Bachelor of Science Project

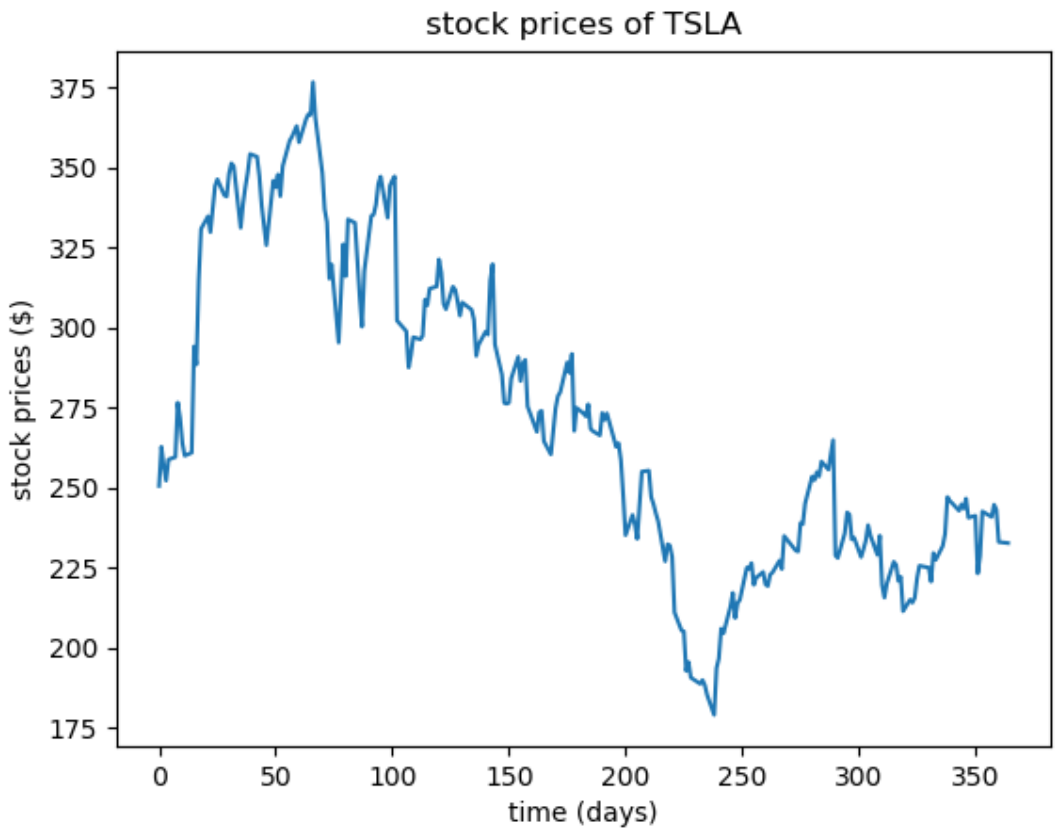
“Analysis of Stock Market Variability”

Research and write up by: Modestas Motiejūnas

Supervised by: Prof. Ian McHardy

Abstract

The prediction of stock market has come a long way from the basic fundamental analysis techniques to the complex machine learning algorithms mostly used nowadays to forecast stock market. This project uses a different approach to try and predict stock prices. Remarkable statistical methods used in Astronomy for analyzing X-ray spectrum, are used to analyze the variability of stock market. To be exact, Discrete Correlation Function (DCF) and Power Spectrum Density (PSD). Investigation of Power Spectrum led to the conclusion that stock prices move according to Brownian motion, so the path followed by the stock can be described by random walk, from which follows that it's impossible to forecast stock prices using this method. Analyzing DCF showed that it's sometimes possible to predict the trend that stock prices will follow in the future and in the case of supplier companies, if most of the profit is made out of the supplied company, correlation between these companies will be stronger and vice versa. Implying, that DCF technique can be potentially used by investors for higher returns.



Contents

1 Introduction

What if you knew the behavior of stock market in the future? Imagine all the money that you could make, if you knew this kind of information, you would have the potential to become the richest man on earth and as everything revolves around money these days, you could do anything and I mean anything: fund all the most important researches: find the cure for cancer, aging, put people on mars, make humans intergalactic species, the future is in your hands ! But there is a little problem, stock market depends on too many factors, which makes it almost impossible to forecast. A lot of banks are investing huge amounts of money into sophisticating machine learning algorithms to help find patterns in stock market. In this research paper, a different kind of approach is used for trying to predict stock market. It includes the use of statistical methods, that are known for their usefulness in astronomy.

One of these methods is called Discrete Cross Correlation function (DCF). In astronomy it's used to find correlation between X-ray and infrared spectrum in some time lag [1]. What makes it useful in analyzing stocks is that the values calculated, are normalized, because of the standard deviations in the denominator. Usually the stock prices of two different companies have drastically different price points, so this feature works in advantage compared to other cross correlation functions. Another benefit of DCF is that it makes it possible to analyze unevenly sampled variability data [2]

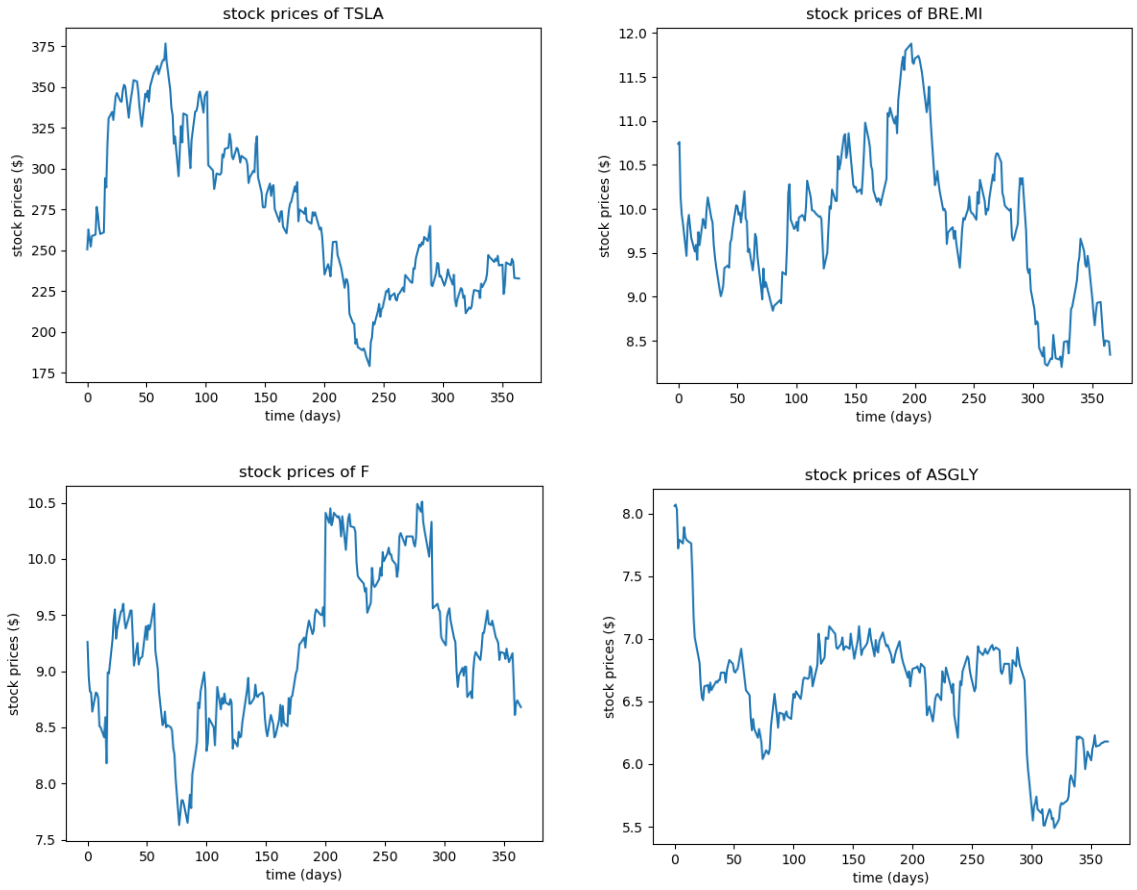
Second method used is called Power Spectrum Density (PSD). Widely used in astronomy to check if a signal received is some random process (noise) or if there is periodicity which is caused by an underlying force. This is going to be used to get a power spectrum of a company's stock price variability and compare it to some known power spectrums.

In this report I analyze Tesla (TSLA), It's competitors: Ford (F), Honda Motor Company (HMC) and suppliers: Asahi Glass (ASGLY), Brembo (BRE.MI), Modine Manufacturing Company (MOD). The motivation behind my interest in analyzing Tesla is belief that in the future all vehicles will be powered by sustainable energy and Tesla is the leading Electric vehicles (EV) manufacturer in the world. Also, it is one of the few car manufacturers in USA that survived the 2008 crisis. For these specific reasons, I would like to investigate if there are any significant correlations between Tesla, it's suppliers and competitors and if there exists an underlying force behind their fluctuations.

2 Method

2.1 Handling stock prices

Before the application of the previously mentioned methods, the data must be prepared for the usage. This can be done by downloading the stock prices of the wanted companies, converting the calendar dates into days for convenience and interpolating missing days and prices to make the time series continuous. To begin with, finance yahoo was used for getting the raw data for a year, taking 2018/10/08 as a starting point for the chosen companies. Secondly, python library *Datetime* was used to convert from regular calendar dates to Gregorian calendar dates, that provide time in seconds from the start of the epoch. This made it possible to get time in days, relative to the very first date. Finally, the only thing left was linear interpolation of the missing days, as the stock market closes on weekends and holidays, in order to make it continuous. The code used for this can be seen in Appendix.



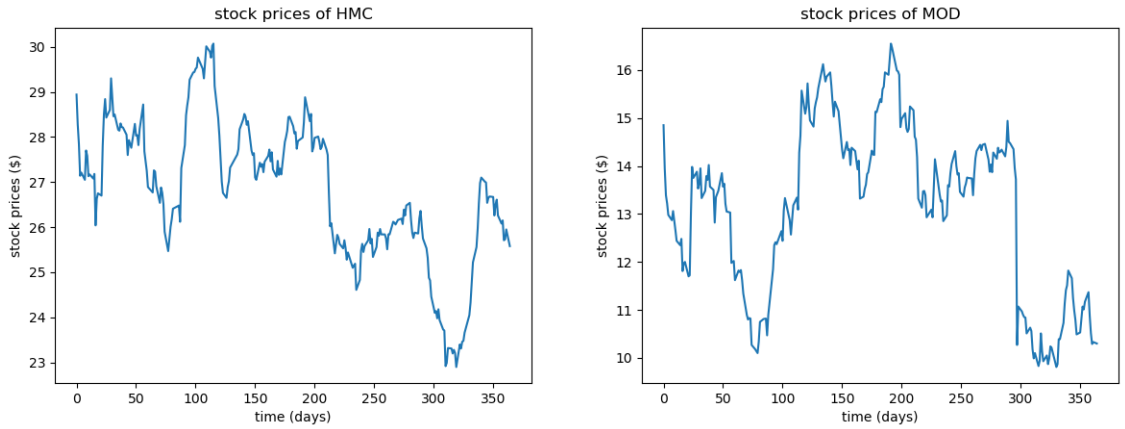


Figure 1 *Closing prices of the companies over one year*

2.2 Discrete Cross Correlation Function (DCF)

Now that the time series are all sorted, the DCF can be used to get the correlation between two companies in some time lag. As mentioned before, DCF has the advantage over other Cross Correlation Functions, because it doesn't require interpolation, it eliminates spurious effects due to correlated errors and finally DCF gives a quantitively meaningful error estimate [2] This information would prove to be crucial for any investor seeking to make profit.

$$DCF(\Delta T) = \frac{\sum(x_n - x_{mean})(y_n - y_{mean})}{\sigma_x \sigma_y} \qquad \textit{(Equation 2.1)}$$

In the *Equation 2.1* x represents prices of one company and y of the other one. To get the variance σ_x I used *Equation 2.2* to get the variance manually:

$$\sigma_x = \sqrt{\frac{\sum(x_n - x_{mean})^2}{N-1}} \qquad \textit{(Equation 2.2)}$$

But then I found out that *Numpy* library in Python has module called *std* that gets the same standard deviation in just one line, so for the sake of convenience I used this module.

ΔT in the Equation 2.2.1 is the time lag between two prices. $\Delta T = t_x - t_y$. The way my algorithm works is, it starts from time lag of 1 day between two given companies and calculates all the values of DCF possible, then moves to 2 days' time lag and so on. The algorithm executes the same commands, till ΔT is not bigger than number of days in the excel spreadsheet. At this point, it's possible to plot a preliminary graph of DCF values against time lags (*Figure 2*).

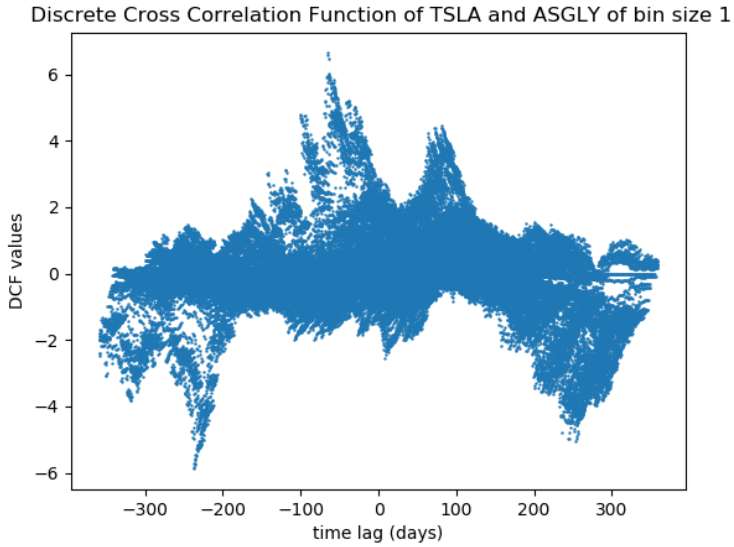


Figure 2 *DCF of TSLA and ASGLY*

As the amount of data points in *Figure 2* is immense, it makes it extremely hard to interpret the graph and draw any plausible conclusions from it. That's where binning up comes in handy. Data points at each time lag are put in separate arrays and an average value of them is calculated with an error which comes from the standard deviation. This reduces the number of data points significantly and the graph becomes ready for interpretation (*Figure 3)*

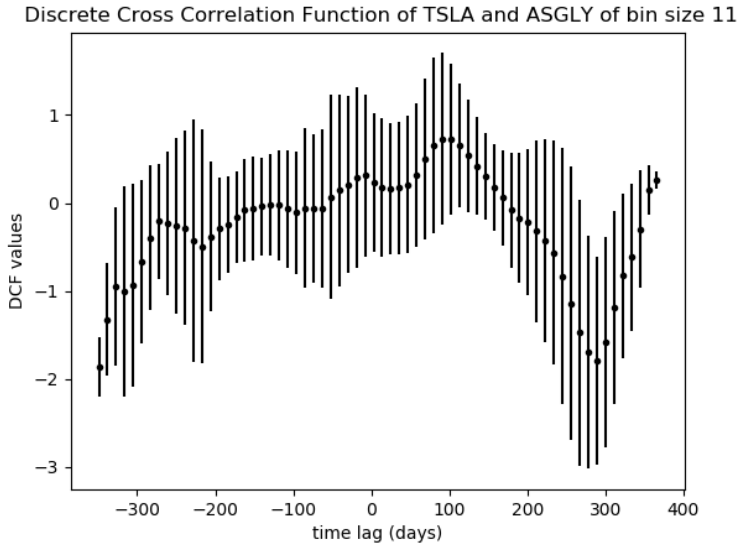


Figure 3 *Binned up DCF of TSLA and ASGLY*

2.2.1 Probability density function (PDF)

It's easy to get an uncertainty in cross correlation value, we must calculate the spread of all the values in separate time lags and it provides us with error bars. But obviously there is an uncertainty not only in DCF value, but also in time lag, so how can this uncertainty be extracted? One of sophisticating methods is called Probability Density Function (PDF), it is

used to specify the probability of random variable falling within a particular range of values. The procedure consists of couple of steps. To begin with, random sample of around 70% of data points must be taken a lot of times of each stock price graph. For this step I used Python's module called *Random* and the condition $if\ random.random(x) < 0.7$ it adds a stock price and the date to the list, where $random.random(x)$ produces a number from 0 to 1. After the random sampling, the DCF values must be calculated and, in this case, it's not as simple as before, because the random function will take different days for different companies. To solve this problem, I've used a condition that checks if $day1 + timelag = day2$, if condition is satisfied DCF value is calculated using *Equation 2.1*

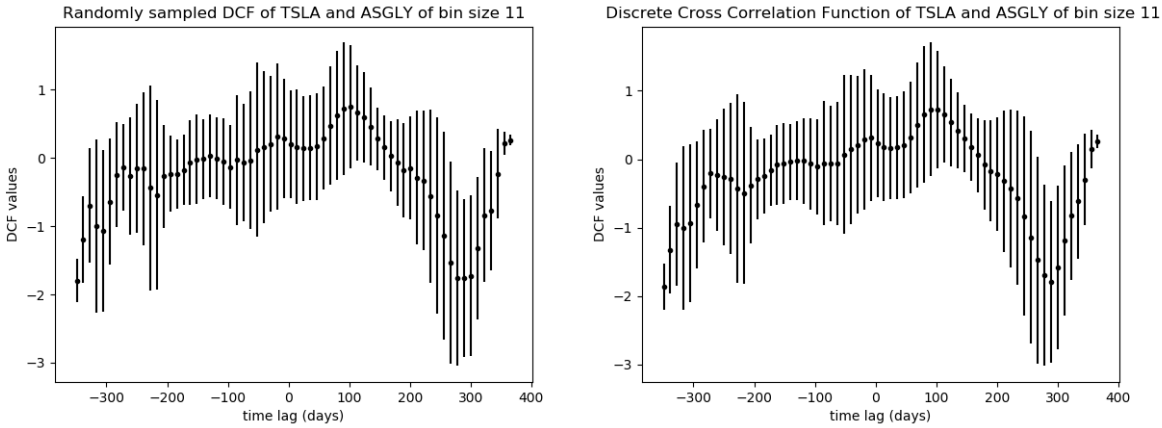


Figure 4 *Comparison of regular DCF (right graph) with DCF after random sampling (left graph)*

As can be seen from *Figure 4*, the DCF graph after random sampling is slightly different from the normal DCF graph. That's because around 30% of days and prices are missing from original graphs and when DCF values are calculated at each time lag, the scatter graph has less values what leads to different average DCF values and uncertainties. Finally, for the last step the peak value of each random sampled DCF had to be taken, the more samples, the better graph of PDF can be produced. Due to time constraints, I chose to get the randomly sampled DCF 100 times. The execution of all these steps led to PDF of two companies (*Figure 5*).

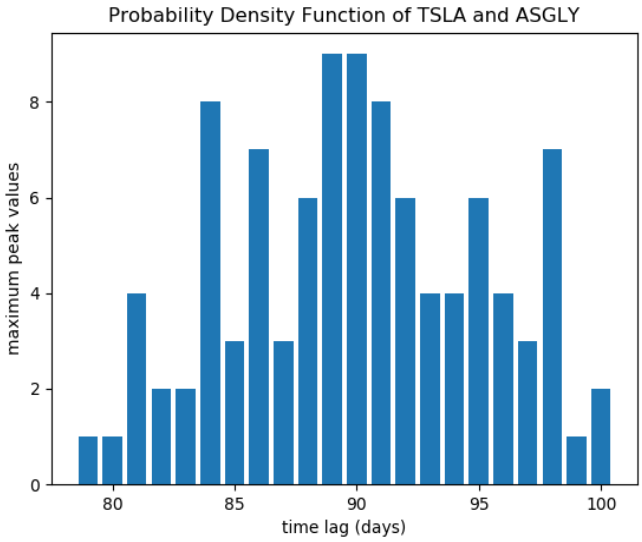


Figure 5 *PDF of TSLA and ASGLY*

Finally, from the best fit of normal distribution it should be possible to extract the value of 1σ that gives a 68% certainty that our peak value will fall within that range. Due to a lot of failed attempts to fit a gaussian in Python, I was forced to use Origin software to determine the uncertainty in time lag. *Figure 6* shows the final result of PDF with the best fit gaussian distribution.

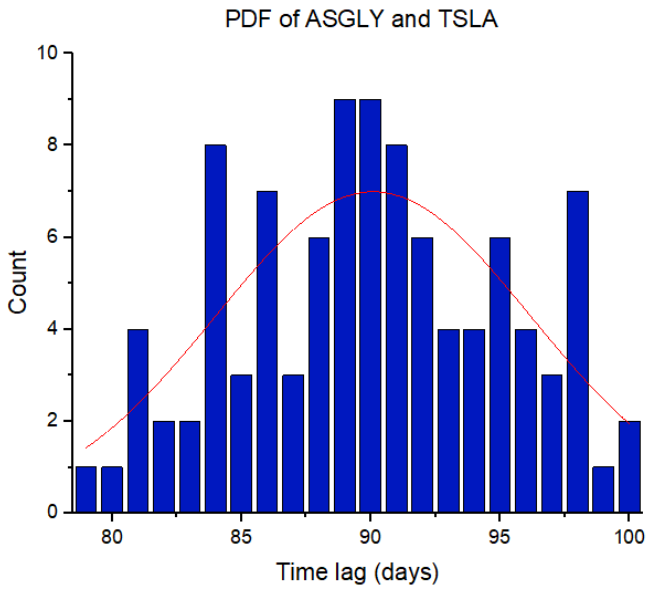


Figure 6 *PDF of ASGLY and TSLA with the best fit Gaussian distribution*

2.3 Auto-Correlation function (ACF)

Auto-Correlation function gives a correlation of time series with a delayed copy of itself. This method provides information on how fast and how drastically the stock prices of a company fluctuate, also, it's a remarkably useful tool for checking the validity of DCF and getting the most optimal DCF graph possible. To check if the DCF that we are getting is correct, we can check the shape of ACF, if the peak is at 1 and it's symmetric to both sides, it means that DCF produced will be valid. Furthermore, the width of ACF gives information on the memory of DCF function, if we were to bin up time domain (let's say take the mean value of 10 days) we would get a graph with less data points and therefore a different width. If that width is not too different from the unbinned (in time's domain) graph's width, it means that it can be binned up to the point where the difference between widths becomes significant. This will allow us to produce a DCF with less data points, but without losing too much essential information.

The equation for calculating ACF values is very similar to *Equation 2.1*, the only difference is instead of having two companies x and y, we only have x or y

$$\mathrm{A}CF(\Delta T) = \frac{\sum(y_n - y_{mean})^2}{\sigma_y^2} \qquad \textit{(Equation 2.3)}$$

Finally, the graph produced is represented by *Figure 7*

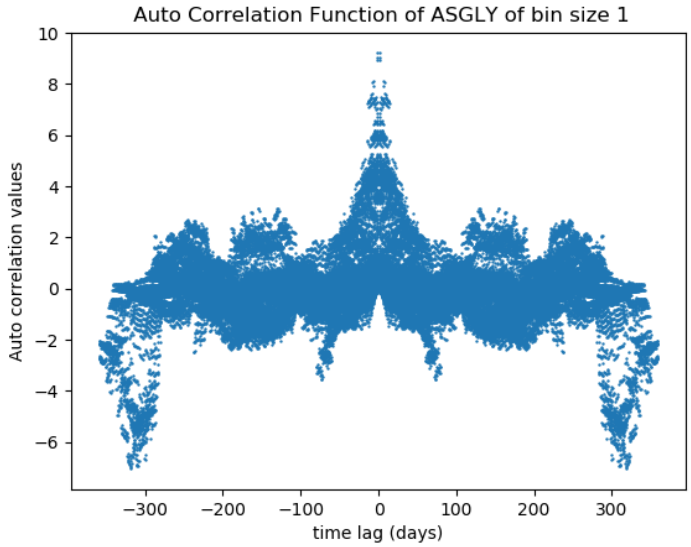


Figure 7 *ACF of ASGLY*

Contrary to DCF, it is possible to interpret the unbinned version of ACF. The graph is symmetric with the peak being at time lag 0, this supports the validity of our DCF graph, though it makes it hard to tell what the exact width is, for this reason we have to do the same procedure as for DCF, taking all the data points at each time lag and getting an average of them. *Figure 8* shows the result of that, where the uncertainties come from standard deviation.

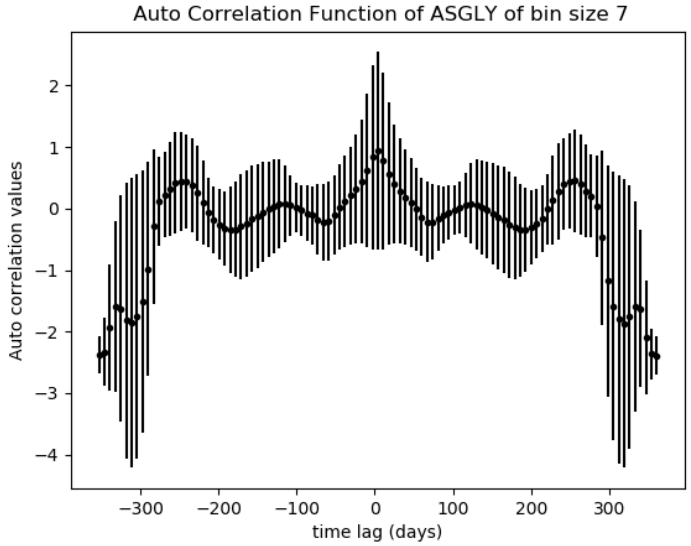


Figure 8 *binned up ACF of ASGLY*

The only step that's left is to bin up values in time lags, till the most optimal ACF graph is produced, containing just the right amount of information for investigation.

2.4 Power spectrum

Another sophisticating technique used for trying to forecast stock market prices is called Power Spectrum Density (PSD). It is widely used in astronomy for processing various signals and it describes how power of a signal is distributed over frequency. From the gradient of power spectrum, it's possible to tell whether it's just a random process/noise or if there is some underlying force producing the signal.

To achieve this, stock prices had to be transformed into real and complex amplitudes of the fundamental waves that are building blocks of the complicated fluctuation of stock prices. This can be done without too much effort with a Fast Fourier Transform (FFT), which can be imported from the *Numpy* library. Furthermore, the time domain must be converted to frequency domain as it is amplitudes of the frequencies that make up the original graph. In order to do that we had to use *Equation 4*

$$f_{max} = \frac{1}{\tau_{min}} \qquad \textit{(Equation 2.4)}$$

Where τ_{min} is the smallest time interval in time series, for our case it's one day or 86400 seconds. This provides us with the highest frequency and from 0 to f_{max} there are as many intervals, as days in the original graph. This leaves us with *Figure 9*

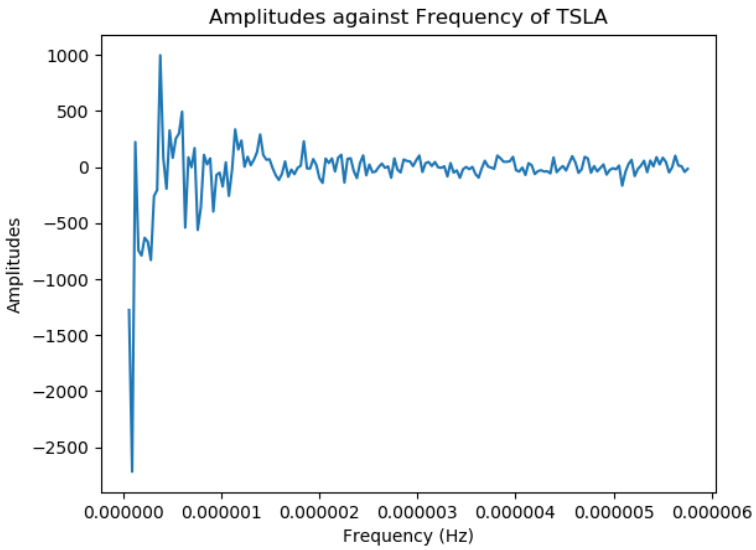


Figure 9 *Fourier transformed stock price graph of TSLA*

The next step in producing a power spectrum is to convert the amplitudes to power by taking an absolute value of amplitudes and squaring it.

$$Power = |Amplitudes|^2 \qquad \textit{(Equation 2.4)}$$

The only thing that's left to get a Power Spectrum is taking a $\log_{10} x$ of both Power and Frequencies (*Figure 10*)

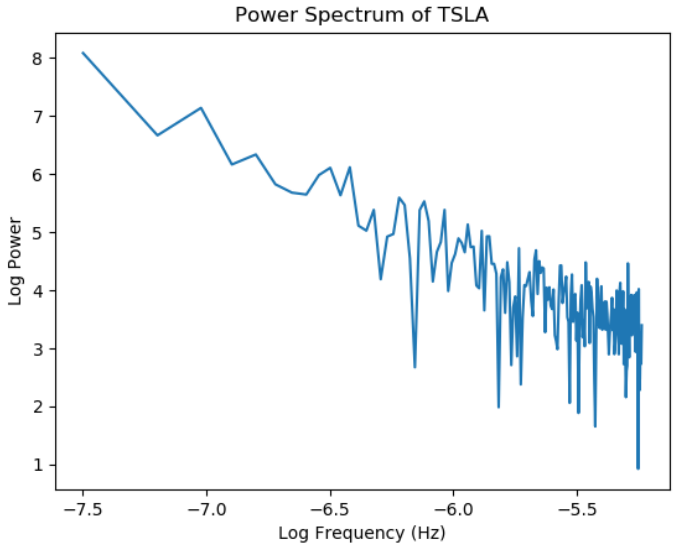


Figure 10 *Power spectrum of TSLA*

Although all the steps to get a power spectrum have been described, to extract useful information from the power spectrum, there still are some procedures left to do. One of them is binning up the power spectrum to equal intervals in frequency domain to get fewer points with uncertainties that come from standard deviation. This makes it easier to get the best linear fit, which is then used to get the gradient of power spectrum. The algorithm for this procedure works like this: it divides the frequency domain by number of bins inputted, adds all the data points in each interval to separate arrays and then gets the average value of each array (uncertainties come from the standard deviation).

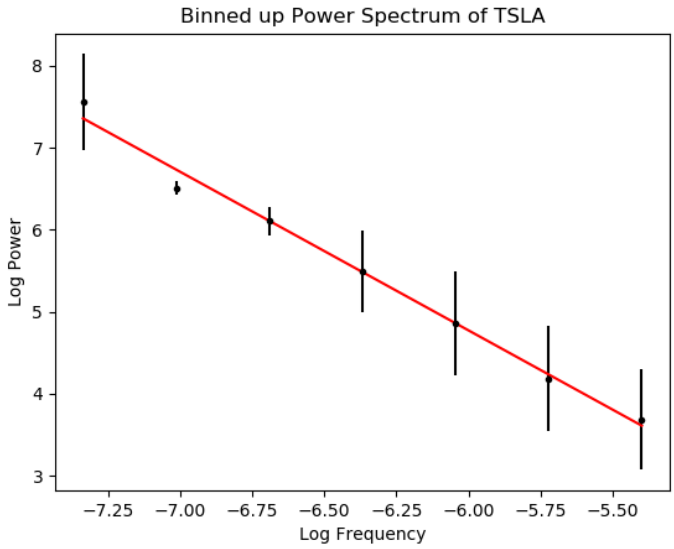


Figure 11 *Binned power spectrum of TSLA*

Another problem that arises is that power estimates have a constant bias, but not to worry it can be easily corrected using I.E.Papadakis and A.Lawrence method, by “subtracting” the bias from the logarithms of power spectrum (adding 0.253 to each value in power spectrum) [3]. This can be considered an unbiased estimate of the logarithm of power density.

2.4.1 Recreating time series

One remarkable method that needs to be mentioned in this project is simulating time series from the power spectrum. Using this method, it's possible to produce variety of time series that have the same gradient in power spectrum. In astronomy, this is used for simulation of X-ray time series of AGN lightcurves in order to judge the data in correct manner, because there are gaps in observed lightcurve, that make it almost impossible to derive characteristics of the present fluctuations from power spectrum[4] Therefore, for our research it can be used to simulate the behavior of stock prices from the power spectrums.

The new algorithm proposed by J. Timmer and M. König randomizes both phase and amplitude of the Fourier transform, while previously used algorithms would randomize only phase and choose a deterministic amplitude. Huge drawback of the previously used algorithms is that simulated time series would provide a trend caused by dominating lowest frequency.

The recipe for simulating time series is as follows: when power spectrum of a company is produced, the power is expressed as the logarithm of base 10, so it needs to be transformed back by taking a logarithmic power amplitude as a power of 10. From there it's possible to get an absolute amplitude by taking a square root of every power value. Finally, with the use of inverse Fourier transform closing prices of each day can be calculated, providing us with simulated behavior of stock market.

3 Results and Analysis

3.1 Cross Correlation

Prior to discussing Discrete Cross Correlation Function of the companies we have to analyze the results of Auto Correlation Function as different bin sizes are taken. The main feature that we are looking for is the change in width around 0 day time lag, as the bin size increases. If the width changes by a significant amount, it means that too much information is lost and the bin size is too big.

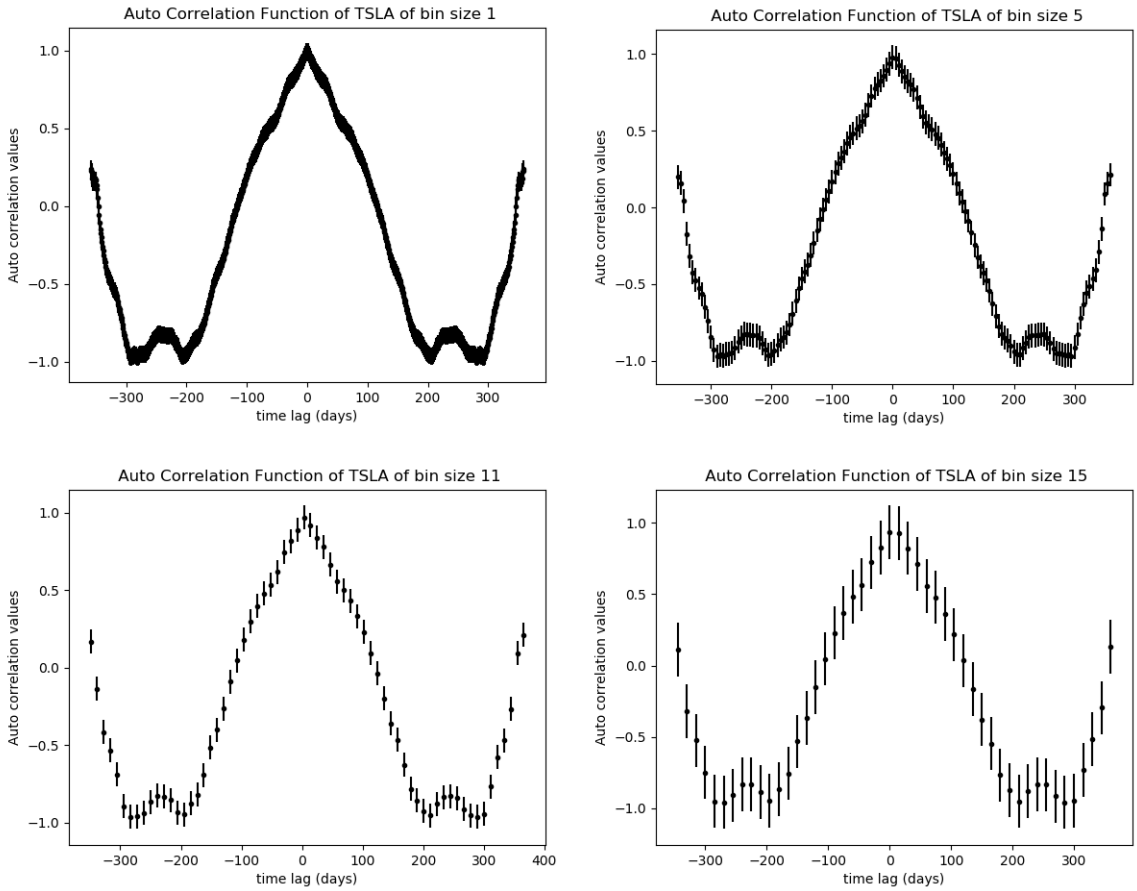


Figure 12 *ACFs of different bin sizes of TSLA*

Without complicated analysis, just by using pure reasoning from what can be seen on *Figure 12*, I have concluded that bin size of 11 days is perfect because it still contains all the important information and also it is the limit where the graph starts losing it's original form

Also, the uncertainties of each ACF were of a huge interest for me, as they were significantly different in each company. For example, if we compare BRE.MI with TSLA (*Figure 13)* we can clearly see this phenomenon.

If we get enormous errors and they come from standard deviations, it means that there is an immense spread of ACF values at all time lags. Furthermore, from *Equation 3* we can extract useful information regarding this question. The only variable that is not constant is y_n and this leads to a conclusion that one of the factors influencing the uncertainties is how quick the prices vary on the original time series.

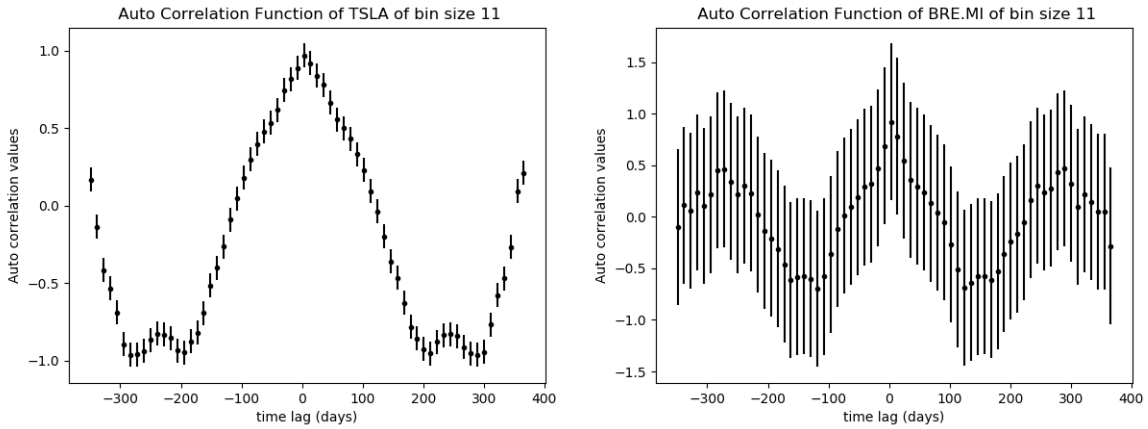


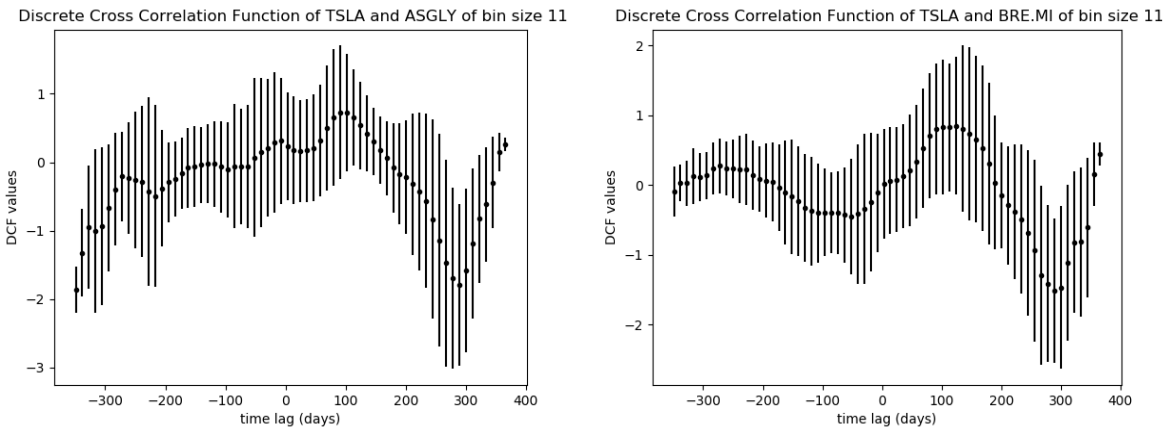
Figure 13 *ACFs of TSLA and BRE.MI*

Another factor that is related to the size of the error bars is the standard deviation of the original stock prices. Bigger standard deviations lead to smaller ACF values, as the variance is used to normalize the values. For these two main reasons we get so drastically different uncertainties in ACF graphs

```
ASGLY standart deviation 0.481412674277781526
BRE.MI standart deviation 0.8266523154449099
F standart deviation 0.66340142137556
HMC standart deviation 1.5605514709512074
MOD standart deviation 1.7624224021189498
TSLA standart deviation 48.11074146405605
```

Figure 14 *standard deviations of original time series*

Even though ASGLY and F have lower standard deviations than BRE.MI (*Figure 14*), their uncertainties are smaller and that is for the reason mentioned earlier: their fluctuations aren't as quick and as far from the mean value as for BRE.MI graph.



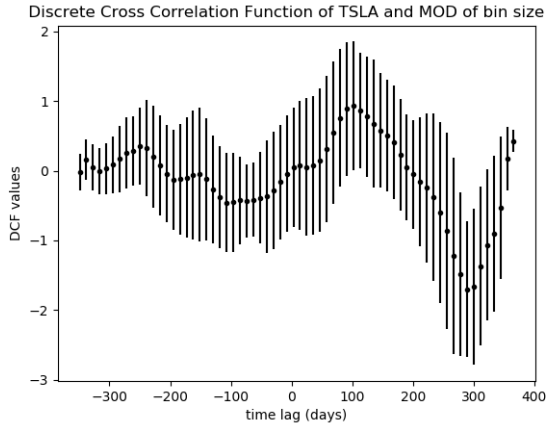
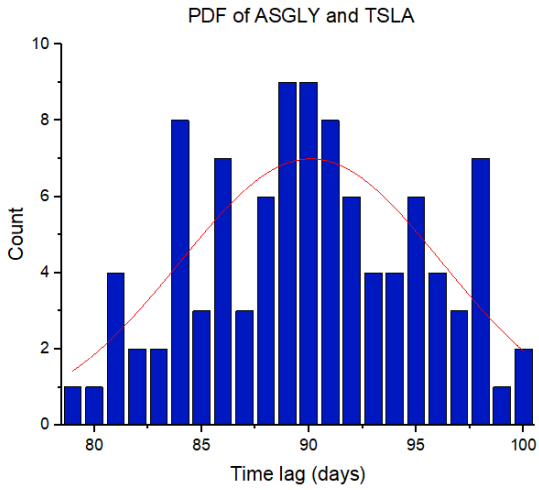


Figure 15 *DCF of TSLA and its suppliers*

Theoretically, it is known that there should be a correlation between company share's variability and its supply chain [5]. But the main question to answer is not whether there exists a correlation, but rather, what is that correlation quantitively. There are a lot of factors to consider: how big the companies are, how many suppliers they have, how much of the profit comes from the supplied company. *Figure 15* illustrates the DCFs of Tesla and its suppliers, it's remarkable to see that all the graphs follow similar patterns. For all the mentioned companies the DCF value increases from day 0 time lag to around day 100, reaches the maximum value and then plummets down to a minimum value at around day 300 time lag. The minimum value won't be considered because its value is below -1, which is unrealistic, and it happens due to the lack of data points at bigger time lags. The situation is a bit different at negative time lags. TSLA/BRE.MI and TSLA/MOD seem to follow a similar trend by slowly decreasing and then increasing, while TSLA/ASGLY just keeps on declining with frequent fluctuations

To get the uncertainty in time lag and check whether peaks correlate with one another (fall within each other's standard deviation) PDF graphs need to be analyzed (*Figure 16*)



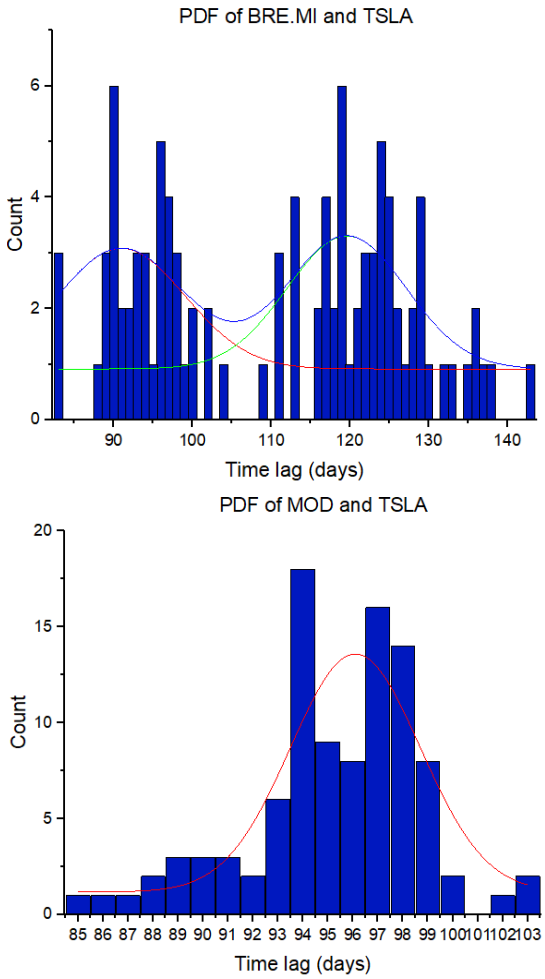


Figure 16 *PDFs of TSLA and its suppliers*

At first glance, the PDF graph for BRE.MI and TSLA might seem wrong, as it appears to have two probability distributions at different time lags. But this is completely valid and if we want to understand this behavior, we must analyze the original DCF graph by zooming in on the time lags (89 to 130 days).

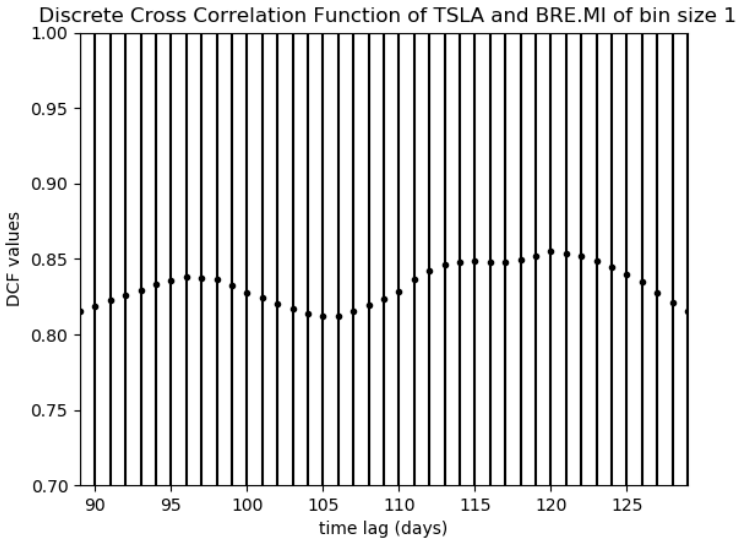


Figure 17 *Zoomed in DCF of TSLA and BRE.MI*

As suspected, this behavior in *Figure 17* arises because of the peak's shape in the original DCF graph. It has two peaks with similar DCF values, but at different time lags. When graphs are randomly sampled, the DCF produced, gets the maximum value at one or the other time lag, that's why the distributions in *Figure 16* form in this particular way. Therefore, for TSLA and BRE.MI we must consider two different time lags as the potentially best correlations.

Company	Peak value	Uncertainty	Time lag	Uncertainty
ASGLY/TSLA	0.73	0.98	90	14.3
BRE.MI/TSLA	0.85	0.98	91, 119	16.1, 15.1
MOD/TSLA	0.93	0.92	102	6.1

Table 1 *Best DCF values and their time lags with uncertainties of suppliers*

Table 1 gives the best DCF values and time lags (both with uncertainties that origin from standard deviation) of TSLA and its suppliers. If we plotted these time lags with their errors in one dimension, all three ranges would overlap, meaning that TSLA's supply chain follow very similar trend and have strong correlations at similar time lags. Furthermore, I wanted to investigate why MOD/TSLA has a maximum DCF value of 0.93 ± 0.92, while ASGLY/TSLA has 0.73 ± 0.98. My theory is that if a company is smaller and most of its profit comes from TSLA, it will depend more on it and therefore there will be a stronger correlation. To test my theory, I had to find a factor that in some way relates to the size of the company and that factor is market cap. It refers to how much a company is worth as determined by stock market. For MOD market cap is 392M($), BRE.MI 3.6B($) and MOD 8.2B($). This data supports my theory, the smaller the supplier the stronger the correlation. Although, this has some flaws, as market cap doesn't measure the equity value of a company, since shares are often over- or under valuated. Finally, the best investment plan, if taken *Table 1* as a reference, would be to wait for TSLA's stock prices to increase, then buy shares of MOD, wait 102 ±6.1 days for them to follow similar trend and sell stock prices when they have raised.

Now for TSLA's competitors F and HMC, we get DCFs shown in *Figure 18*

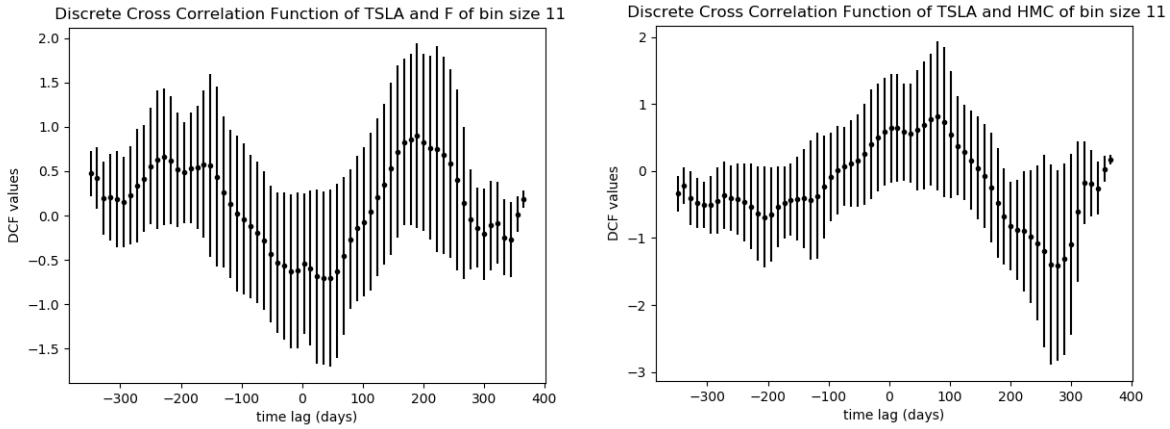


Figure 18 *DCFs of TSLA/F and TSLA/HMC*

The results for both DCF graphs are quite interesting. The TSLA/F graph seems to have a strong anti-correlation of around -0.75 at relatively short time lags up to ±50 days, from there

it starts to increase to eventually reach significantly high correlations at time lag of approximately ±200 days. Contrary to the previously described graph, TSLA/HMC graph has a positive correlation around time lag 0 that rises up to day -20 and 80 to reach the maximum values and then proceeds to plummet down to the minimum value of around -0.8 at day -200 and 200 respectively. The minimum value of around -1.3 is not considered, as it has no interpretation, since the best anti-correlation can be -1. That happens because at time lag of 280 days there aren't as many data points to produce a valid average DCF value.

Similarly, for defining the uncertainties in peak's time lag, PDFs of both companies have been plotted and are represented by *Figure 19*

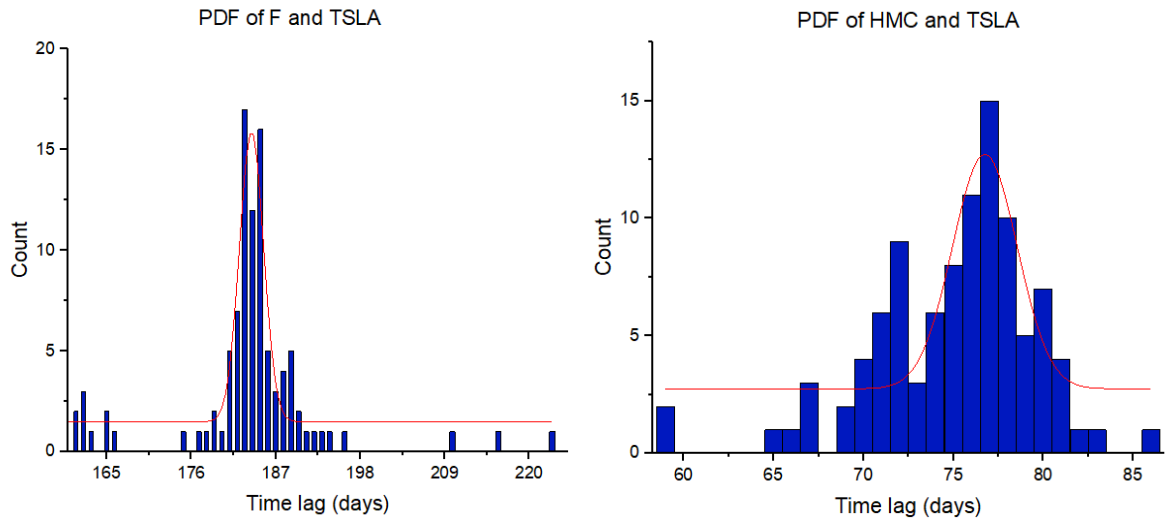


Figure 19 *PDFs of TSLA/F and TSLA/HMC*

And with the extraction of standard deviation in time lag, it was possible to fill in and analyze the table that provides the best correlation values and their time lags with uncertainties

Company	Peak value	Uncertainty	Time lag	Uncertainty
F/TSLA	0.9	1.03	187	3.7
HMC/TSLA	0.82	1.11	78	4.2

Table 2 *Best DCF values and their time lags with uncertainties of competitors*

The maximum DCF value time lags of F and HMC are not within the range of each others' uncertainties and therefore are not correlated. This is completely logical as both F and HMC are extremely large car manufacturers that aren't focused mainly on electric vehicles and make most of their profit from the sales of combustion engine vehicles, while TSLA on the other hand, only sells electric vehicles. From the analyzed competitors, F seems to be the better option when it comes to investing, it has a maximum DCF value of 0.9 ± 1.03 at time lag of 187 ± 3.7, also, as can be noticed it has the smallest error in time lag.

3.2 Power spectrum

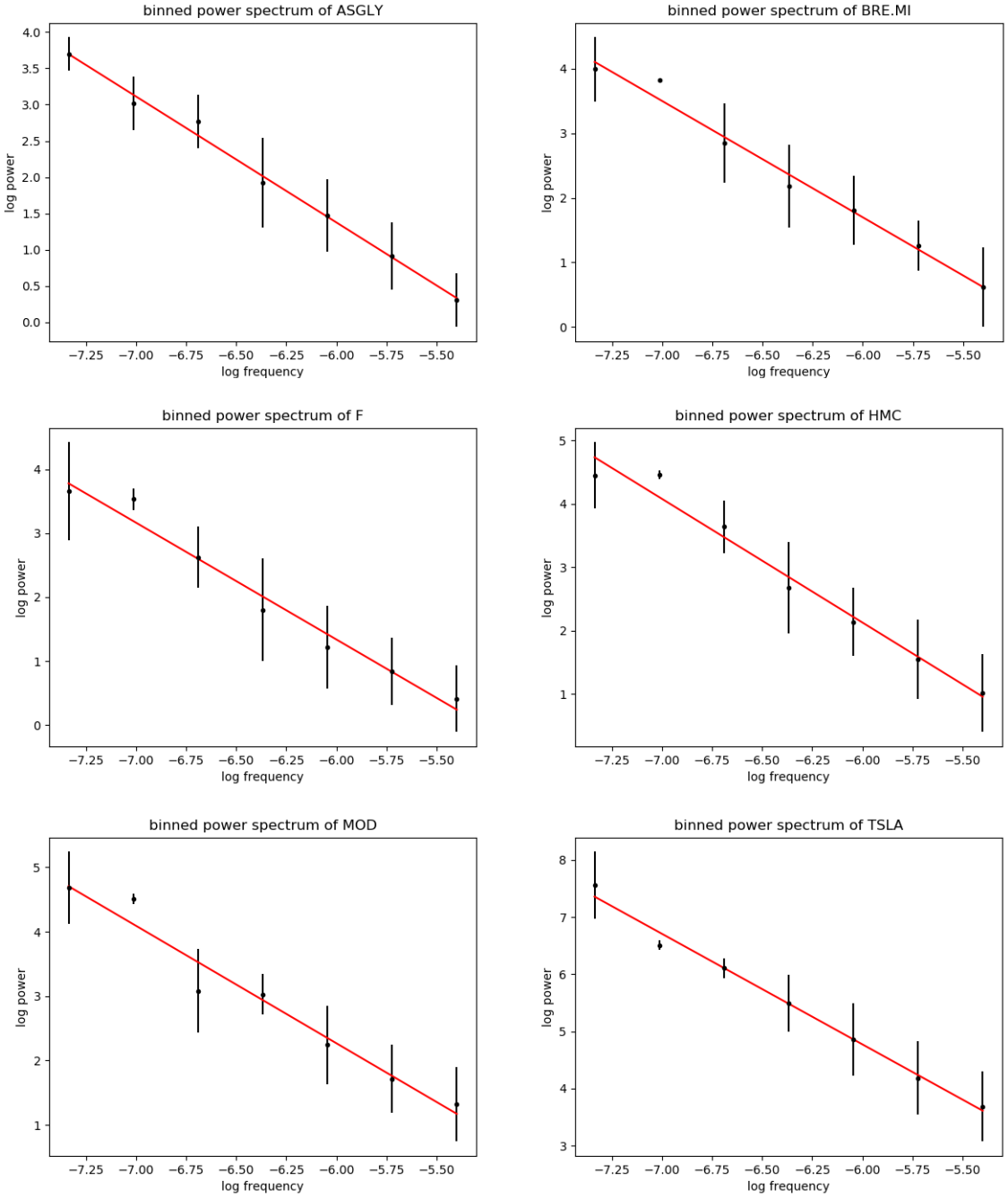


Figure 20 *Power spectrums of all companies with implemented Papadakis and Lawrence offset*

Figure 20 shows the already binned version of power spectrums with red line being the best linear fit to the data points. The I.E.Papadakis and A.Lawrence offset of 0.253 is already implemented in graphs and we get the unbiased estimate of logarithm of power density. The power spectra P(f) is proportional to $\frac{1}{f^{beta}}$, where beta defines the type of noise depending on its value [6]. Therefore, from the linear fit we have managed to extract the best fitting slope and intercept for each graph.

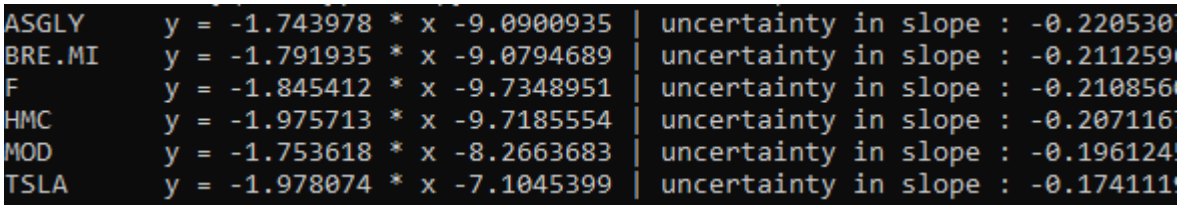


Figure 21 *values of the best fit linear graph and slopes' uncertainties*

The values for slopes and their uncertainties can be seen in *Figure 21*. The only known noise that falls within the range of all the slopes' uncertainties is the $beta = -2$ noise, which corresponds to Brownian noise or alternatively called random walk noise, the most famous example of it is the path traced by a molecule as it travels in a liquid or gas. This result supports the random walk hypothesis, which states that stock market prices evolve according to random walk. "The theory of the market as efficient and characterized as a random walk states that successive price changes in individual securities are independent and a series of stock price changes has no memory; thus, the past history of the series cannot be used to predict the future history."[7]

All in all, the power spectrum method is not sufficient for predicting stock market, as the variability of stock prices appear to fluctuate randomly and there isn't a single underlying force that moves the stock market. It seems to be impacted by many different factors that when working together produce something similar to a Random walk.

4 Conclusion

In conclusion, with the use of statistical methods mentioned in the method part, it's not possible to entirely predict the variability of stock market, although it's possible to provide the information of the trend that the stock might follow in the future with some credibility.

For the power spectrum of the companies, we got the slopes to be around -2 (ASGLY: -1.74 ± 0.21, BRE.MI: -1.79 ± 0.21, F: -1.84 ± 0.21, HMC: -1.98 ± 0.21, MOD: -1.75 ± 0.2, TSLA: -1.98 ± 0.17). This gradient corresponds to Brownian noise, also known as random walk noise, that's apparent in movement of the molecules. Also, this supports the random walk hypothesis, which states that stock market prices evolve according to random walk.

For the Discrete Correlation Functions' part, the best cross correlation value extracted was for MOD/TSLA the maximum peak value was 0.93 ± 0.92 at time lag 102 ± 6.1 (days). Also, for all the supplier companies, the DCF seemed to follow a similar trend (all the maximum peaks were distributed at the same time lag range (within each other's uncertainties)). Investigation of the suppliers and their maximum DCF values, lead to a conclusion that if a company is small and most of its profit comes from TSLA, it will have a stronger correlation and vice versa. For the competitors, the best correlation was for F/TSLA of 0.9 ± 1.03 at time lag of 187 ± 3.7 (days), both competitors had drastically different DCFs and that's due to the fact that both F and HMC focus mainly on combustion engine vehicles, so even if TSLA dominates the EV market, it's not much of a loss for the mentioned competitors.

The future of this project can go in a lot of different directions. It would be interesting to explore both DCFs and PSDs of different time scales. To see if there is some periodicity in shorter or longer time intervals, so it would be possible to provide both short- and long-term investors with essential information. Furthermore, investigating if strong correlation in a year interval fades away if longer time intervals are taken, would provide how credible our results are. Another area of interest, would be to dive deeper into the relation between the features of the company (market cap, number of clients, how much of the profit comes from the sales to the correlated company) and the maximum DCF value, this might reveal an underlying law for better understanding of stock market.

5 References

1. Ian McHardy, Anthony Lawson, Andrew Newsam, Alan P. Marscher, Andrei S. Sokolov, C. Megan Urry, Ann E. Wehrle (2007), Simultaneous X-ray and infrared variability in the quasar 3C273 – II. Confirmation of the correlation and X-ray lag, *Monthly Notices of the Royal Astronomical Society*, Volume 375, Issue 4, Pages 1521–1527
2. Edelson, R. A. & Krolik, J. H. (1988), The discrete correlation function - A new method for analyzing unevenly sampled variability data, *Astrophysical Journal*, Part 1 (ISSN 0004-637X), vol. 333,p. 646-659.
3. I. E. Papadakis, A. Lawrence (1933), Improved methods for power spectrum modelling of red noise, *Monthly Notices of the Royal Astronomical Society*, Volume 261, Issue 3, Pages 612–624
4. Timmer, J. & Koenig, M. (1995), On generating power law noise, Astronomy and Astrophysics, v.300, p.707
5. Pandit, Shail & Wasley, Charles & Zach, Tzachi. (2011). Information Externalities along the Supply Chain: The Economic Determinants of Suppliers' Stock Price Reaction to Their Major Customers' Earnings Announcements. Contemporary Accounting Research. 28. 1304 - 1343. 10.1111/j.1911-3846.2011.01092.x.
6. Procaccia, I. and Schuster, H.G. Functional renormalisation group theory of universal 1/f noise in dynamical systems. Phys Rev 28 A, 1210-12 (1983)
7. Eugene F. Fama (1995) Random Walks in Stock Market Prices, *Financial Analysts Journal*, 51:1, 75-80

6 Appendix

Reading the file function

```
#reads csv file using pandas dataframe
def fileread(file):
    data = pd.read_csv(file, usecols = ['Date', 'Close'])
    return data
```

Date conversion function

```
#converts from calendar dates to days taking some date as a reference
def dateconverter(date, times, timesreal, count):
    epochtime = datetime.fromisoformat(date).timestamp()                #converting from calendar date to seconds from the epoch start
    times.append(epochtime)                                             #adding the converted elements into a list
    days = (times[count] - times[0])/60/60/24                           #converting to days and taking first date as a reference point
    timesreal.append(int(days))                                         #adding days to a list
    return timesreal
```

Linear interpolation function

```
#function for linear interpolation
def lininterpolation(timesreal, pricelist, var):
    x1 = timesreal[var]
    x0 = timesreal[var - 1]
    y1 = pricelist[var]
    y0 = pricelist[var - 1]
    x = int(x0 + 1)
    y = (y0 * (x1 - x) + y1 * (x - x0)) / (x1 - x0)
    return x, y
```

Inserting missing days and prices function

```
#function which interpolates missing days and prices and adds them to a list
def insertprices(data, var, count, pricelist, timesreal, times):
    for date in data['Date']:                                           #reading all the data of the column labeled 'Date'
        realtime = dateconverter(date, times, timesreal, count)         #uses date conversion function
        if var >= 1:                                                    #if there are at least to days in the list
            difference = int(realtime[var] - realtime[var - 1])         #gets the difference between two dates
            while difference != 1:                                      #while difference is more than 1 day
                x, y = lininterpolation(realtime, pricelist, var)       #interpolates
                realtime.insert(var, x)                                 #inserts interpolated date to a list
                pricelist.insert(var, y)                                #inserts interpolated price to a list
                var += 1                                                #increases index variable
                difference = int(realtime[var] - realtime[var - 1])     #gets the difference and goes back to the start of the loop
        var += 1
        count += 1
    return realtime, pricelist
```

Standard deviation function

```
#standart deviation of prices
def stdev(pricelist):
    standartdeviation = np.std(pricelist)
    return standartdeviation
```

Looking for csv files that need to be handled

```
source = r"C:\Users\modestas\Desktop\dissertation"      #path
for files in os.walk(source, topdown = True):          #goes through all the files in the given path
    for file in files[2]:
        if file.find(".csv") != -1:                    #if file is csv file, then..
```

Writing new data to a file

```
name = (file[:file.find(".csv")] + "converted.csv")          #name of a new file
if os.path.isfile(name) == False:                            #if file doesn't already exist in the directory
    with open(name, 'w', newline = '') as f:                 #creates file
        for lis in range(len(daystesla)):
            wr = csv.writer(f)
            row = []
            if (lis == 0):                                   #if it's the first row
                row.append("Days")                           #adds name of column
                row.append("Prices")                         #adds name of column
                wr.writerow(row)                             #saves in the file
                row = []
            row.append(daystesla[lis])                       #adds days in the column
            row.append(pricestesla[lis])                     #adds prices in the column
            wr.writerow(row)                                 #saves in the file
```

Scatter DCF function

```
#function for getting unbinned DCF
def scatterfunc(steps, CC, lag, days1, prices1, prices2, average1, average2, std1, std2, N):
    while steps < 360:
        for day in days1:
            Tplus = day + steps
            Tminus = day - steps
            if Tplus < len(days1):
                DCF = ((prices1[day] - average1) * (prices2[Tplus] - average2)) / (std1 * std2)
                CC.append(DCF)
                lag.append(steps)
            if Tminus >= 0:
                DCF = ((prices1[day] - average1) * (prices2[Tminus] - average2)) / (std1 * std2)
                CC.append(DCF)
                lag.append(-steps)
        steps += 1
    return lag, CC
```

```
#loop runs till lag is smaller then number of days
#for each day in the list
#day plus positive time lag
#day plus negative time lag
#if day plus time lag is less than length of days
#calculates DCF
#puts DCF values into a list with all DCF values at all time lags
#puts time lag values into a list
#if day minus time lag is more than 0 then..
#calculates DCF
#puts DCF values into a list with all DCF values at all time lags
#puts time lag values into a list
#time lag increases by one
```

Calculating the average DCF values at each time lag function

```
#bins up all the DCF values at each time lag and gets one average value
def DCFwerr(means, errors, lags, lagcount, lag, CC, N):
    while lagcount < 360:                                   #loop runs till lag is smaller then number of days
        binsize = N                                         #binning up of time lags, where N is the variable that defines the size o
        CCs = []
        while binsize >= 1:
            index = 0
            for i in lag:                                   #for every time lag in lags list
                if i == lagcount:                           #if time lag is equal to lagcount variable
                    CCs.append(CC[index])                   #append all DCF values at that time lag to CCs list
                index += 1
            lagcount += 1                                   #lagcount variable increases
            binsize -= 1                                    #binsize value decreases
        if len(CCs) != 0:                                   #if list is not empty
            avrg = sum(CCs)/len(CCs)                        #calculates the average of that list
            means.append(avrg)                              #adds that average to another list of averages
            std = np.std(CCs)
            errors.append(std)
            lags.append(lagcount - 1)                       #appends time lag value

    return lags, means, errors
```

Finding all csv files that are not Tesla

```
source = r"C:\Users\modestas\Desktop\dissertation"                  #this is the path to where I keep all my code and csv files
for files in os.walk(source, topdown = True):                       #goes through every file in the source
    for file2 in files[2]:                                          #takes only the name of every file
        if file2.find("converted") != -1:                           #if it finds file that has "converted" in its' name
            if file2.find("TSLA") == -1:                            #if file is not TESLA file
                path = os.path.join(source, file2[:file2.find("converted")])    #goes into a folder named as the file read
                steps = 1                                           #default values needed for functions (time lag at the beginning)
```

Plotting DCF graphs

```
plt.scatter(lag, CC, s = 0.5)
plt.title(" Discrete Cross Correlation Function of " + file1[:file1.find("con")] + " and " + file2[:file2.find("con")] + " of bin size " + str(N)
plt.ylabel('DCF values')
plt.xlabel('time lag (days)')
plt.savefig(os.path.join(path, "Cross correlation function of " + file1[:file1.find("con")] + " and " + file2[:file2.find("con")] + str(N) + ".pn
plt.close()
print (file1, " and ", file2)
print ("peak value ", max(means), "error", errors[(means.index(max(means)))], " at ", lags[(means.index(max(means)))])
plt.errorbar(lags, means, yerr = errors, ecolor = 'black', fmt = '.k')
plt.title(" Discrete Cross Correlation Function of " + file1[:file1.find("con")] + " and " + file2[:file2.find("con")] + " of bin size " + str(N)

plt.ylabel('DCF values')
plt.xlabel('time lag (days)')
plt.savefig(os.path.join(path, "Discrete cross correlation function of " + file1[:file1.find("con")] + " and " + file2[:file2.find("con")] + str(
plt.close()
```

Converting to logarithms function

```
def logconv(frequency, logf, power, fourierprices):
    count = 0
    for f in frequency:
        if f != 0:
            log10f = math.log10(f)
            logf.append(log10f)
            log10power = math.log10(abs(fourierprices[count] ** 2))
            power.append(log10power)
        count += 1

    return power, logf
```

Average bin method function

```
def binpowerspectrum(bins, std, binfrequency, xerr):
    N = 1
    index = 0
    onebin = []
    for i in logf:
        if i < (logf[0] + (binsize * N)):   #if datapoint is inside the bin
            onebin.append(power[index])     #append that datapoint to a list "onebin"
        if i >= (logf[0] + (binsize * N)):  #if datapoint is outside the bin
            onebin.append(power[index])     #append the last datapoint that is inside the bin
            bins.append((sum(onebin)/len(onebin)))#append a mean value of binned values to another list "bins"
            std.append(np.std(onebin)) #getting standart deviation of the binned values
            onebin = []                #emptying the bin
            binfrequency.append(logf[0] + (binsize * (N - 1) + (((binsize * N) - (binsize * (N - 1))) / 2))) #getting the middle value of each bi
            N += 1    #next bin
        index += 1    #index increases

    return bins, binfrequency, std
```

Papadakis and Lawrence offset

```
bins, binfrequency, std = binpowerspectrum(bins, std, binfrequency, xerr)
binsfinal = []
for bi in bins:
    binsfinal.append(bi + 0.253)
xs = np.array(binfrequency, dtype=np.float64)
ys = np.array(binsfinal, dtype=np.float64)

m, b = best_fit_slope_and_intercept(xs, ys)

regression_line = [(m*x)+b for x in xs]
```

Power spectrum average bin graph

```
plt.errorbar(binfrequency, binsfinal, std, ecolor = 'black', fmt = '.k')
plt.title("Binned up Power Spectrum of " + file[:file.find("con")])
plt.ylabel("Log Power")
plt.xlabel("Log Frequency")
plt.plot(xs, regression_line, color='red')
plt.savefig(os.path.join(path, "binned power spectrum of " + file[:file.find("con")] + ".png"))
plt.close()
```

Random sampling stock prices function

```
def convlists(data):
    df = pd.DataFrame(data)
    lists = df.values
    days = []
    prices = []
    for lis in lists:
        rand = random.random()
        if rand <= 0.7:
            days.append(int(lis[0]))
            prices.append(lis[1])
    return days, prices
```

Calculating DCF of randomly sampled graphs function

```
#function for getting unbinned DCF
def scatterfunc(steps, CC, lag, days1, days2, prices1, prices2, average1, average2, std1, std2, N):
    while steps < 360:                                                          #loop runs till lag is smaller t
        for day in days1:                                                       #for each day in the list
            Tplus = day + steps                                                 #day plus positive time lag
            Tminus = day - steps                                                #day plus negative time lag
            if Tplus < days1[len(days1) - 1]:
                if (Tplus in days2) == True:

                    if (day in days1) == True:                                  #if day plus time lag is less than
                        DCF = ((prices1[days1.index(day)] - average1) * (prices2[days2.index(Tplus)] - average2)) / (std1 * std2)
                        CC.append(DCF)                                          #puts DCF values into a
                        lag.append(steps)                                       #puts time lag values in
            if Tminus >= 0:
                if (Tminus in days2) == True:

                    if (day in days1) == True:
                                                                                #if day minus time lag is more than 0 t
                        DCF = ((prices1[days1.index(day)] - average1) * (prices2[days2.index(Tminus)] - average2)) / (std1 * std2)
                        CC.append(DCF)                                          #puts DCF values into a
                        lag.append(-steps)                                      #puts time lag values in
        steps += 1                                                              #time lag increases by one
    return lag, CC
```

Calculating 100 DCF graphs

```
while variable < 100:                                       #gets DCF graph 100 times
    path = os.path.join(source, file2[:file2.find("converted")])    #goes into a folder named as the file read
    steps = 1                                               #default values needed for functions (time
    CC = []
    lag = []
    means = []
    errors = []
    lags = []
    lagcount = -(720/2 - 1)
    N = 11                                                  #bin size of time lags


    data1 = fileread(file1)
    data2 = fileread(file2)

    days1, prices1 = convlists(data1)
    days2, prices2 = convlists(data2)

    std1, std2 = stdev(prices1, prices2)

    average1, average2 = meanfunc(prices1, prices2)
```

Adding a maximum DCF value to a new list for getting PDF

```
    maxvalues.append(max(means))
    maxtimelags.append(lags[(means.index(max(means)))])
    print(file2, " loop " ,variable)
    variable += 1
print (maxtimelags)
height = []
x = []
for values in maxtimelags:
    if (values in x) == False:
        count = maxtimelags.count(values)
        height.append(count)
        x.append(values)
```

Writing PDF values to a file

```
name = (file2[:file2.find("converted.csv")] + "_PDF.csv")            #name of a new file
if os.path.isfile(name) == False:                                   #if file doesn't already exist in the directory
    with open(name, 'w', newline = '') as f:                        #creates file
        for lis in range(len(height)):
            wr = csv.writer(f)
            row = []
            if (lis == 0):                                          #if it's the first row
                row.append("time lag")                                  #adds name of column
                row.append("Height")                                #adds name of column
                wr.writerow(row)                                    #saves in the file
                row = []
            row.append(x[lis])                                  #adds days in the column
            row.append(height[lis])                                 #adds prices in the column
            wr.writerow(row)                                        #saves in the file
```

Plotting PDF

```
plt.bar(x, height)
plt.title("Probability Density Function of " + file1[:file1.find("con")] + " and " + file2[:file2.find("con")])
plt.ylabel('maximum peak value count')
plt.xlabel("time lag (days)")
plt.savefig(os.path.join(path, "PDF of " + file1[:file1.find("con")] + " and " + file2[:file2.find("con")] + ".png"))
plt.close()
```

Producing ACF of different bin sizes

```
N = 1
source = r"C:\Users\modestas\Desktop\dissertation"                   #this is the path to where I keep all my code and csv files
while N <= 20:
    for files in os.walk(source, topdown = True):                       #goes through every file in the source
        for file2 in files[2]:                                          #takes only the name of every file
            if file2.find("converted") != -1:                           #if it finds file that has "converted" in its' name
                file1 = file2
                path = os.path.join(source, file2[:file2.find("converted")])    #goes into a folder named as the file read
```