IS602 Term Project Report

Student: Justin Hink
Instructor: Andrew Hernandez
Due Date: December 23, 2014

Introduction and Motivation

The goal of this term project is twofold. First, I aim to demonstrate competency in the Python programming language, including the numerical analysis packages we have covered in the course. Second, I wish to gain exposure to the world of financial data analysis in a rather basic but still interesting way. To do the later I used real world financial data and perform a number of analysis tasks. These will be summarized below.

In the hopes of increasing clarity, I have broken up my project into 4 main sections. These sections are:

1. Section 1 – Demonstration of competency with underlying Python tools and programming techniques.
2. Section 2 – A retrospective look at aggregated market behaviour.
3. Section 3 - Basic frequency domain analysis of financial instruments
4. Section 4 (Bonus, not previously outlined in Project Spec) – Creating and back testing a trading algorithm based on the techniques explored in Section 3. Note that originally this bonus section was going to look at a Kalman filter algorithm. However I decided that building upon the results in previous sections made more sense than introducing an entirely new technique here. A Kalman filter approach is still of interest but it will not be implemented in this bonus section.

Instructions to Run Project Code

Each section of the project has its own Python file that should be run isolation. The files are independent programs and do not rely on each other in any way. These files can be run from any standard Python development environment (command line, IDE such as PyCharm etc).

The names of these 4 key source files are:

- finalproj_part1.py
- finalproj_part2.py
- finalproj_part3.py
- finalproj_part4.py

Section 2 also requires a CSV file titled *sp_historical.csv* (included in submission package).

Library/Package Dependencies

The project operates within the standard Python data science stack that we've used in throughout the course plus one addition for sections 3 and 4. The required package installations to run the software are:

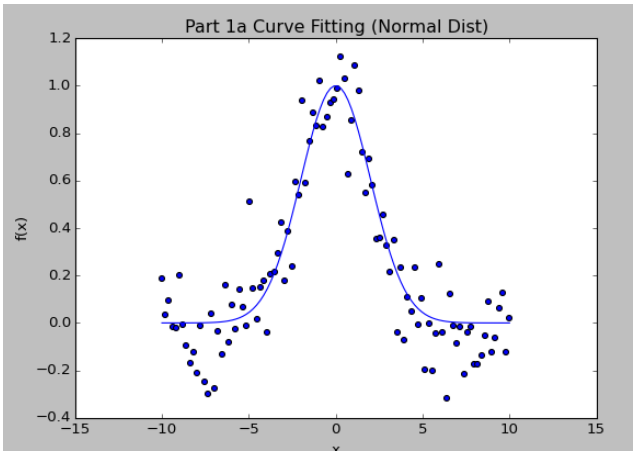
- NumPy
- SciPy
- Pandas
- Matplotlib
- Zipline (Available for download here: https://pypi.python.org/pypi/zipline)

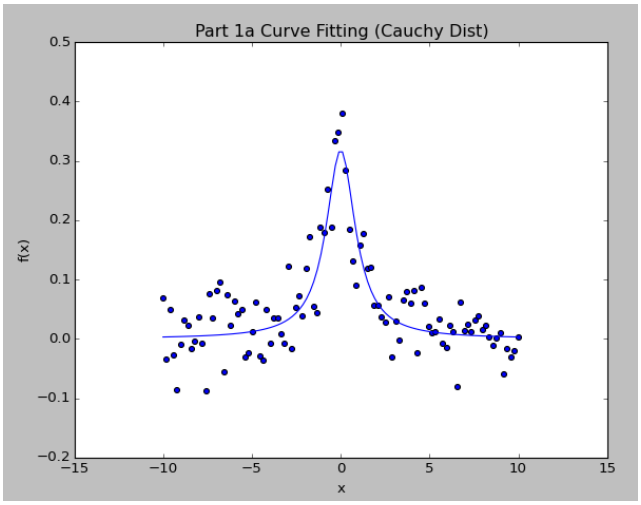
Results and Discussion

Section 1: Core Competency in Python

This section is a rather basic demonstration of the underlying skills and knowledge of specific Python packages that will be used in the project's more interesting later parts. Specifically, in this section I performed the following tasks:

1) Generated predefined curves (ex $\mathcal{N}(\mu, \sigma^2)$), add random noise to the curve and use the libraries available in SciPy to attempt to fit the noisy data with known statistical distributions. The following charts are output by the code in this section and demonstrate the functionality.



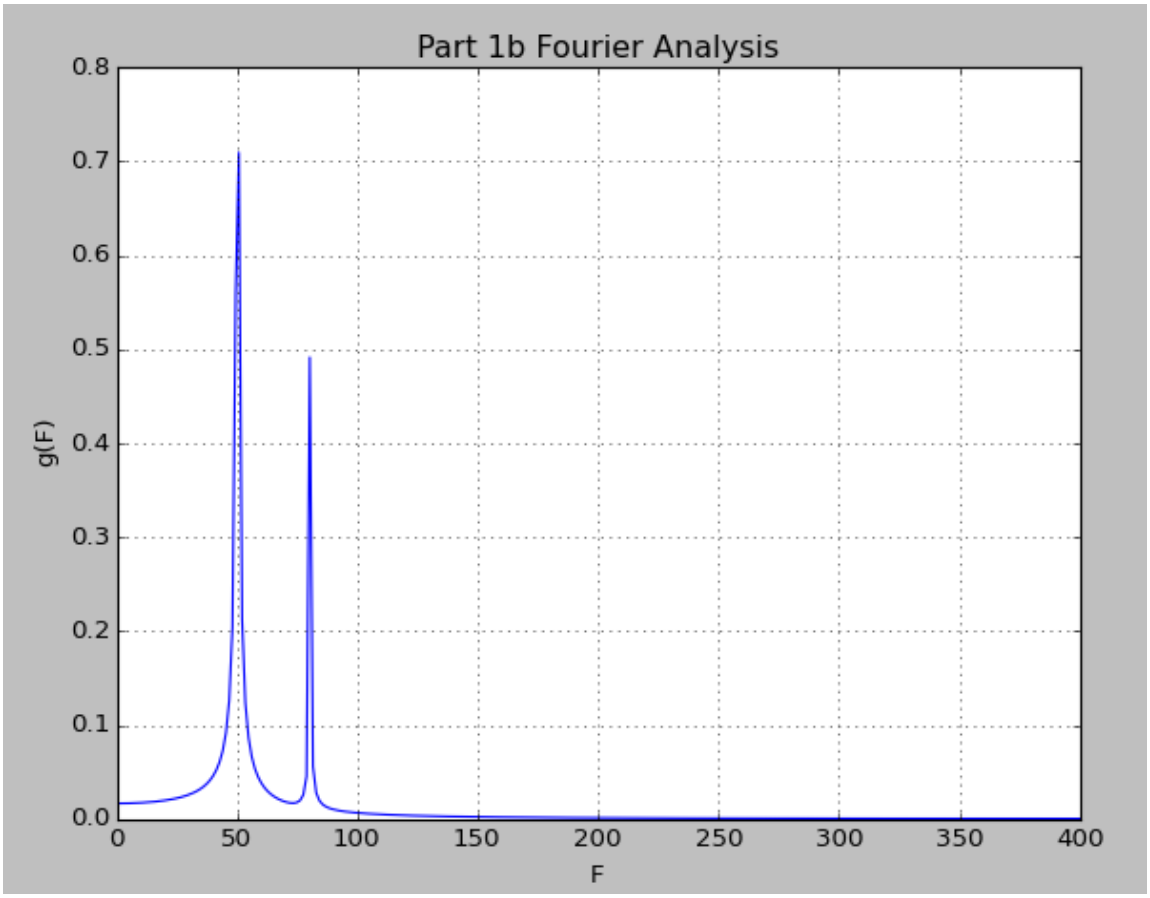


2) Used pre-built Fast Fourier Transform (hereafter FFT) algorithms available in SciPy to analyze a basic periodic function with 2 easily discernable periods. Specifically I analyzed the spectrum for the function:

$$g(t) = \sin(2\pi f_1 t) + \frac{1}{2}\sin(2\pi f_2 t)$$

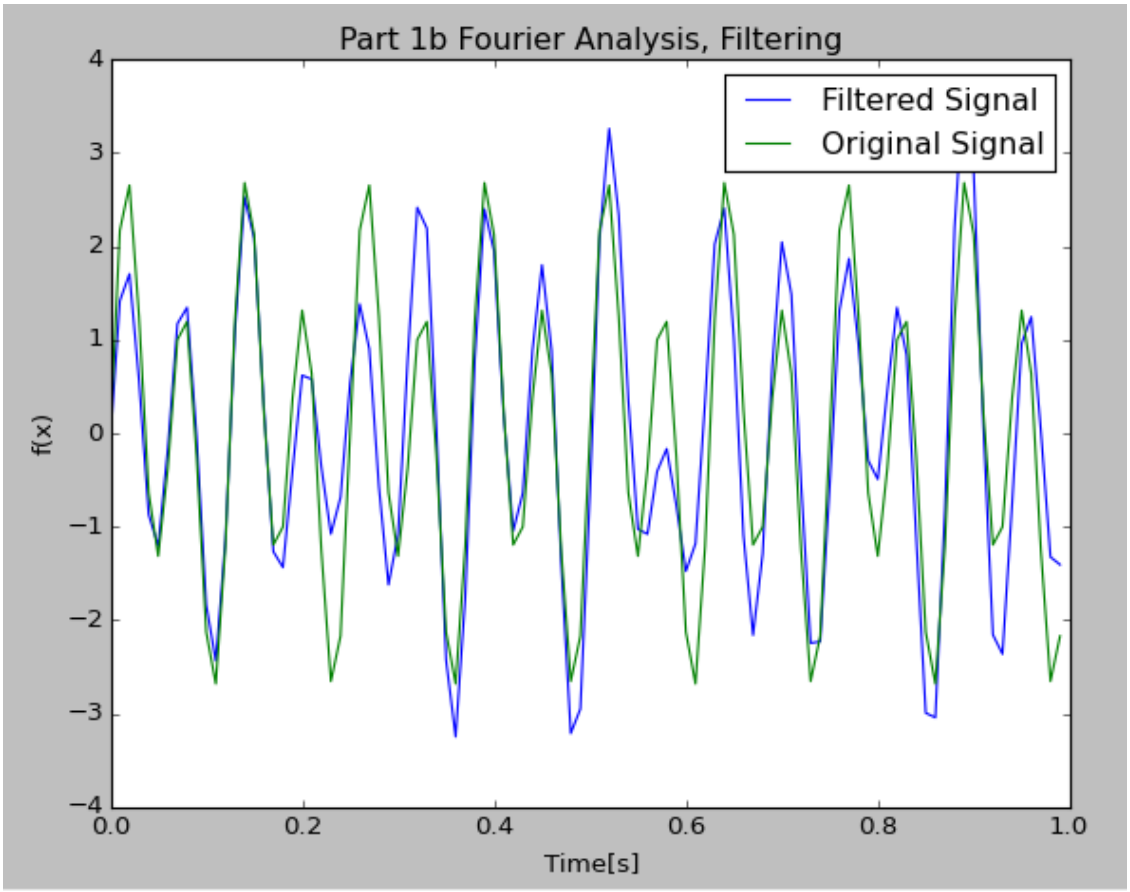
$$f_1 = 50\ Hz$$
$$f_2 = 80\ Hz$$

The following plot shows the frequency spectrum for the above function.



As we would expect, 2 disctinct spikes show up in our frequency domain. These two spikes correspond with the frequencies of the two underlying sinusoidal signal components.

In this section I also went through the exercise of Gaussian noise to another distinctly periodic time series signal, filtered the noise out in the frequency domain with a band pass filter and then reconstructed the signal to see how well it would match the original signal. The resulting plot of this exercise is shown below.

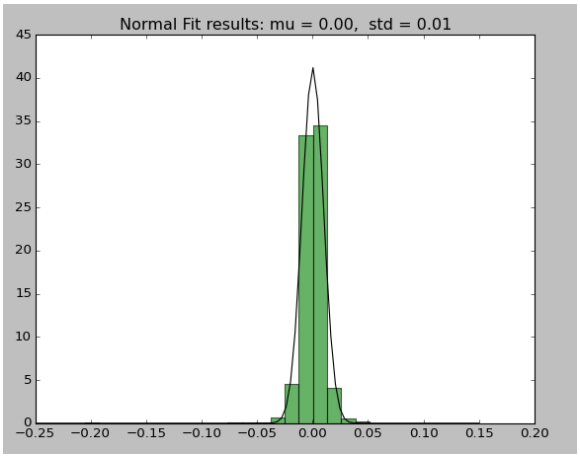


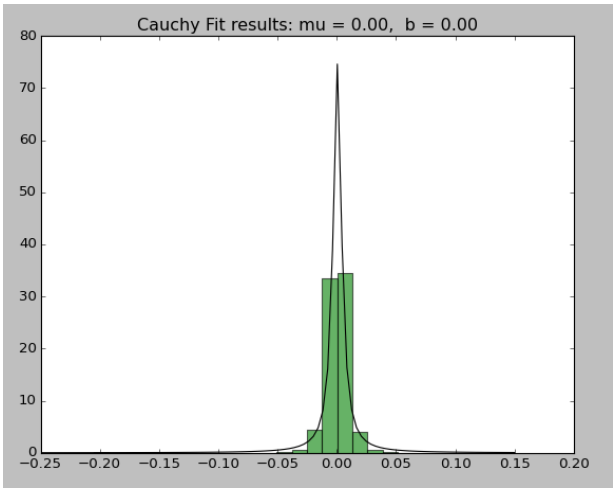
I was able to reconstruct the original, noiseless signal successfully with this technique. It should also be noted that "designing" a better filter here creates and even better match between the original and filtered/reconstructed signals.

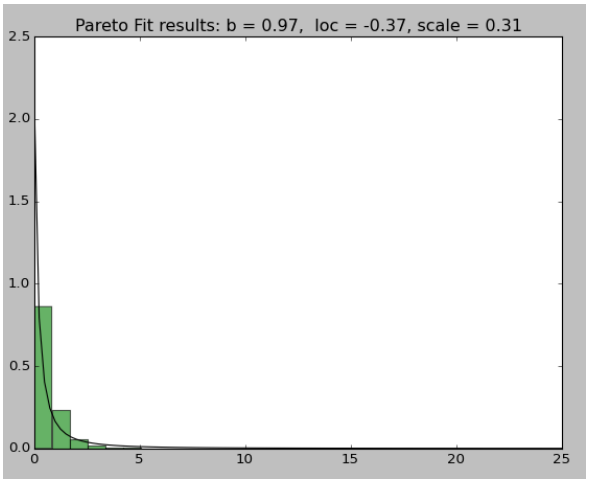
Section 2: Historical Market Behaviour

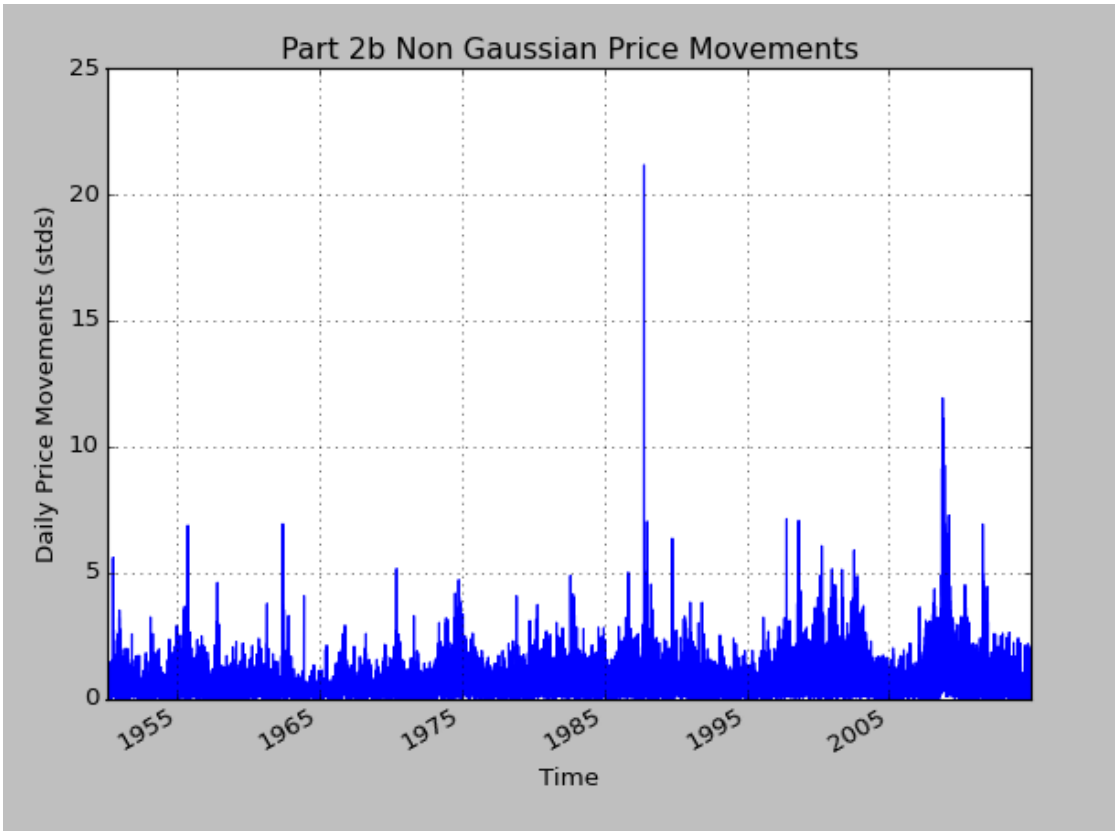
In this section, I look at past market behaviour. Specifically, I showed how distributions of finite variance do not always provide an accurate picture of market behaviour. To do this, I fit historical market data (using the S&P 500 as my proxy) to a number of statistical distributions ranging from the basic Normal distribution to the Cauchy Distribution of which there are no defined statistical moments. In the common parlance, I showed that the tails of the PDF defining market movements are thicker (fatter) than found in a Normal distribution.

The histograms and curve-fits for Normal, Cauchy and Pareto Distributions can be seen below as well as a time series look at price movements in terms of their scale in standard deviations. The Black Monday spike in 1987 stands out clearly. It's a 21 σ event which should shed immediate doubt on the validity of the assumption that a Normal distribution or the concept of "standard deviation" are useful measures of a signal driven by non-physical (but rather behavioural) factors.


Normal Fit results: mu = 0.00, std = 0.01
45
40
35
30
25
20
15
10
5
0
−0.25
−0.20
−0.15
−0.10
−0.05
0.00
0.05
0.10
0.15
0.20


Cauchy Fit results: mu = 0.00, b = 0.00
80
70
60
50
40
30
20
10
0
−0.25
−0.20
−0.15
−0.10
−0.05
0.00
0.05
0.10
0.15
0.20

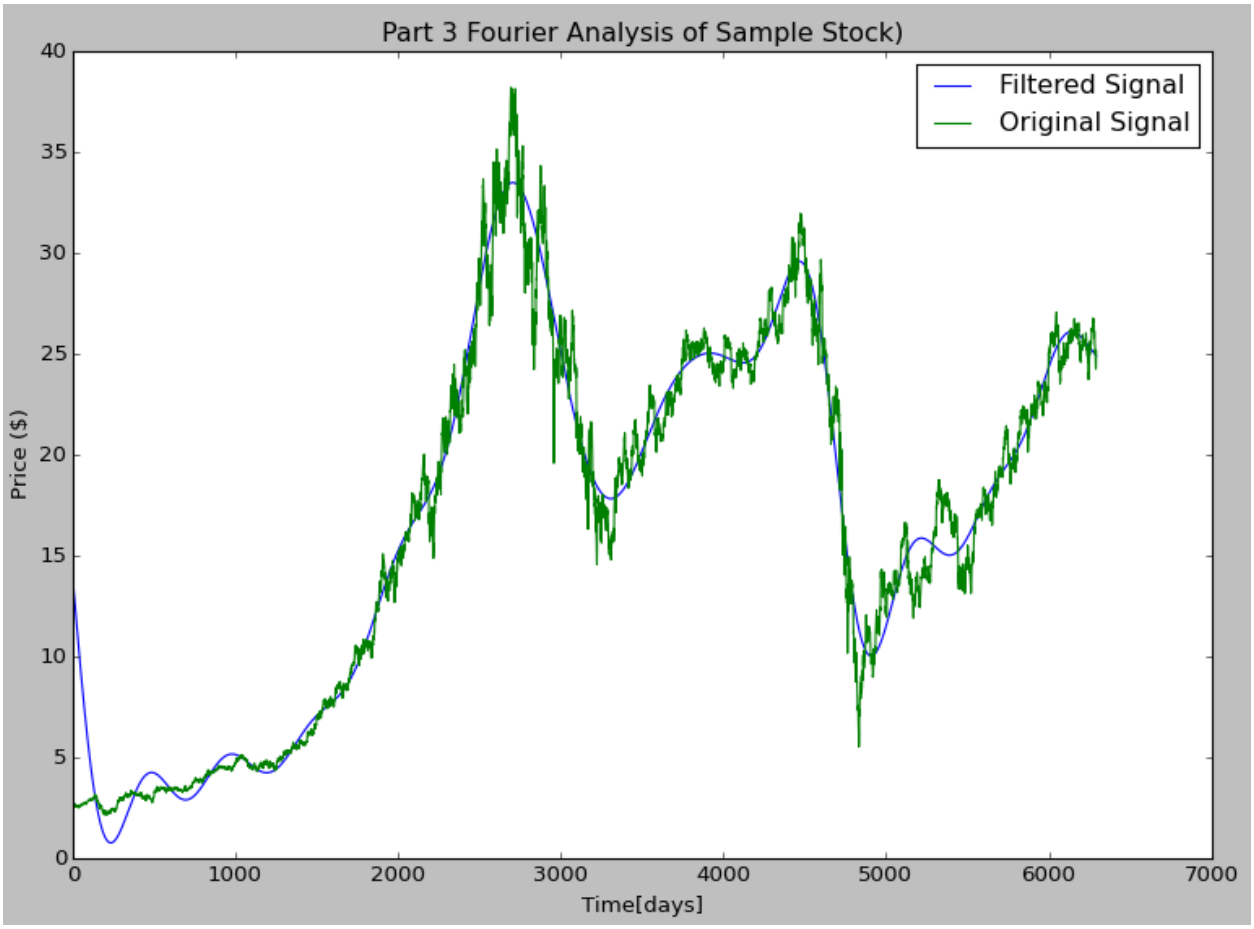

Pareto Fit results: b = 0.97, loc = -0.37, scale = 0.31
2.5
2.0
1.5
1.0
0.5
0.0
0
5
10
15
20
25



Section 3: Frequency Domain Analysis of Financial Data

In section 3 of the project I took the techniques used in Section 1 to filter a periodic function and applied them to real world financial market data. I was unsure how this would work/look due to the periodicity of market signals being far less apparent than toy sinusoids. However I thought filtering out the highest frequencies I the market data might clean our signal and give us something to make better decisions with.

To filter out the highest frequency noise, I used a simple low pass filter. The parameters of the filter cut-off were not determined by anything more sophisticated than an educated guess. This setting is inherently tweakable. The following plot shows a stock price's actual signal vs the signal "de-noised" (high frequencies removed). Note: the stock analyzed here is General Electric (GE).



The result is a smoothed signal that fluctuates less than the actual stock price vs time plot. Whether this is actually useful is not entirely clear but it does accomplish what I set out to do in this section - remove the high frequencies of the signal resulting in a signal that fluctuates less on a day to day basis when looking at the data retrospectively.

Section 4 (Bonus): A First Pass at a FFT/Low Pass Filter Based Trading Algorithm

In section 4 I took the techniques developed in section 3 and applied them in the context of a basic trading strategy. To test the algorithm I used the "Zipline" backtesting engine.

The logical steps of the algorithm is as follows:

For each day:

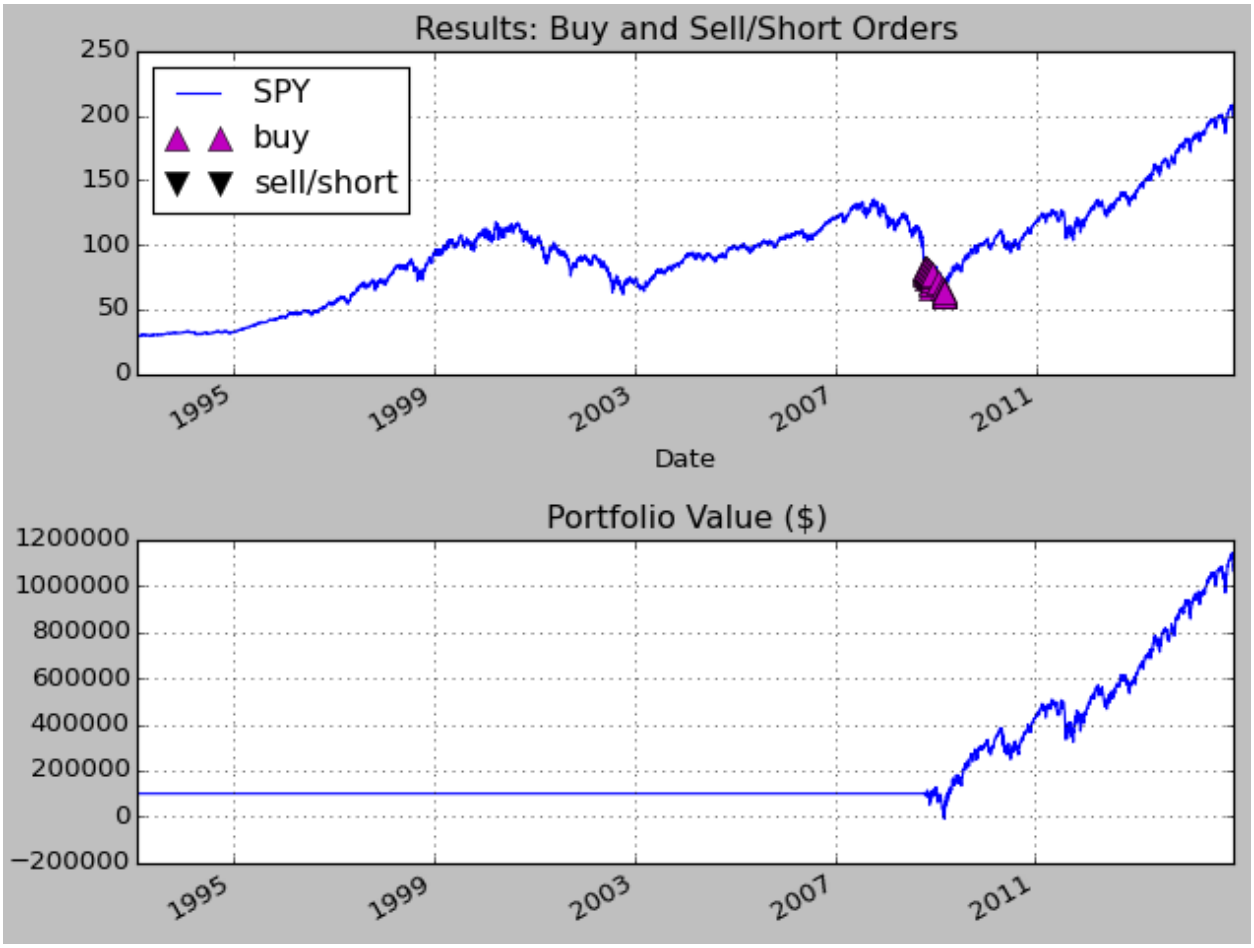
- Use historical data to create a frequency domain representation of the past
- Low pass filter this spectrum

- Recreate a time-series signal
- Compare the current market price to what the recreated (filtered/smoothed) signal think the current price should be.
- If the market price is 20% higher than the smoothed signal's current price, sell a fixed amount of this stock. If no stock is owned, short sell that same fixed amount.
  If the market price is 20% lower than the smoothed signal's current price, buy a fixed amount of this stock.

To generate a reasonable signal to create a filtered "true" signal from, trading is not done for the first year of a stock's existence.

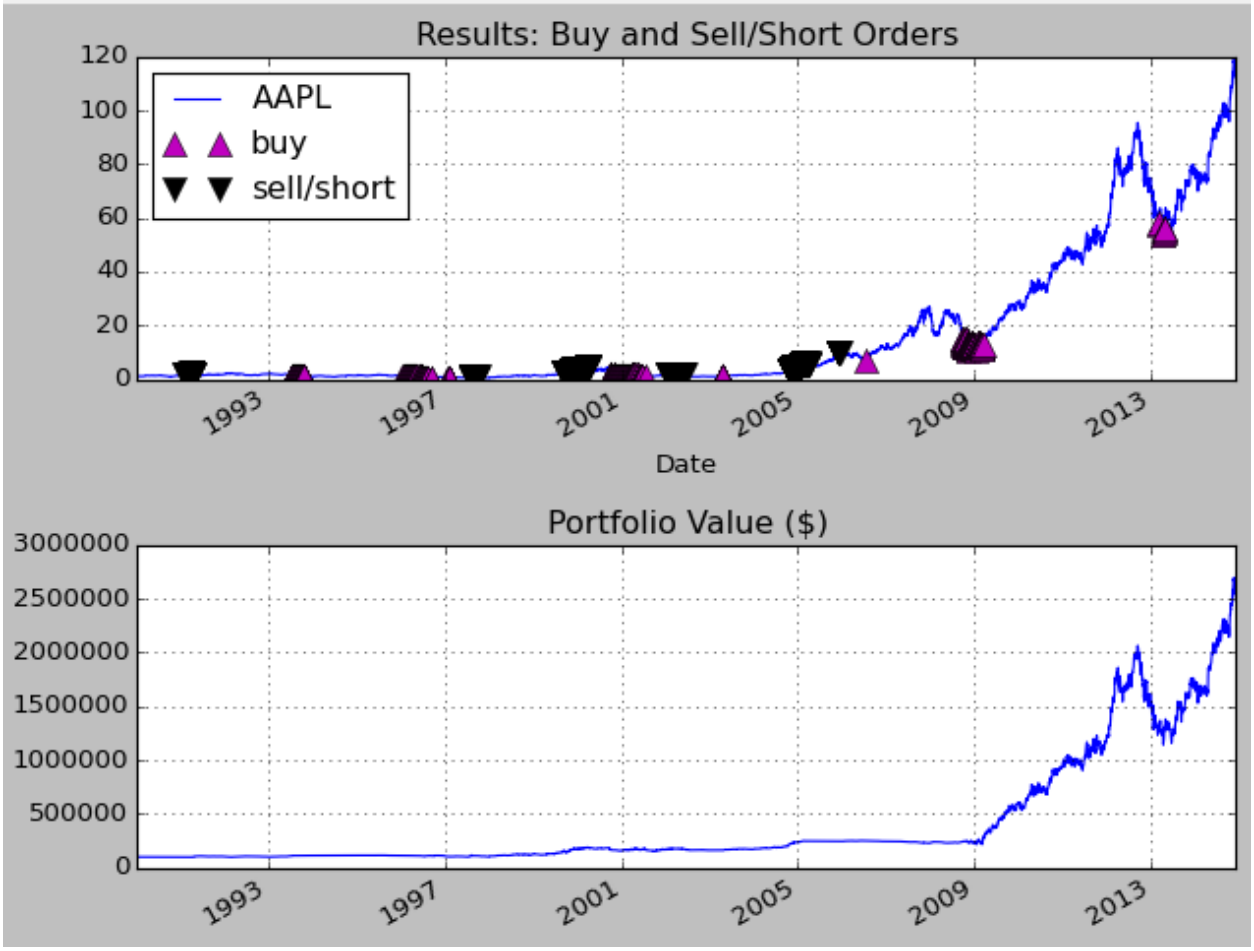
I ran a number of stocks/ETFs through the algorithm and the results were actually more encouraging than I was expecting for such a simple trading mechanism.

Here is what the back test for SPDR S&P 500 ETF Trust (SPY) looked like:



The top chart is simply the stock price with our buy and sell/short orders marked. In this instance, the algorithm was fairly agnostic on SPY until it detected an opportunity in 2008 at which point it bought a large amount of the fund. The result of this was an increase of portfolio value from $100,000 (our starting point) to $1,200,000.

A run of the algorithm on Apple stock looks as follows:



This time the algorithm was much more active in recommending buy and sell/short orders. It successfully identified local minimums in price during 2009 and 2013 leading to a nice result ($100k to $2.7M).

As I mentioned, this algorithm is just a first pass at examining this type of technique. It's nowhere near ready to use in a real-world environment. A number of improvements can be made for version 1.0 that would make this more robust. Some of these include (but are not limited to):

- Expand to include a portfolio of stocks/securities/financial instruments instead of the simplistic cash + one stock approach the algorithm uses right now.
- Scale bet sizes based the perceived "edge" over the market instead of betting a fixed amount. Something Kelly Criterion-ish would be a first step here.
- Employ some sort of risk management controls. There are literally none implemented and you can certainly short yourself to absolute death with this approach. It's very risky.