stockyPuck Models for Zipline.io

IS609 Mathematical Modeling - CUNY

Daniel Dittenhafer & Justin Hink

December 20, 2015

Introduction

TBD - Justin

Analysis of the Problem

Buying and selling stocks is a lot like betting on a horse race. There is a lot of uncertainty. Past performance may or may not be an indicator of future success. But knowing as much as you can about the variables involved and balancing those variables appropriately can reap rewards. Everyday people trade shares based on an expectation of a share price increasing or decreasing. Sometimes automated software systems perform trades based on as much available information as an analyst could program into the algorithm.

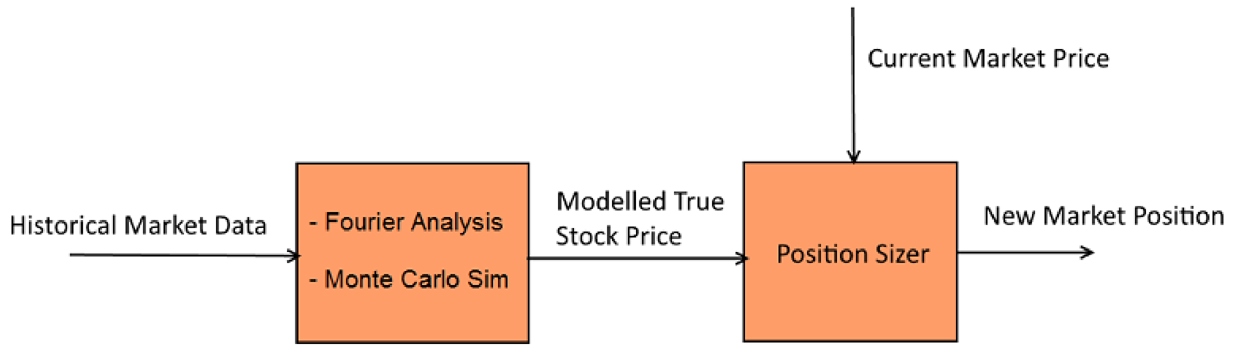
How do these analysts model the stock price in a way that they can successfully buy or sell for a profit? There are surely many ways and many variables that can and maybe should be considered, but with this project we wanted to begin understanding what is required to model a stock price using different modeling approaches.

Methodology

With this in mind, we decided to use the Zipline.io framework from the team at Quantopian, Inc to test some mathematical modeling ideas against real-world historical stock activity (Quantopian, Inc, 2015). The Zipline framework makes it very straight forward to focus on the algorithm, while the framework brings in the historical pricing for specified securities, and helps track portfolio value and position over time. There are many features of Zipline we did not use such as commissions and slippage. Rather we focused on the raw behaviour of our algorithms.

Our approach included developing (using the Python language) and testing two main models which both fed into a "position sizer" which helped us choose how much to buy or sell as a result of the predictions from the main algorithms.

- Fast Fourier Transform (FFT)
- Monte Carlo Simulation (MC)



Basically, we plugged our models into the Zipline.io framework by deriving from Zipline's `TradingAlgorithm` Python base class and implementing our own `_handle_data` functions. For more information, refer to the Zipline documenation.

Fast Fourier Transform

TBD - Justin

Monte Carlo Simulation

The Monte Carlo simulation used the daily price differences from the prior 10 days as a basis for a normal distribution. The mean and standard deviation where computed from the price difference vector and passed to the `monteCarloIteration` function for sampling over the distribution. The sampling was performed 100 times, as defined by the `mcIterations` variable. After gathering the possible price changes, we used the mean of these as tomorrow's predicted price, as shown on the last line of the code segment below.

```python
priceDiffs = histData[sym].diff()
meanDiff = priceDiffs.mean()
sdDiff = priceDiffs.std()

mcResults = list()
for i in range(0, self.mcIterations, 1):
    res = self.monteCarloIteration(meanDiff, sdDiff, curPrice)
    mcResults.append(res)
# Convert to a pandas series so we can use the statistics functions.
mcResultsPd = pd.Series(mcResults)

# What is the price we predict for tomorrow?
predictedPrice = mcResultsPd.mean()
```

The `monteCarloIteration` function is structured with the capability to predict prices n days in the future and could also be used for a crude value-at-risk computation. For our purposes, we only sample and walk 1 day into the future (`mcFutureDays` $= 1$), thereby returning tomorrow's predicted price for the given Monte Carlo iteration.

```python
def monteCarloIteration(self, mean, std, start):
    import random
    sample = list()
    for i in range(0, self.mcFutureDays, 1):
        sample.append(random.gauss(mean, std))

    curPrice = start
    walk = list()
    for d in sample:
        newPrice = curPrice + d
        curPrice = newPrice
        walk.append(curPrice)

    return walk[-1]
```

Kelly Criterion

TBD - Justin

Findings

In order to test the models, we selected a set of 10 securities, mostly stocks and the S&P 500, and measured the performance of the models based on their ending portfolio value after 4 years. Each model was executed once per security with a starting portfolio value of $100,000.00 on January 1, 2010 for a total of 20 model runs. The models had the opportunity to buy or sell shares of the security each day until January 1, 2014.

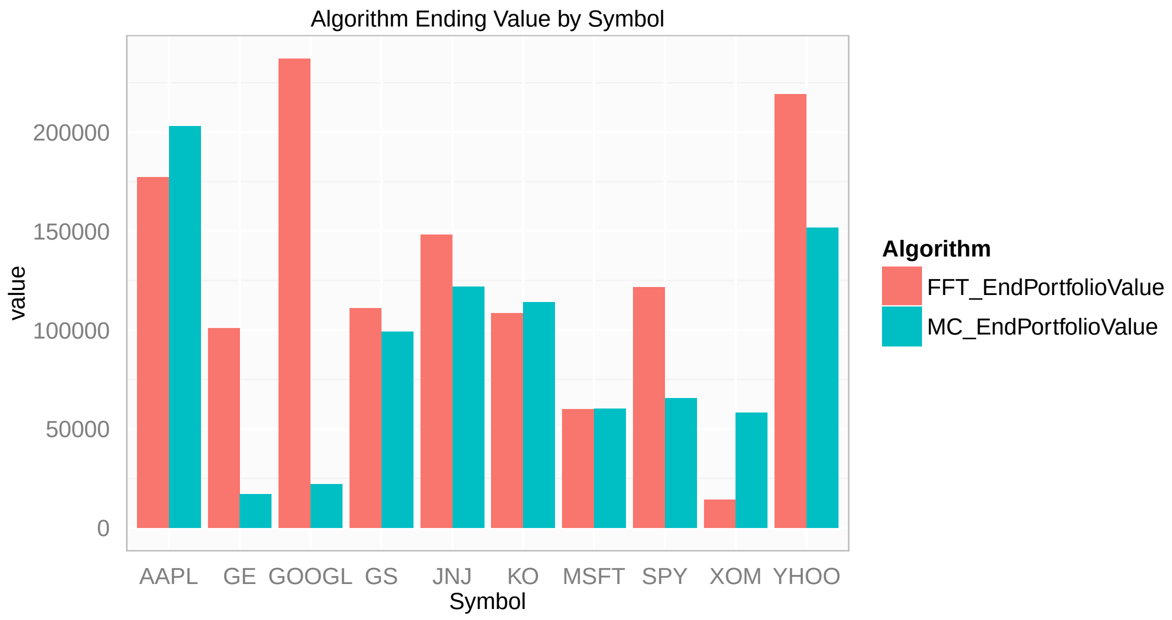
- Portfolio Starting Value: $100,000
- Back test: Jan 1, 2010 - Jan 1, 2014 (4 years)
- Variety of stocks and Index fund of S&P500

Raw Results

The results of each run are listed in the following table:

Symbol	Company	FFT_EndPortfolioValue	MC_EndPortfolioValue
AAPL	Apple Inc.	177,356.96	203,134.69
MSFT	Microsoft	60,084.11	60,276.67
SPY	(Index)	121,767.04	65,525.69
GE	General Electric	101,005.63	17,058.37
GOOGL	Google	237,267.42	22,173.25
XOM	Exxon Mobil	14,367.01	58,186.75
YHOO	Yahoo	219,306.11	151,692.87
JNJ	Johnson and Johnson	148,266.79	121,994.36
KO	Coca Cola	108,674.82	114,223.32
GS	Goldman Sachs	111,160.22	99,225.44

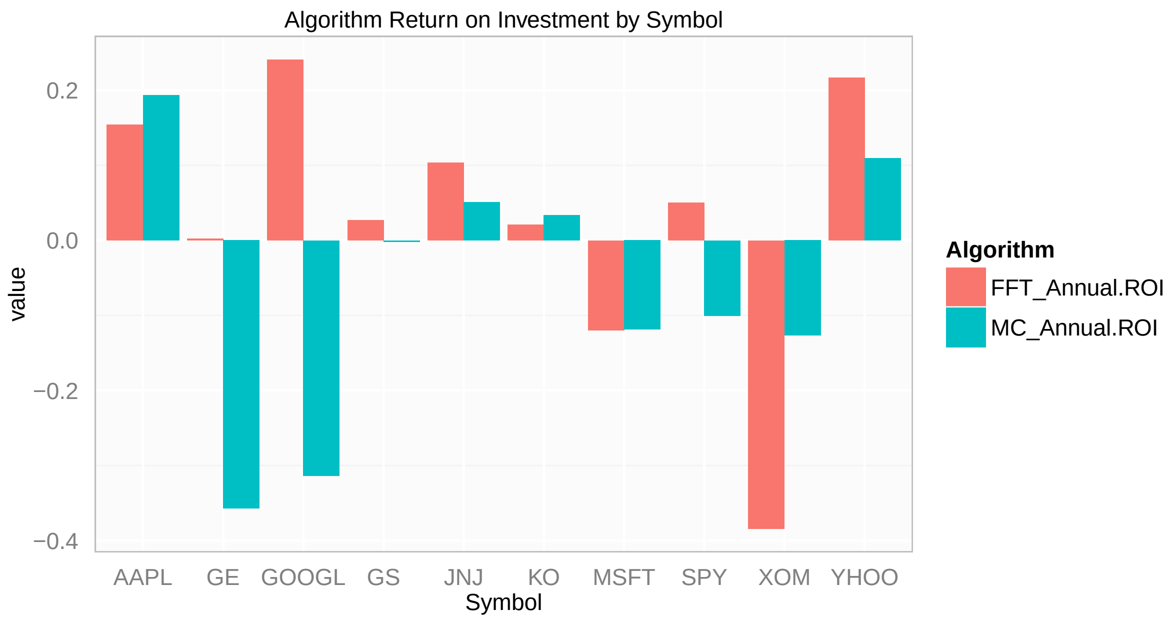
The results are a bit easier to view visually. From this view, it might seem the Fourier Transform is performing better than the Monte Carlo Simulation.



Certainly the Fourier Transform's mean and median are higher than the Monte Carlo approach, though the standard deviation is higher as well.

Algorithm	median	mean	sd	n
FFT_EndPortfolioValue	116,463.63	129,925.61	68,341.92	10
MC_EndPortfolioValue	82,375.57	91,349.14	58,403.09	10

What do the results look like when viewed from an annualized return perspective? Again, it seems the Fourier Transform is performing better than the Monte Carlo Simulation.



Algorithm	median	mean	sd
FFT_Annual.ROI	0.0386353	0.0312389	0.1812261
MC_Annual.ROI	-0.0511157	-0.0630453	0.1763825

Hypothesis Test: Fourier Transform

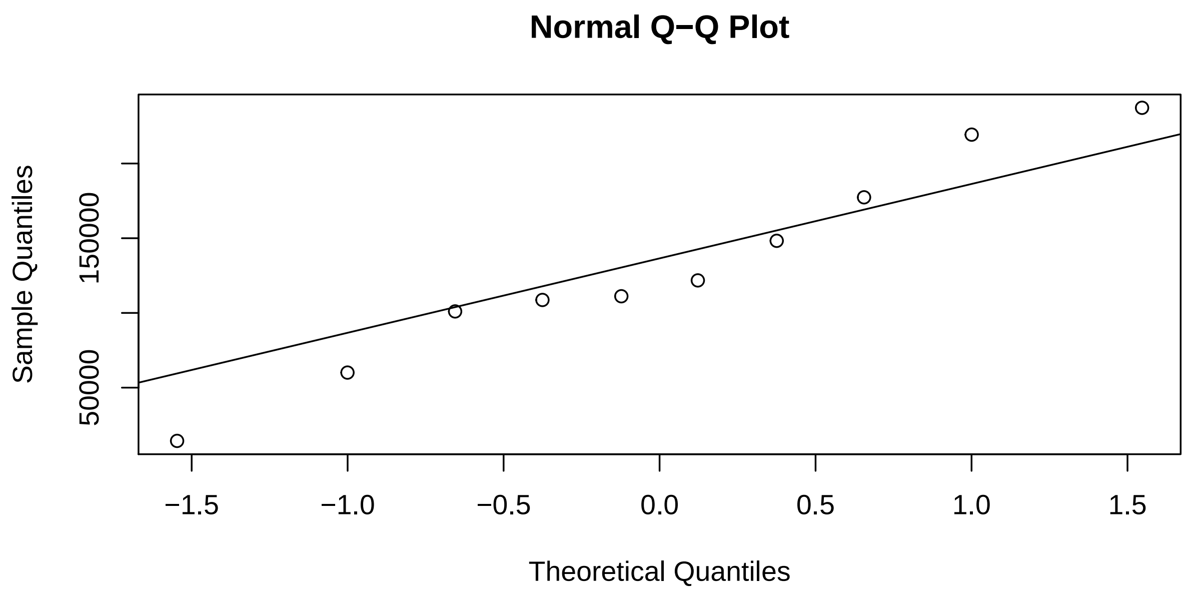
How can we conclusively test to see if our models are actually performing better than doing nothing? A statistical hypothesis test could work. First let's run a test to see if the model's ending portolio value is higher than the starting value. We'll run this upper tail test with a 0.05 alpha level, meaning a 95% confidence level.

$$H_0 : \mu_{model} = 100,000$$

$$H_a : \mu_{model} > 100,000$$

$$\alpha = 0.05$$

We'll start by applying our hypothesis test to the Fourier Transform. We need to check the distribution of the data to ensure it is a nearly normal. The closer the data is to the diagonal line, evenly spread, the better.



As shown in the Q-Q plot, above, there are deviations from the normal distribution, but for our purposes we accept this a nearly normal and can move forward with the hypothesis test. The following `R` code computes the relevant statistics which are presented below.

```r
# Grab the raw statistics out of the data frame
fftStats <- dfStats[dfStats$Algorithm == "FFT_EndPortfolioValue",]
mean <- fftStats$mean
sd <- fftStats$sd
n <- fftStats$n
df <- n - 1
```

```
# Compute t-test values and CI margin of error
se <- sd / sqrt(n)
tVal95 <- qt(0.975, df=df)
me <- tVal95 * se

# t score and p value
tScore <- (mean - 100000) / se
pVal <- pt(tScore, df, lower.tail=FALSE)
```

The standard error of the sample mean is computed as follows:

$$SE_{fft} = \frac{68,341.92}{\sqrt{10}} = 21,611.61$$

And our t-score for this hypothesis test is computed as:

$$T_{fft} = \frac{129,925.6 - 100,000}{21,611.61} = 1.384701$$

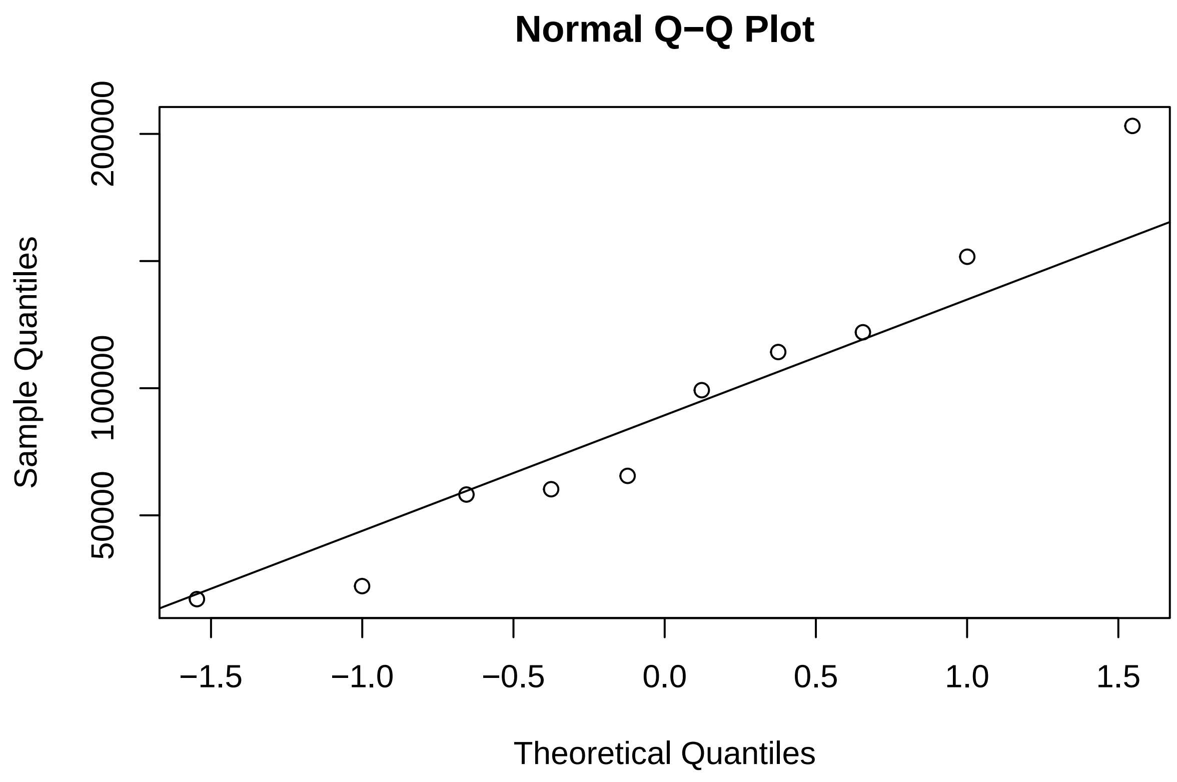
$$\text{p-value } = 0.09975275 > 0.05$$

95% Confidence Interval: $129,925.6 \pm 48,888.86 = 81,036.75$ to $178,814.5$

Based on the p-value ≈ 0.0998 and the 95%CI crossing the portfolio starting value of 100,000, we accept the null hypothesis and conclude that the Fourier Transform model, as written, is not performing significantly better.

Hypothesis Test: Monte Carlo

Next up, we will perform the same hypothesis test on the Monte Carlo Simulation. Again, we check the normality of the data and find a nearly normal distribution, as shown in the following Q-Q plot.



Once again, we use R to help us with the mathematics of the hypothesis test.

```
mcStats <- dfStats[dfStats$Algorithm == "MC_EndPortfolioValue",]
mean <- mcStats$mean
sd <- mcStats$sd
n <- mcStats$n
df <- n - 1

# Compute t-test values and CI margin of error
se <- sd / sqrt(n)
tVal95 <- qt(0.975, df=df)
me <- tVal95 * se

# t score and p value
tScore <- (mean - 100000) / se
#tScore
pVal <- pt(tScore, df, lower.tail=FALSE)
#pVal
```

$$SE_{fft} = \frac{58,403.09}{\sqrt{10}} = 18,468.68$$

$$T_{fft} = \frac{91,349.14 - 100,000}{18,468.68} = -0.4684071$$

$$\text{p-value } = 0.6746847 > 0.05$$

95% Confidence Interval: $91,349.14 \pm 41,779.05 = 49,570.09$ to $133,128.2$

Based on the p-value ≈ 0.6747 and the 95%CI crossing the portfolio starting value, we accept the null hypothesis and conclude that the Monte Carlo model as written is not performing significantly better.

Difference of Means Test

Now, let's look at how the Fourier Transform compares to the Monte Carlo approach. Are they similar, or is the Fourier Transform performing better at a statistically significant level? We setup our hypothesis test as a difference of means as shown with the following null and alternate hypotheses.

$$H_0 : \mu_{fft} = \mu_{mc}$$

$$H_a : \mu_{fft} > \mu_{mc}$$

$$\alpha = 0.05$$

We perform the calculations in `R` code as shown below, followed by the mathematical notation.

```r
# Compute standard error and difference of means
se_diff <- sqrt((fftStats$sd^2 / fftStats$n) + (mcStats$sd^2 / mcStats$n))
mean_diff <- fftStats$mean - mcStats$mean

# Compute t-score and lookup p-value
tScore_diff <- (mean_diff - 0) / se_diff
pVal <- pt(tScore_diff, df, lower.tail=FALSE)
```

$$SE_{\bar{x}_{fft}-\bar{x}_{mc}} = \sqrt{\frac{68,341.92^2}{10} + \frac{58,403.09^2}{10}} = 28,428.05$$

$$T_{diff} = \frac{38,576.47 - 0}{28,428.05} = 1.356986$$

$$\text{p-value } = 0.1039196 > 0.05$$

Based on the p-value ≈ 0.1039 we do not reject the null hypothesis and therefore conclude that the Fast Fourier Transform is not significantly better than the Monte Carlo Simulation.

Conclusions

After developing and testing 2 main models combined with a position sizing Kelly Criterion, we found our models were not much better than keeping our money under our pillows. And surprisingly, neither model was significantly better than the other. Possibly with a broader test set (i.e. more stocks) we could better discern a difference between the methods. With that said, we believe the models are a good place to start for further tuning and improvement.

Mathematical models are notoriously poor at predicting outside their original data set range, and predicting the future (an unknown) is implicitly outside the data set range. The approach we took was generic for any stock or mutual fund, but developing more specific approaches, based on industry or security type, might yield improved results.

The original code, test results, paper and presentation are available in the stockyPuck repository on Github (Dittenhafer and Hink, 2015).

Referernces

Dittenhafer, D. and J. Hink. stockyPuck. 2015. URL: https://github.com/dwdii/stockyPuck.

Quantopian, Inc. Zipline, a Pythonic Algorithmic Trading Library. 2015. URL: https://github.com/quantopian/zipline.