Devin Hurley
Dr. Vernizzi
May 3rd, 2013
The Fourier Transform and stock prices

Introduction:

Predicting the behavior of prices in the market has been an ongoing effort since the mid 19th century. Louis Bachelier was a French mathematician who attempted to map the prices changes of stocks as a normal distribution. Since then, many people have attempted to build upon his methods, many, including famous mathematician Benoit Mandelbrot, have denounced the normal distribution of price changes in favor of the Levy distribution. The Levy distribution applies probabilities to random variables and represents this information as a graph.

My project is much simpler. Heating oil prices are supposed to have a fairly uniform distribution. They will go up during the winter when demand is high and down in the summer when demand is low. Or at least this is how they should perform in theory. In a perfect world, where inflation is constant and the percent that prices increase or decrease is constant, the plot of these prices would look like a simple sinusoid, and predicting whether prices of heating oil will go up or down becomes something very simple. Just look at the graph.

Oil prices and other stock and commodity prices are not simple sinusoids. They are dependent on outside forces. These forces include foreign conflicts, environmental movements and the consumers' overall attitude at the time. Due to these forces, even with the probabilities of random variables that might affect the markets, it becomes very difficult to accurately predict the price of a stock.

Linear regression is one technique I used to extrapolate / interpolate data in my graph. This is a good technique for interpolation, but it is still a very rough approximation of where the market is going. Using the Fourier Transform, we can 'clean' our data and use these points to more accurately predict when and where the price of heating oil will go up or down. We already have a basic intuition of how our prediction should look (Something like a sinusoid) but we should also understand that this is not accurate if that is all we get.

When I approached Dr. Vernizzi about this project he suggested that I use the Fourier Transform. The Fourier Transform changes a graph by changing the plot from a function of amplitude over time to a function of amplitude over frequency. By using the Fourier Transform and the subsequent Inverse Fourier Transform, I was able to 'clean' my data using the first 20 points of the Fourier Transform. I then used a polynomial best fit for the original data and extrapolated it to the following 28 months and found a polynomial best fit for the 'cleaned' data plus the following 28 months. I then combined the two fits and this gave me an approximate reading on where oil prices ought to go based on their previous behavior.

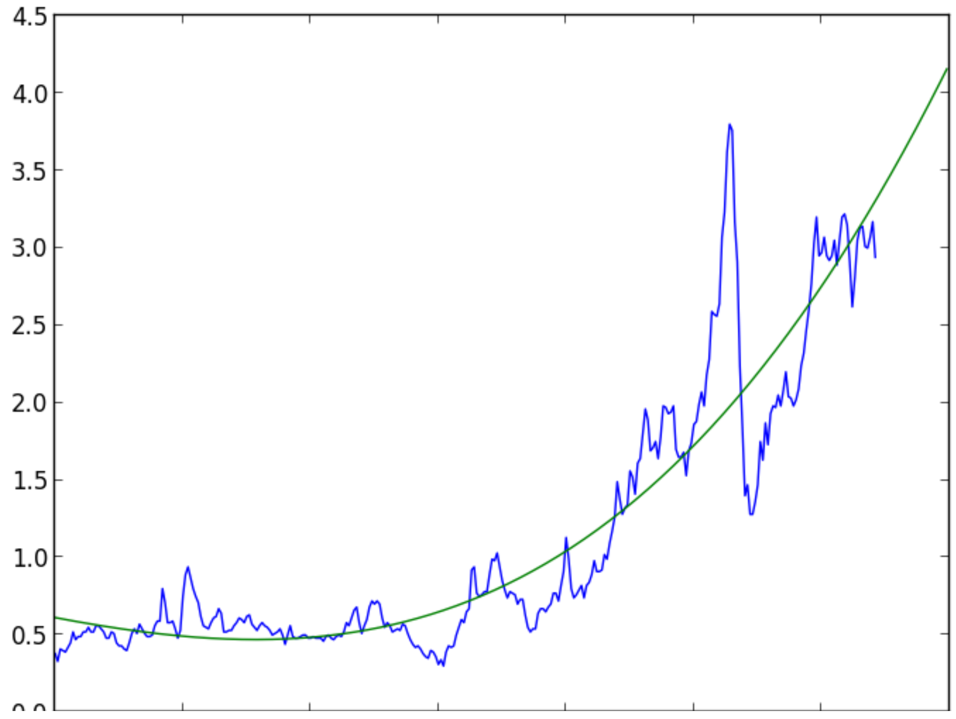
Forecasting something based on it's previous behavior is, as my economics textbook author put it, "like driving a car while blindfolded and being given directions from someone behind you looking out the rear window." Based off of this description, this is fairly accurate. However, we can conclude that our prediction is fairly accurate, not just by comparing it to the real values, but by intuitively knowing how heating oil is produced. We know that oil is getting scarcer. When there is not a lot of supply, demand goes up. When demand goes up, prices tend to go up. This, and rates of inflation account for this increase in prices.

Analysis:

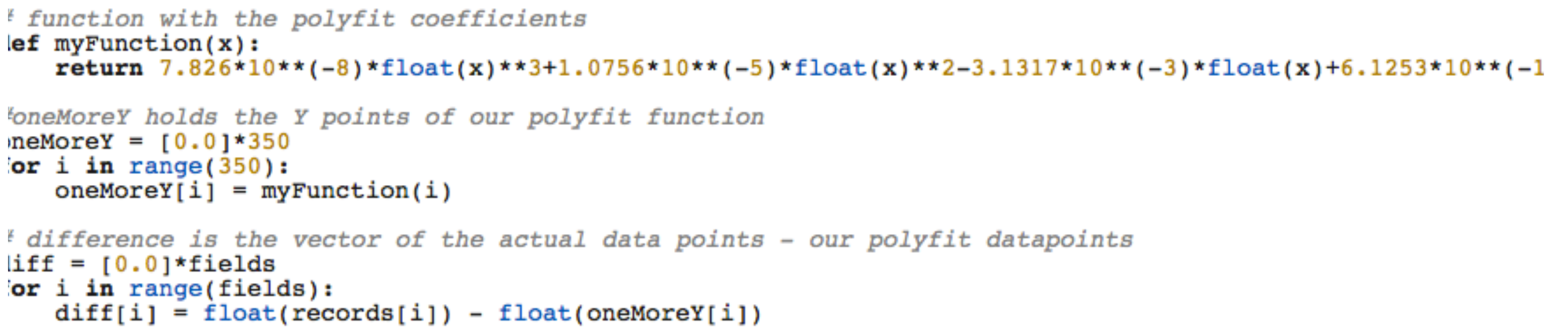
I began by looking for data on the subject. My data on heating oil prices goes from June 30th, 1986 and samples the price once every month from then up until March 31st, 2013. I was interested in whether a cubic polynomial would fit this data better than a quadratic. As it turns out, they look almost identical. The coefficient for the cubic function was 7.826*10^(-8) = 0.000000007826… This is very close to zero, so we can assume that the cubic polynomial had almost no effect on the data. However, since the coefficient was not outrageously small, I decided that using the cubic would not be terrible. Using the polyfit function we developed in class, I found the entire cubic polynomial to be of the form

$$0.61253 - 0.0031317\,x + 0.000010756\,x^2 + 7.826 \times 10^{-8}\,x^3$$

The following is the graph of our fitted curve. I also extended the curve out to the 28 months we would like to predict up to. Months are on the X axis and price on the Y.



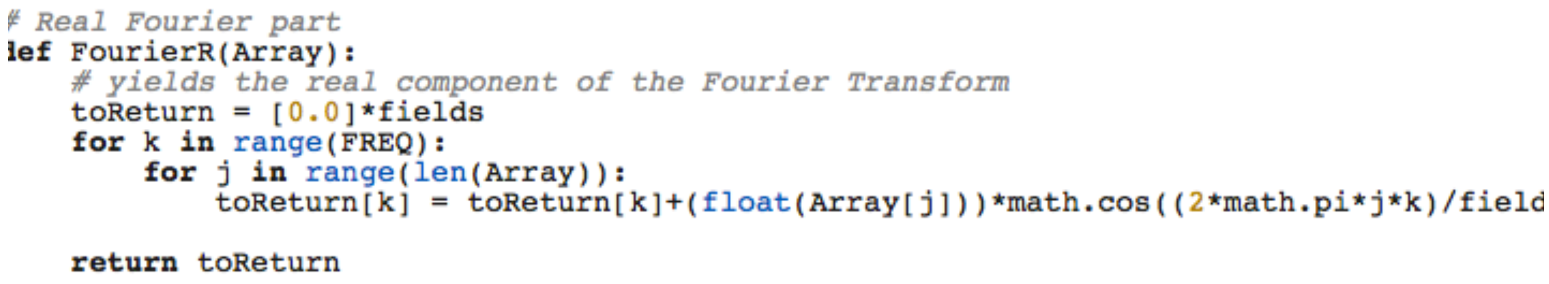
I then took the difference of the polynomial function at all 323 months and the real data. This is the code that generated this difference:



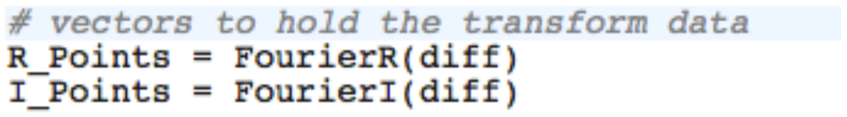
```
# function with the polyfit coefficients
ef myFunction(x):
    return 7.826*10**(-8)*float(x)**3+1.0756*10**(-5)*float(x)**2-3.1317*10**(-3)*float(x)+6.1253*10**(-1

#oneMoreY holds the Y points of our polyfit function
neMoreY = [0.0]*350
or i in range(350):
    oneMoreY[i] = myFunction(i)

# difference is the vector of the actual data points - our polyfit datapoints
iff = [0.0]*fields
or i in range(fields):
    diff[i] = float(records[i]) - float(oneMoreY[i])
```

Interestingly, the stock seems to behave more predictably if we do not clean any points.

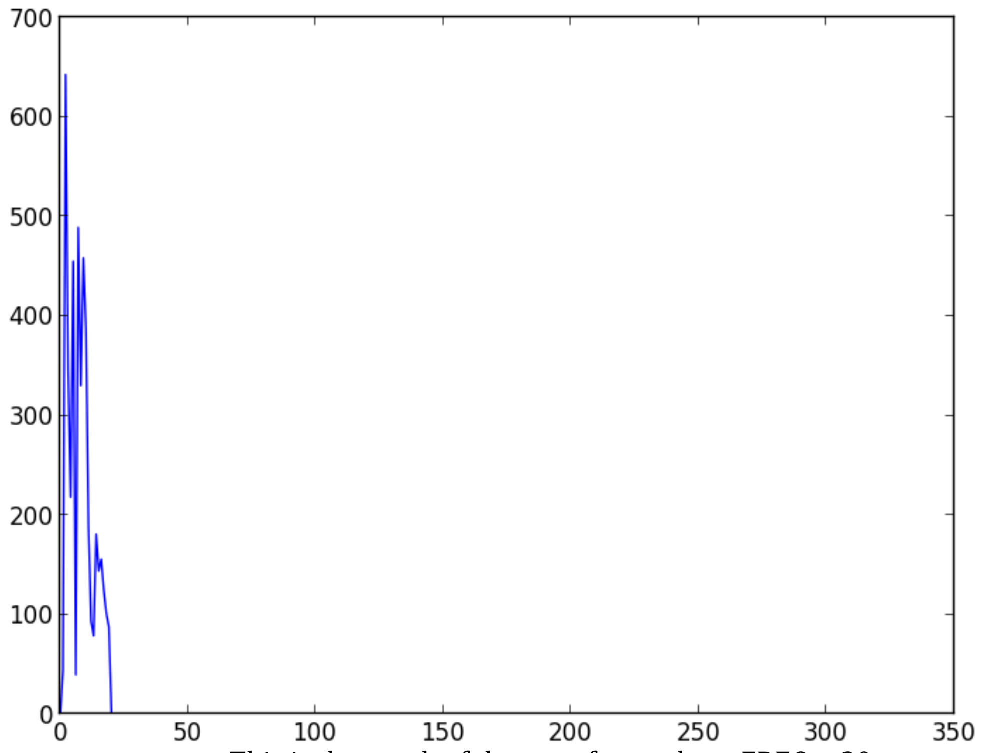
After we accomplished finding this difference, we calculated the Fourier Transform. The transform will allow us to select points so that we may clean out the 'noise' in our original data. The following are graphs of the Fourier Transform as well as the code for the Real data points. The complex parts use the same equation, but are calculated using a sine instead of a cosine.



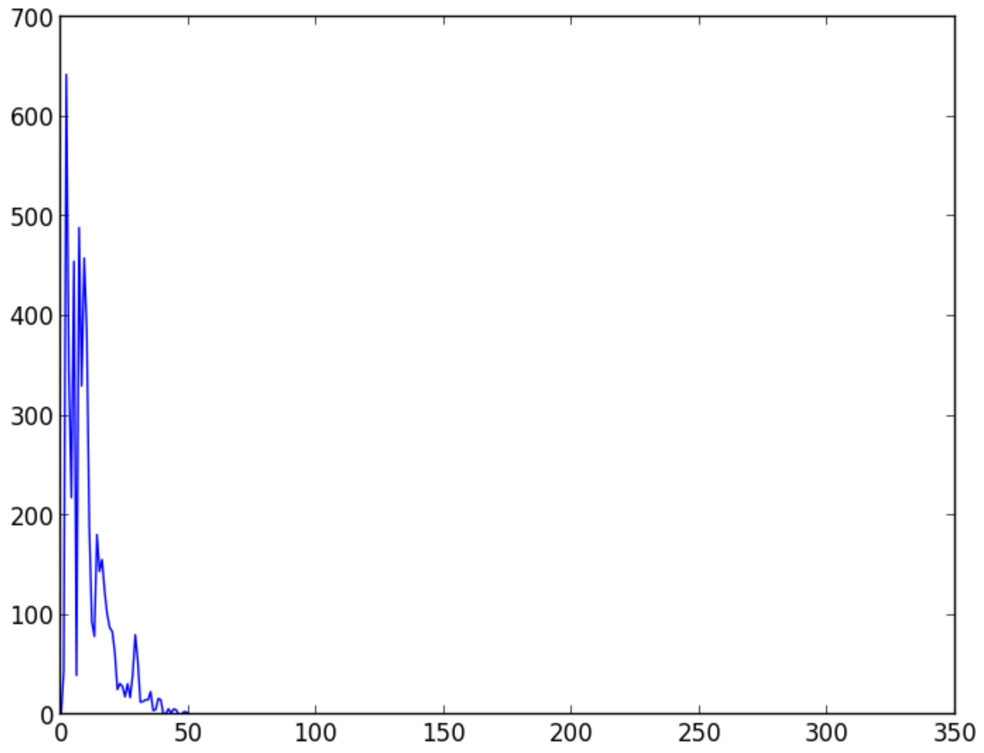
```
# Real Fourier part
ef FourierR(Array):
    # yields the real component of the Fourier Transform
    toReturn = [0.0]*fields
    for k in range(FREQ):
        for j in range(len(Array)):
            toReturn[k] = toReturn[k]+(float(Array[j]))*math.cos((2*math.pi*j*k)/field

    return toReturn
```



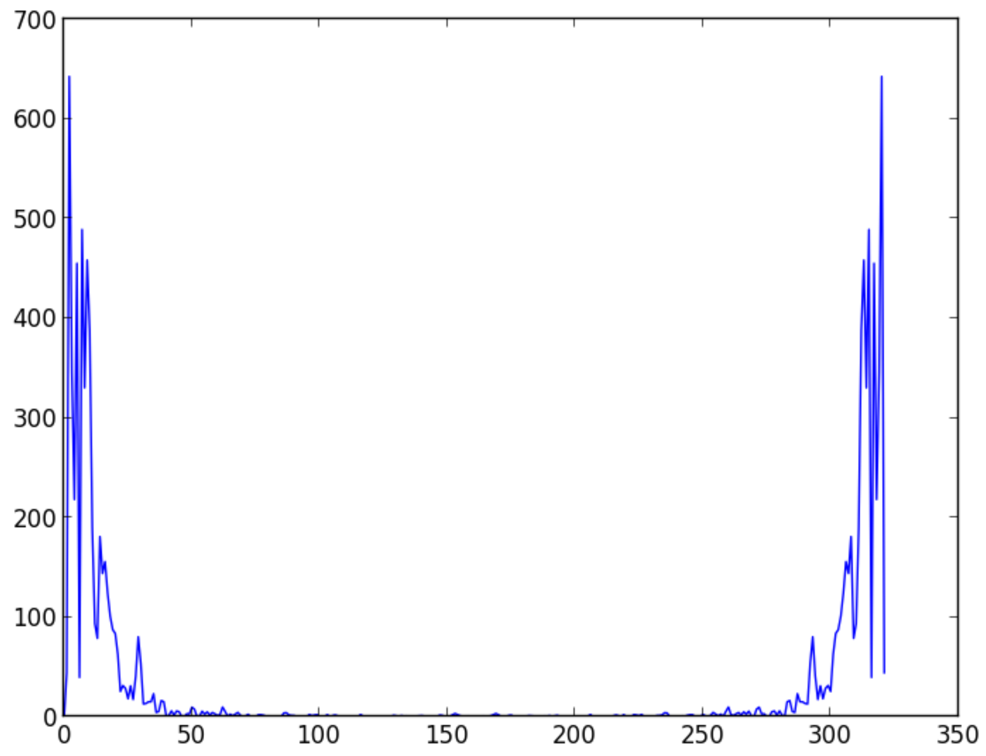
```
# vectors to hold the transform data
R_Points = FourierR(diff)
I_Points = FourierI(diff)
```


This is the graph of the transform where FREQ = 20,

Here we are using the first twenty points of the data. Frequency is on the X-axis

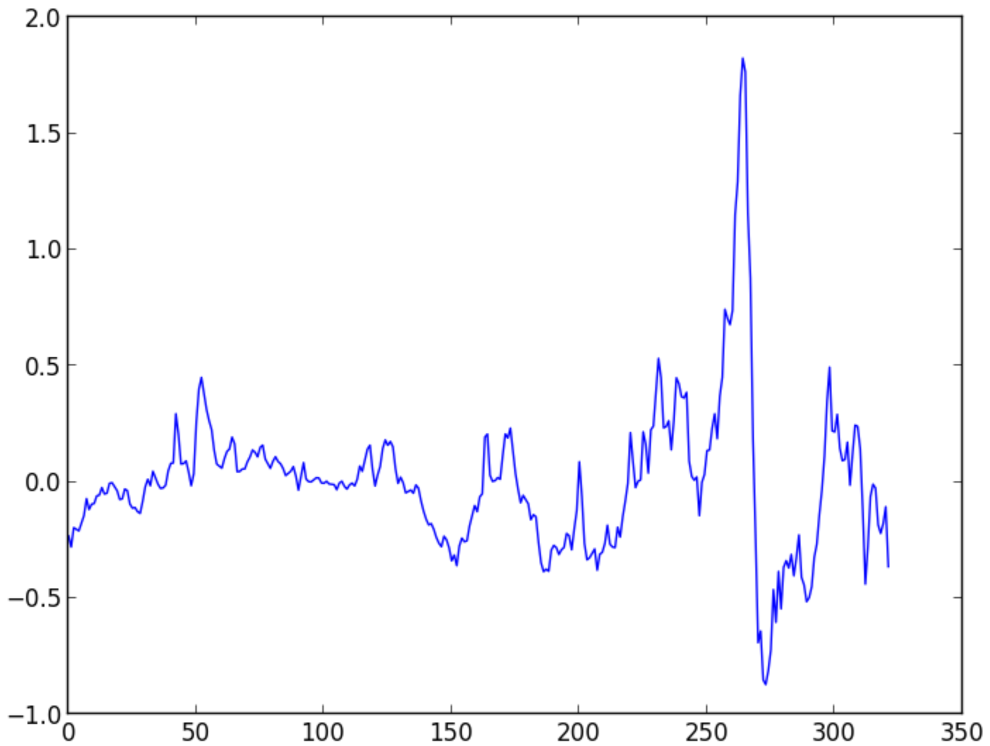


FREQ = 50

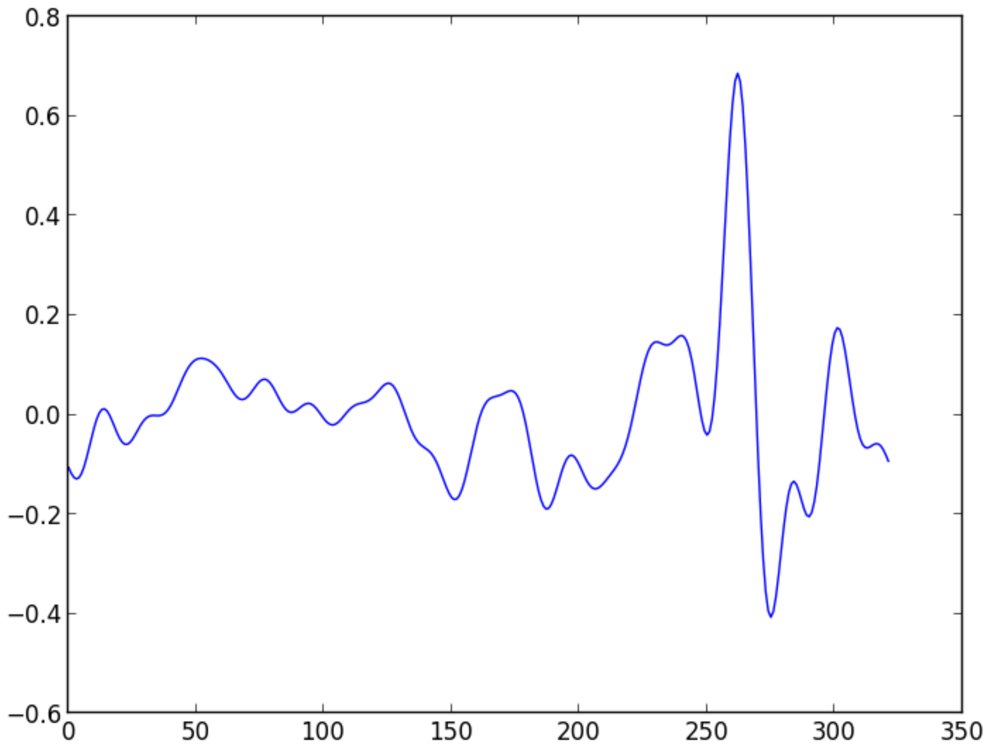


FREQ = 322 (using all of the points)

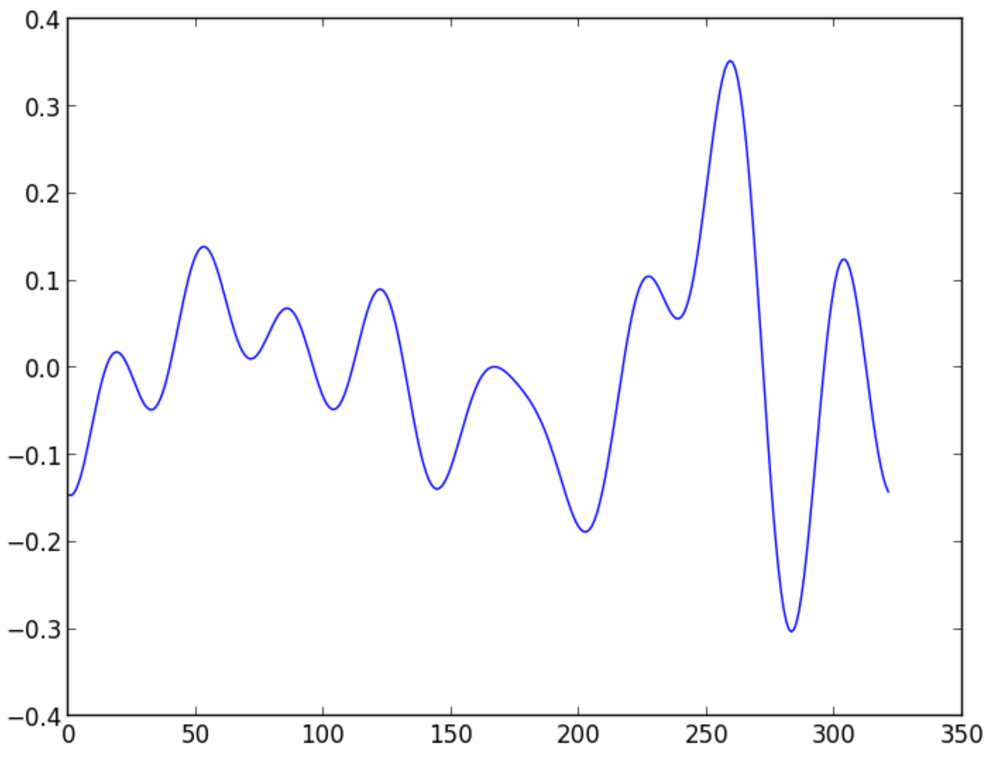
After calculating the transform, we can take the inverse Fourier Transform using only the cleaned data points, and compare this to our original data to see what cleaned version we think is the most accurate, but with the least amount of noise. The following graphs show the IFT using 322, 20, and 10 points, respectively. This is to show how our signal is getting 'cleaner.'



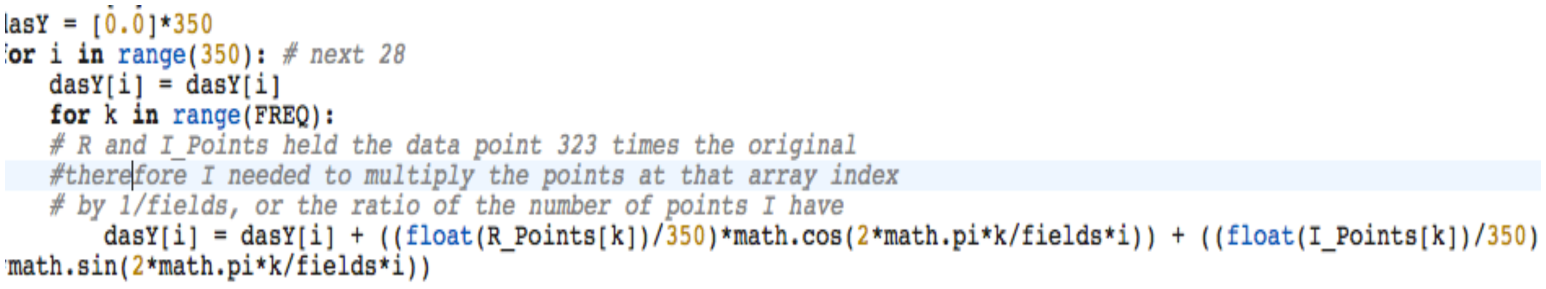
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
0
50
100
150
200
250
300
350



0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
0
50
100
150
200
250
300
350



After choosing which graph to use, we will extrapolate prices for the following 28 months. This is done using a modified version of the equation that calculates the discrete Fourier Transform. We store this output in our vector dasY.



```
lasY = [0.0]*350
or i in range(350): # next 28
    dasY[i] = dasY[i]
    for k in range(FREQ):
    # R and I_Points held the data point 323 times the original
    #therefore I needed to multiply the points at that array index
    # by 1/fields, or the ratio of the number of points I have
        dasY[i] = dasY[i] + ((float(R_Points[k])/350)*math.cos(2*math.pi*k/fields*i)) + ((float(I_Points[k])/350)
math.sin(2*math.pi*k/fields*i))
```

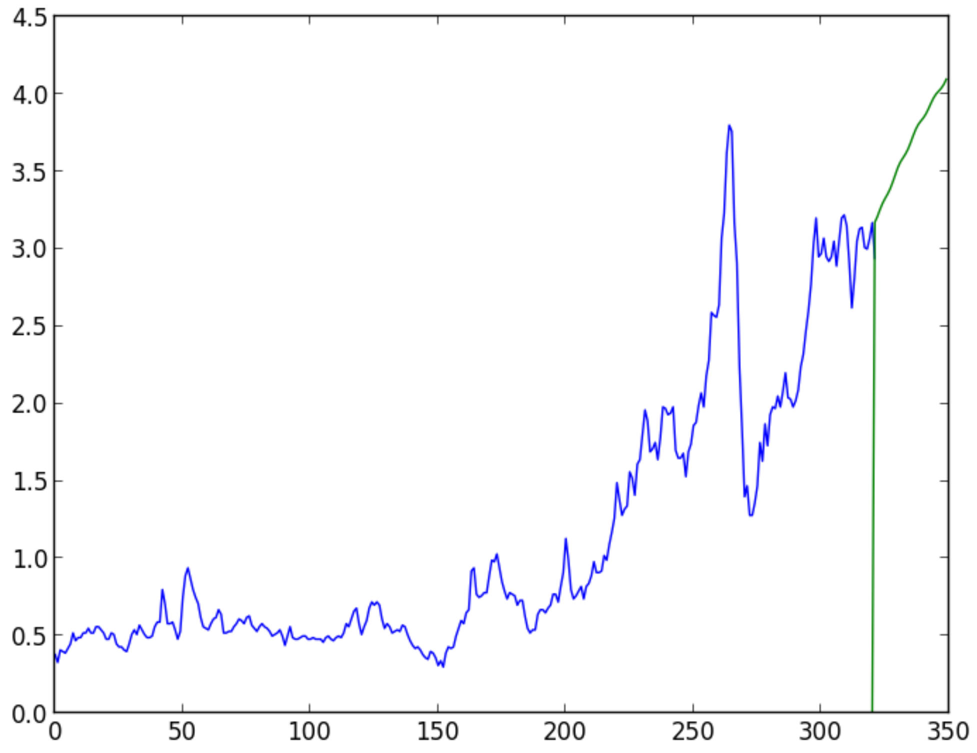
With this equation, we are able to extrapolate the IFT curve we obtained for the first 323 data points and extrapolate it for the next 28 months. Once we have the extrapolation of the IFT we add that to the best-fit curve line that we found in the beginning of our program, which is saved in the vector onemoreY.

```
# function with the polyfit coefficients
def myFunction(x):
    return 7.826*10**(-8)*float(x)**3+1.0756*10**(-5)*float(x)**2-3.1317*10**(-3)*float(x)+6.1253*10**(-1)

#oneMoreY holds the Y points of our polyfit function
oneMoreY = [0.0]*350
for i in range(350):
    oneMoreY[i] = myFunction(i)
```

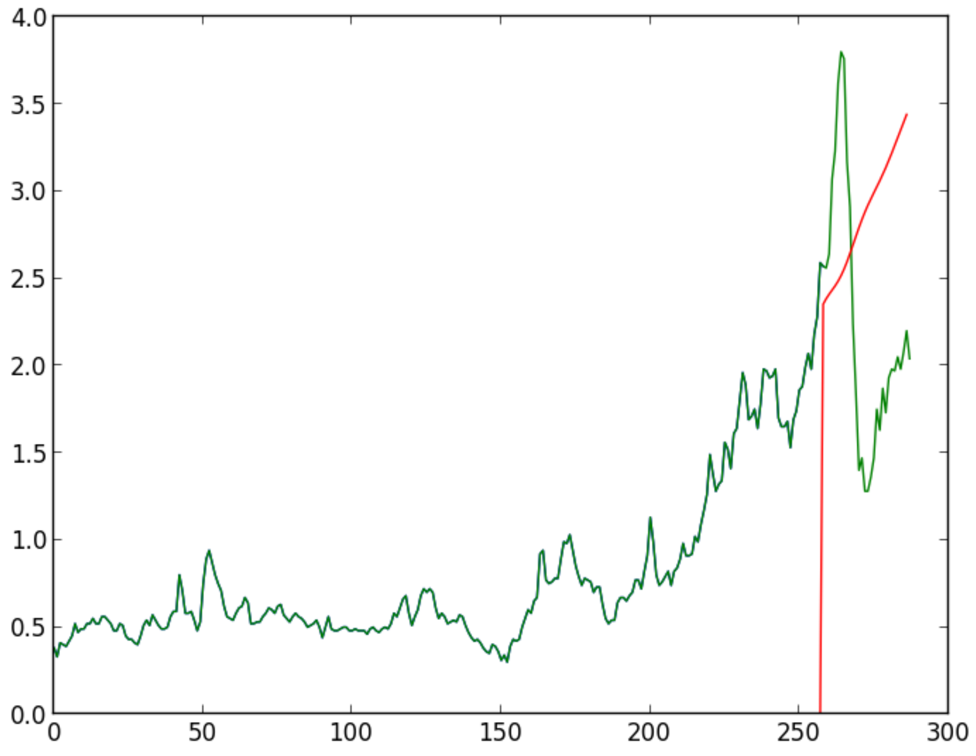
This is the code from our original best fit line.

```
### finalY is the vector that holds the two best fit components.
### This will be our prediction.
finalY = [0.0]*350
for i in range(321,350):
    finalY[i] = dasY[i] + oneMoreY[i]
```

These two components will provide us with a prediction of how the prices will behave for the following 28 months. This is the graph of our prediction using FREQ = 50 points.

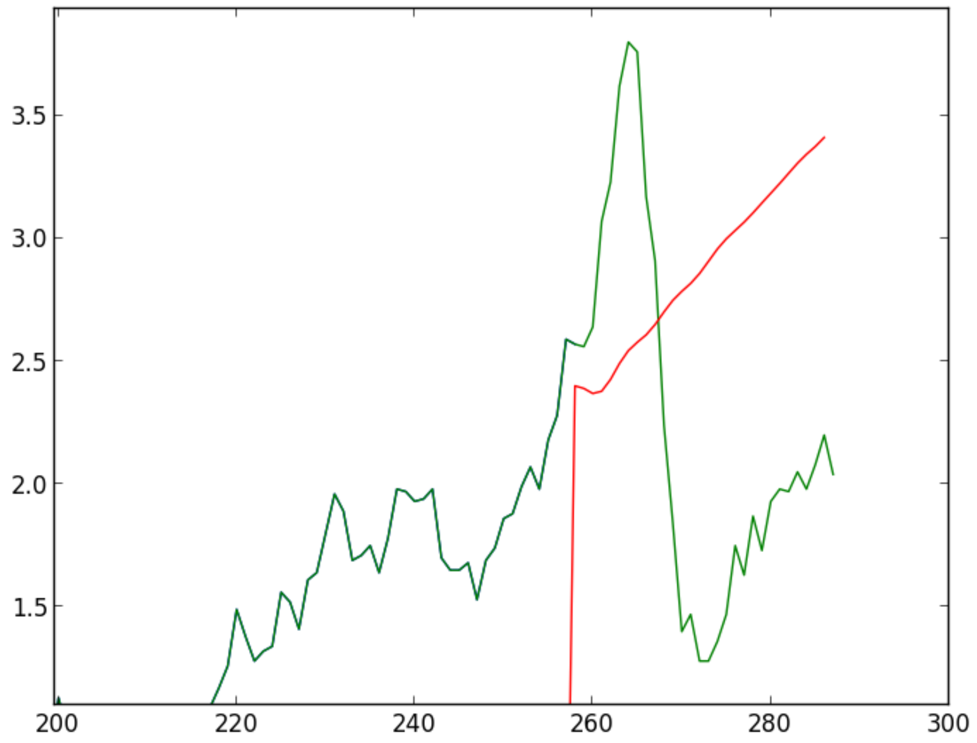


This is pretty cool, except we don't know how well it can predict this stuff because we don't have data yet. Let's modify our data to prices before 2008, and extrapolate it for the next 28 months and see what we get.

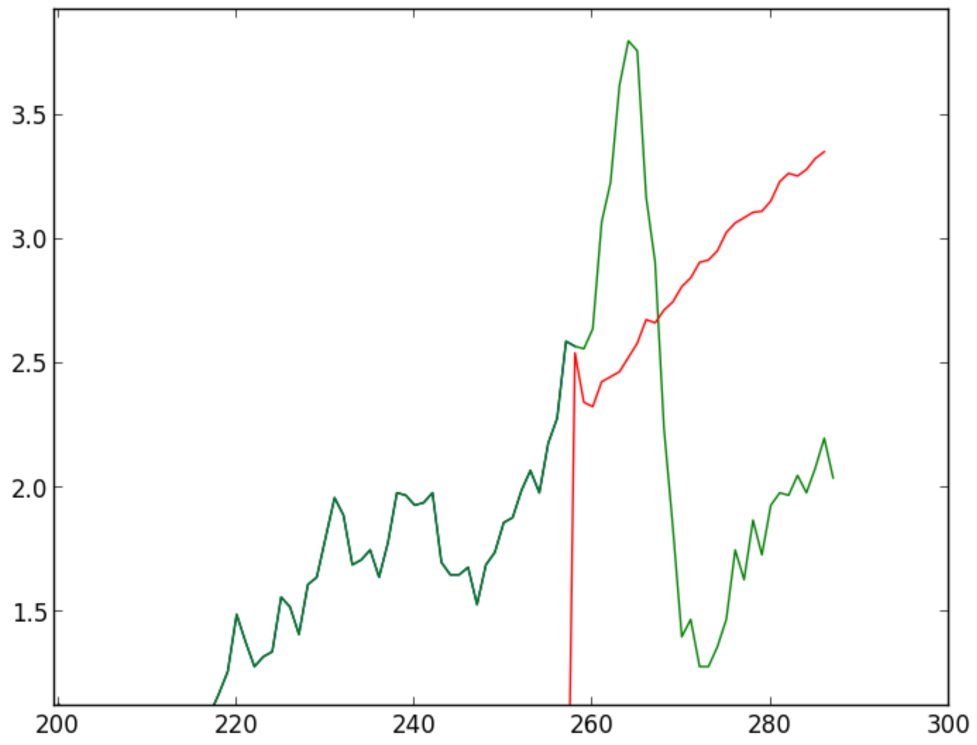


The green line is the data from 1986 to 2010, our prediction is sort of right. FREQ = 20.

This predicts fairly well whether the stock price will fall or rise, per month. However we cannot accurately predict the price, as the above graph shows. As mentioned before, oil prices are volatile. We cannot predict the price of oil accurately without looking at world events, which have a huge impact. Looking at the summer of 2008, prices skyrocketed to near 3.80 a gallon, then dropped to about one dollar, and have since jumped up to a fairly stable cycle. Taking this into account, we know that each point in our original graph tells a story. If we take out some of the points through our cleaning methods, we might miss something. The following graphs show how the price would change if we used more data points for the transform.



Close-up using FREQ = 50. Below is FREQ = all points.



As we can see, using more points doesn't necessarily mean we can more accurately predict the price. Commodities such as oil can act in a predictable but sometimes there are outside forces that can blow our predictions out of proportion. However, using solely this technique on stocks other than commodities can be even more risky, and our driving analogy becomes much more relevant.