1 if SRE buys
                sre_sells = 0

                srow = row.split(',')

                time = int(srow[1])
```

```
        if srow[0] == 'A':
                order_reference_number = int(srow[2])
                buy_sell_indicator = srow[3]
                quantity = int(srow[4])
                trade_price = float(srow[5])
                if buy_sell_indicator == 'B':
                        new_order = Buy_Order(order_reference_number,
quantity, trade_price)
                        bisect.insort_right(buy_side, new_order)
                        active_orders[order_reference_number] = new_order
                if buy_sell_indicator == 'S':
                        new_order = Sell_Order(order_reference_number,
quantity, trade_price)
                        bisect.insort_right(sell_side, new_order)
                        active_orders[order_reference_number] = new_order

        if srow[0] == 'EC' or srow[0] == 'XD':
                order_reference_number = int(srow[2])
                order = active_orders[order_reference_number]
                order_qty = order.quantity
                reducing_qty = int(srow[3])
                if order_qty <= reducing_qty:
                        if type(order) == Buy_Order:
                                buy_side.remove(order)
                        else:
                                sell_side.remove(order)
                        del active_orders[order_reference_number]
                else:
                        order.quantity -= reducing_qty

        if srow[0] == 'EC':
                if re.sub('\W', '', srow[6]) == 'SRE':    # SRE buys
```

```
                    sre_trades = 1                  # 1 if SRE trades,
0 otherwise
                    sre_buys = 1                    # 1 if SRE buys, -1
if they sell
                if re.sub('\W', '', srow[7]) == 'SRE':    # SRE sells
                    sre_trades = 1                  # 1 if SRE trades,
0 otherwise
                    sre_sells = 1                   # 1 if SRE buys, -1
if they sell

            buy_side.sort(reverse=True)
            sell_side.sort()

            if len(buy_side) > 0 and len(sell_side) > 0:

                #take the quantity at best price
                buy_price = buy_side[0].trade_price
                buy_qty = 0
                for buy_order in buy_side:

                    if buy_order.trade_price != buy_price:
                        break
                    buy_qty += buy_order.quantity

                sell_price = sell_side[0].trade_price
                sell_qty = 0
                for sell_order in sell_side:

                    if sell_order.trade_price != sell_price:
                        break
                    sell_qty += sell_order.quantity

                #take the quantity at the three best prices
                buy_price = buy_side[0].trade_price
```

```
                buy_qty_three = 0
                p_iter = 3
                for buy_order in buy_side:

                        if buy_order.trade_price != buy_price:
                                buy_price = buy_order.trade_price
                                p_iter -= 1
                                if not p_iter:
                                        break
                        buy_qty_three += buy_order.quantity

                sell_price = sell_side[0].trade_price
                sell_qty_three = 0
                p_iter = 3
                for sell_order in sell_side:

                        if sell_order.trade_price != sell_price:
                                sell_price = sell_order.trade_price
                                p_iter -= 1
                                if not p_iter:
                                        break
                        sell_qty_three += sell_order.quantity

                #take the total quantity
                buy_total = 0
                sell_total = 0
                for buy_order in buy_side:
                        buy_total += buy_order.quantity
                for sell_order in sell_side:
                        sell_total += sell_order.quantity


                outfile.write("%f,%d,%f,%d,%d,%d,%d,%d,%d,%d,%d\n"      %
(buy_price, buy_qty, sell_price, sell_qty,\
```

```
                                    buy_qty_three,sell_qty_three,  buy_total,  sell_total,
sre_trades, sre_buys, sre_sells))


def CleanOutput(infile, outfile):

        infile.seek(10000

                )
        prev_row = ""
        i = 0
        for row in infile:
                i += 1
                new_row = row
                if len(new_row) < 70:
                        print "Error in row %d" % i
                        continue
                if new_row == prev_row:
                        continue
                outfile.write(row)
                prev_row = new_row



#main
datafile = None
#path = 'data/reduced3'
path = 'data/buggbugg'
path_output = 'data/ten/'
```

```
try:
        outfile = open('%stop_exec2.log' % path_output,'w')
        for infile in glob.glob(os.path.join(path, '*.log')):
                print infile[14:-4]
                datafile = open(infile,'r')
                tempfile = open('data/tmp.log','w')
                Build(datafile, outfile)
                tempfile.close()
                tempfile = open('data/tmp.log','r')
                CleanOutput(tempfile, outfile)




except IOError as (e, s):
        print  "I/O Error({0}): {1}".format(e, s)
        sys.exit(1)

finally:
        if datafile:
                datafile.close()
```

For calculating VaR Historical Simulation

```
%
% Data of the kind decribed below is imported by
% running data.m which is generated from
% NASDAQ OMX ITCH-files by a python parser.
%
%
```

```
%+------+----------+-------+-------+-------+-----
%|      |          |Stock 1|Stock 2|Stock 3|Sto...
%+------+----------+-------+-------+-------+----
%|      |          |  Id   |  Id   |  Id   | ...
%+------+----------+-------+-------+-------+----
%|Date 1|Interval 1|Prices |.....
%+------+----------+-------+
%|Date 1|Interval 2|   .     .
%+------+----------+   .       .
%|Date 1|    .         .         .
%+------+    .         .           .
%|  .   |    .         .
%|  .   |    .
%|  .   |    .
%+------+----------+
%|Date 2|Interval 1|
%+------+----------+
%|  .   |    .     |
%|  .   |    .     |
```

```
clear, clc, format long

run data_ten_sec

periods = 400;

alpha = 0.05;

disp('Periods 10 sec')
disp('200 intevals of simulation')

disp('alpha')
disp(alpha)
```

```
prices = cell2mat(prices_data(3:end, 3:end));

inventory = cell2mat(weights_data(3:end, 3:end));

returns = price2ret(prices, [], 'Periodic');
returns = log(1 + returns);

%figure(1)
%hist(returns)



VaR = zeros(size(returns, 1)-periods, 1);

for i = 1+periods:size(returns, 1)
   historical_returns = zeros(periods, 1);
   for j = 1:periods
      for k = 1:size(returns, 2)
         historical_returns(j) = historical_returns(j) + returns(i-j,k) * inventory(i,k) *
prices(i,k);
      end
      historical_returns = sort(historical_returns, 'ascend');
      VaR(i-periods) = historical_returns(floor(periods*alpha));
   end
end




x = (0:1:length(VaR)-1)';
figure(1)
%scatter(x, VaR, 15, 'filled', 'blue'), grid, hold on
plot(x, VaR, 'b'), grid, hold on

realized_return = cell2mat(return_data(3:end, 3:end));
```

```
interval_return = zeros(length(VaR),1);


for i = 1:length(VaR)
   for j = 1:size(returns, 2)
      interval_return(i) = interval_return(i) + realized_return(i+periods, j);
   end
end

figure(1)
%scatter(x,interval_return, 15, 'filled', 'red')
plot(x,interval_return, 'r')

%Kupiec

pval = 1-alpha;
I = (interval_return < VaR);
N = sum(I);
T = length(I);
a = N/T;


disp('alpha hat')
disp(a)


%LR = 2*log(((1-a)/(1-alpha))^(T-N)*(a/alpha)^N);
%LR = 2*log((((1-a)/(1-alpha))^(T-N))*(a/alpha)^(N));
LR = -2*(N*log(pval)+(T-N)*log(1-pval)-N*log(N/T)-(T-N)*log(1-N/T));

Kupiec = 1-chi2cdf(LR,1);

disp('Kupiec')
disp(Kupiec)
```

```
n00=0; n01=0; n10=0; n11=0;

for i = 1:length(I)-1
   if I(i) == 0 && I(i+1) == 0
      n00 = n00 + 1;
   elseif I(i) == 0 && I(i+1) == 1
      n01 = n01 + 1;
   elseif I(i) == 1 && I(i+1) == 0
      n10 = n10 + 1;
   elseif I(i) == 1 && I(i+1) == 1
      n11 = n11 + 1;
   end
end

p0 = n01/(n00+n01);
p1 = n11/(n10+n11);
p  = (n01+n11)/(n00+n01+n10+n11);
LR_C = -2*(n00+n10)*log(1-p)-2*(n01+n11)*log(p)+2*n00*log(1-p0)+2*n01*log(p0)+2*n10*log(1-p1)+2*n11*log(p1);

CIT_L = 1-chi2cdf(LR_C,1);

disp('Christofersen')
disp(CIT_L)
```

For calculating VaR Filtered Historical Simulation:

```
clear, clc, format long
```

```
run data_ten_sec

periods = 400;

alpha = [0.1, 0.05, 0.01];

prices = cell2mat(prices_data(3:end, 3:end));

inventory = cell2mat(weights_data(3:end, 3:end));

VaR = zeros(size(prices-1, 1)-periods, 3);
I = zeros(size(prices-1, 1)-periods, 3);
Plot = zeros(size(prices-1, 1)-periods);

for t = periods+1:size(prices,1)-1
   int_prices = prices((t-periods):(t-1),:);
   int_inventory = inventory(t,:)';
   returns = price2ret(int_prices, [], 'Periodic')*int_inventory;
   T = size(returns, 1);
   spec = garchset('Distribution', 'T', 'P', 1, 'Q', 1, 'R', 1, 'VarianceModel', 'EGARCH',
'Display', 'off');
   returns = returns(:,1);
   if sum(returns) == 0
      continue
   end
   [spec, errors, LLF, residuals, sigmas] = garchfit(spec, returns);
   %garchdisp(spec, errors)
   standardizedResiduals = residuals ./ sigmas;
   s = RandStream.getGlobalStream(); reset(s)
   nTrails = 20000;
   horizon = 6;
   bootstrappedResiduals = standardizedResiduals(unidrnd(T, horizon, nTrails));
   preResidual = residuals(end);
   preSigma = sigmas(end);
```

```
    preReturn = returns(end);
    [~, ~, portfolioReturns] = garchsim(spec, horizon, nTrails, bootstrappedResiduals, [],
[], preResidual, preSigma, preReturn);
    cumulativeReturns = sum(portfolioReturns);
    int_VaR = 100 * quantile(cumulativeReturns, [0.10 0.05 0.01]');
    VaR(t-periods, 1) = int_VaR(1);
    VaR(t-periods, 2) = int_VaR(2);
    VaR(t-periods, 3) = int_VaR(3);
    int_prices = prices((t-1):t,:);
    returns = price2ret(int_prices, [], 'Periodic')*(int_inventory.*int_prices(2,:)');
    Plot(t-periods) = returns;
    I(t-periods, 1) = (int_VaR(1)>returns);
    I(t-periods, 2) = (int_VaR(3)>returns);
    I(t-periods, 3) = (int_VaR(2)>returns);
    fprintf( 'Maximum Simulated Loss: %8.4f%s\n' ,
100*min(cumulativeReturns),'SEK')
    fprintf(' Maximum Simulated Gain: %8.4f%s\n' , 100*max(cumulativeReturns),'SEK')
    fprintf('      Realized Return: %8.4f%s\n' , returns,'SEK')
    fprintf('    Simulated 90%% VaR: %8.4f%s\n' , int_VaR(1),'SEK')
    fprintf('    Simulated 95%% VaR: %8.4f%s\n' , int_VaR(2),'SEK')
    fprintf('    Simulated 99%% VaR: %8.4f%s\n', int_VaR(3),'SEK')
    fprintf('         Hit 99%% VaR: %8.4f\n', int_VaR(1)>returns)
    fprintf('         Hit 95%% VaR: %8.4f\n', int_VaR(2)>returns)
    fprintf('         Hit 90%% VaR: %8.4f\n\n', int_VaR(3)>returns)
end

for inter = 1:3
    idx = find(VaR(:,iter)<-1e+4);
    tmp_var = VaR(:,iter);
    tmp_plot = Plot;
    tmp_plot(idx) = [];
    tmp_var(idx) = [];
    x=(1:1:length(tmp_var));
    figure(iter)
```

```
    plot(x,tmp_var, 'b'), hold on
    x=(1:1:length(tmp_plot));
    plot(x,tmp_plot, 'r'), hold on
end



%Kupiec
Kupiec = zeros(3);
Christoffersen = zeros(3);

for iter = 1:3
    pval = 1-alpha(iter);
    Ii = I(:,iter);
    N = sum(Ii);
    T = length(Ii);
    a = N/T;



    %LR = 2*log(((1-a)/(1-alpha))^(T-N)*(a/alpha)^N);
    %LR = 2*log((((1-a)/(1-alpha))^(T-N))*(a/alpha)^(N));
    LR = -2*(N*log(pval)+(T-N)*log(1-pval)-N*log(N/T)-(T-N)*log(1-N/T));

    Kupiec(iter) = 1-chi2cdf(LR,1);




    n00=0; n01=0; n10=0; n11=0;

    for i = 1:length(Ii)-1
        if Ii(i) == 0 && Ii(i+1) == 0
            n00 = n00 + 1;
        elseif Ii(i) == 0 && Ii(i+1) == 1
```

```
        n01 = n01 + 1;
    elseif Ii(i) == 1 && Ii(i+1) == 0
        n10 = n10 + 1;
    elseif Ii(i) == 1 && Ii(i+1) == 1
        n11 = n11 + 1;
    end
  end

  p0 = n01/(n00+n01);
  p1 = n11/(n10+n11);
  p  = (n01+n11)/(n00+n01+n10+n11);
  LR_C = -2*(n00+n10)*log(1-p)-2*(n01+n11)*log(p)+2*n00*log(1-
p0)+2*n01*log(p0)+2*n10*log(1-p1)+2*n11*log(p1);

  Christoffersen(iter) = 1-chi2cdf(LR_C,1);
end

fprintf('    Kupiec 99%% VaR: %8.4f\n\n',  Kupiec(1))
fprintf('    Kupiec 95%% VaR: %8.4f\n\n',  Kupiec(2))
fprintf('    Kupiec 90%% VaR: %8.4f\n\n',  Kupiec(3))

fprintf('    Christoffersen 99%% VaR: %8.4f\n\n',  Christoffersen(1))
fprintf('    Christoffersen 95%% VaR: %8.4f\n\n',  Christoffersen(2))
fprintf('    Christoffersen 90%% VaR: %8.4f\n\n',  Christoffersen(3))

avg_90 = mean(VaR(:,1));
avg_95 = mean(VaR(:,2));
avg_99 = mean(VaR(:,3));
```

Code for displaying return characteristics.

```
clear, clc, format short
```

```
run data_ten_sec
prices = cell2mat(prices_data(3:end, 3:end));

inventory = cell2mat(weights_data(3:end, 3:end));

returns_raw = cell2mat(return_data(3:end, 3:end));

%int_prices = prices();
%int_inventory = inventory()';
w = (inventory.*prices);
wp = w(2:end,:);
returns_raw = price2ret(prices, [], 'Periodic').*wp;




for i = 1:length(returns_raw)
   returns(i) = sum(returns_raw(i,:));
end

%w = w';

for i = 2:length(w)
   rel(i) = sum((w(i-1,1:44) - w(i,1:44))/w(i-1,1:44));
   %(w(i,2) - w(i-1,1))/w(i-1,1)
   %rel(i) = sum((w(i,1) - w(i-1,1))/w(i-1,1));
end



x = (1:1:length(returns));

figure(1)
subplot(3,2,1)
plot(x,returns, 'b')
```

```
x = (1:1:length(rel));

figure(1)
subplot(3,2,4)
plot(x,rel, 'b')


min_ = min(returns);
max_ = max(returns);
mean_ = mean(returns);
median_ = median(returns);
mode_ = mode(returns);
std_ = std(returns);
var_ = var(returns);
skewness_ = skewness(returns);
kurtosis_ = kurtosis(returns);
total_ = sum(returns);

disp('min')
disp(min_)
disp('max')
disp(max_)
disp('mean')
disp(mean_)
disp('median')
disp(median_)
disp('mode')
disp(mode_)
disp('std')
disp(std_)
disp('var')
disp(var_)
```

```
disp('skewness')
disp(skewness_)
disp('kurtosis')
disp(kurtosis_)
disp('total')
disp(total_)


idx = find(returns>0);
nb_positive_ret = length(idx);
idx = find(returns<0);
nb_negative_ret = length(idx);

disp('Number of positive returns')
disp(nb_positive_ret)

disp('Number of negative returns')
disp(nb_negative_ret)


figure(1)
subplot(3,2,2)
autocorr(returns)

figure(1)
subplot(3,2,3)
hist(returns)

figure(1)
subplot(3,2,5)
autocorr(rel)

figure(1)
subplot(3,2,6)
```

```
hist(rel)


min = min(rel);
max = max(rel);
mean = mean(rel);
median = median(rel);
mode = mode(rel);
std = std(rel);
var = var(rel);
skewness = skewness(rel);
kurtosis = kurtosis(rel);
total = sum(rel);

disp('min')
disp(min)
disp('max')
disp(max)
disp('mean')
disp(mean)
disp('median')
disp(median)
disp('mode')
disp(mode)
disp('std')
disp(std)
disp('var')
disp(var)
disp('skewness')
disp(skewness)
disp('kurtosis')
disp(kurtosis)
disp('total')
disp(total)
```

```
idx = find(rel>0);
nb_positive_ret = length(idx);
idx = find(rel<0);
nb_negative_ret = length(idx);

disp('Number of possitive rel')
disp(nb_positive_ret)

disp('Number of negative rel')
disp(nb_negative_ret)
```

Code for calculating net market exposure

```
clear, clc, format short


run data_ten_sec

prices = cell2mat(prices_data(3:end, 3:end));

inventory = cell2mat(weights_data(3:end, 3:end));

returns_raw = cell2mat(return_data(3:end, 3:end));

w = (inventory.*prices);

for i = 1:size(w, 1)
    exposure(i) = sum(w(i,:)/1000000);
end
```

```
x = (1:1:length(exposure));

figure(1)
plot(x,exposure, 'b')
xlabel('Periods [10 s]')
ylabel('Net exposure [MSEK]')
```

File For regression

```
clear all

set more off



*cd "~/Desktop/uppsats egna skrifter/stata/"

cd "\\Viscount\User09$\21918\Desktop"

*cd "C:\Documents and Settings\PC\Skrivbord"



insheet using "data_patrik_nyklar.csv", delim(;) name



drop v32-v34


gen mktshare = volumesre/volumemrkt
gen highlow = (intervalhigh - intervallow) / intervalopen
gen mrktret = (intervalclose/intervalopen) - 1
gen relatvolume = volumemrkt/volumeavg3m
```

gen relattick = ticksize/volumesre

*gen relattick = ticksize/intervalclose

*gen relatbidask = bidask/mrktclose

*gen relatbidask = bidask/volumesre

gen relatbeta = beta/volumesre

gen mrktretstd = mrktret/std

gen riskprem = mrktret - montheuribor

gen capm = montheuribor + beta * (mrktret-montheuribor)

gen andracapm = beta*(mrktret-montheuribor)

gen matched = volumemtchd / volumesre

*gen netty = speed / netqty

gen nety = netqty / beta

gen trade = nbtrades/volumesre

gen noll = over_500/trade

gen ett = over_1m/volumesre

gen tva = over5m/nbtrades

gen vol = vstoxx/volumemrkt

```
gen nettty = netqty/volumesre

sort volumemrkt

gen name1 = _n

drop if name1 < 2385

gen time = 1 if date == 20120210

replace time = 100 if date == 20120216

replace time = 200 if date == 20120217

replace time = 300 if date == 20120321

replace time = 400 if date == 20120328

replace time = 500 if date == 20120508

replace time = 600 if date == 20120509

replace time = 700 if date == 20120510

replace time = 800 if date == 20120511
```

```
replace time = 900 if date == 20120514


gen time1 = time + intervalnr


*tostring date, replace

*gen ndate = date(date,"YMD")

*format ndate %td


xtset id time1
xtreg retpercent matched highlow ticksize montheuribor vstoxx changevol trade, fe r


sum, detail


tabstat retpercent highlow, statistics(mean sd kur skew) columns(stat)


sktest retpercent
```

qnorm retpercent

swilk retpercent