FNGU Pairs Trading Strategy

Yu-Wei Vincent Yeh
University of California, Los Angeles
vincentywyeh@gmail.com

Abstract

The aim of this project is to discover profitable opportunities that stem from price discrepancies between FNGU and its holdings. Discrepancies were first discovered and analyzed using Excel spreadsheet, and these discoveries were later implemented in Python using QuantConnect's LEAN Engine, an open-source algorithmic trading software. The algorithm was created using Pairs Trading Based on Cointegration framework provided by QuantConnect.

Motivation

FNGU is a 3X leveraged ETF established in 2018 that holds ten major technology stocks (arranged by %Assets) including: TWTR (12.88%), TSLA (12.17%), AAPL (9.99%), FB (9.82%), GOOGL (9.74%), BABA (9.55%), NFLX (9.18%), AMZN (9.15%), BIDU (8.97%) and NVDA (8.55%). Since fluctuations in FNGU's underlying holdings would be reflected three times as much in its percent change, in order to amplify the swing it has to constantly adjust different factors such as debt ratio and holdings of other financial derivatives. However, in this ever-changing financial market, it is impossible to make flawless adjustments, thus leaving discrepancies that could be exploited.

I. Devising Trading Strategy Using Excel

A. Observations and Data Visualization

In order to establish a better understanding of the strategy, this subsection is devoted primarily to the visualization of data and providing raw observations.

To begin with, since FNGU reflects three times the change of its underlying holdings each day, to observe the discrepancies, the proper way would be to first create a weighted percent change of its holdings then scale up by three times. Thus, we'd like to multiply each of the ten stock's percent change each day with the percentage it stands in FNGU's holdings and add all ten results up to generate a weighted change. For instance, suppose ten stocks all rise by 1.5% in a given day, then the weighted percent change would be the sum of 1.5% multiplied by the weights of 12.88%, 12.17%, 9.99%, etc. Thereafter, scaling up the sum obtained in the previous step by three times would yield the percentage that FNGU is expected to change. Graphing the difference between scaled-up price movements of FNGU's underlying holdings and its percent change would then yield Figure 1.

As shown in Figure 1 and 2, FNGU is never perfectly leveraged with discrepancies fluctuating around 0%. Thus, the goal is to devise a strategy that could benefit from the more extreme discrepancies such as the values highlighted red in Figure 2.

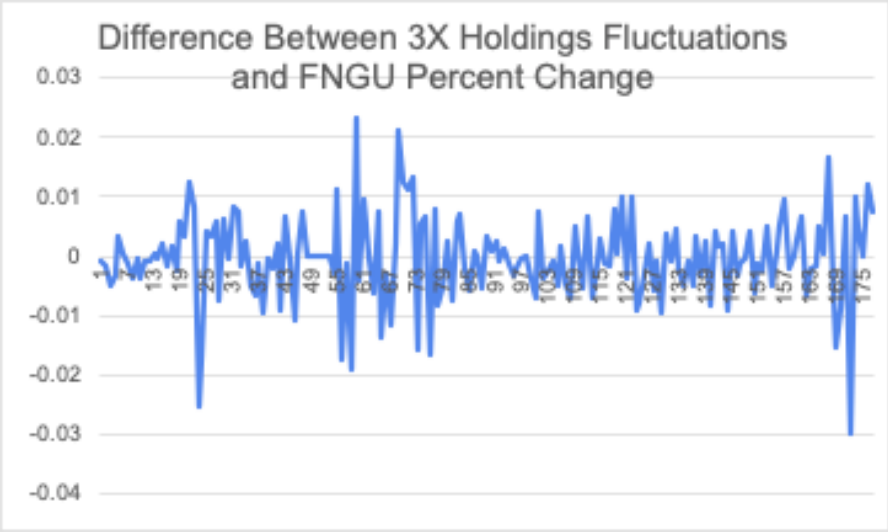

Figure 1. Percent Change Discrepancy (Market Crash Data Excluded for Clearer Presentation)

FNGU	Weighted Change	Difference (3X)	Standardized Normal	
-0.028983285	-0.009422939	-0.000714468	-0.045548766	
0.040429564	0.014008117	-0.001594788	-0.166053719	
0.021250759	0.00876836	-0.005054321	-0.639620844	
0.038941736	0.014079606	-0.003297081	-0.399076545	
0.023748212	0.006752186	0.003491654	0.530217071	
-0.001257686	-0.00054768	0.000385353	0.105003021	
0.068420316	0.023269523	-0.001388254	-0.137781784	
-0.020429544	-0.00560647	-0.003610134	-0.441929645	
-0.011096257	-0.003593194	-0.000316676	0.008903979	
0.014870894	0.006229801	-0.00381851	-0.470453713	
0.010923138	0.003964671	-0.000970875	-0.080647723	
0.011200422	0.003971054	-0.000712741	-0.045312468	
0.002866823	0.000753542	0.000606197	0.13523387	
0.016242204	0.005525714	-0.000334937	0.006404228	
-0.036951796	-0.012965182	0.00194375	0.318328225	
-0.073685608	-0.023980137	-0.001745197	-0.186642896	
0.056184607	0.018104187	0.001872045	0.308512664	
0.018998507	0.007214983	-0.002646442	-0.31001208	
0.019043814	0.004455493	0.005677335	0.829409704	
-0.027313121	-0.010248679	0.003432917	0.522176782	
0.144296655	0.043966968	0.012395752	1.749077615	⇐ line 1
0.116590349	0.0360634	0.00840015	1.202129287	⇐ line 2
-0.093375394	-0.022705222	-0.025259728	-3.405490441	⇐ line 3
0.067849687	0.028337348	-0.017162356	-2.297060687	⇐ line 4
-0.021939828	-0.008648188	0.004004737	0.600451804	

Figure 2. A Snapshot of the Excel Spreadsheet

B. Trading Strategy Proposal

a. Pairs Trading Strategy Summary:

Being one of the most influential statistical arbitrage strategies in history, Pairs Trading didn't become well-known to the world until the Internet became prevalent. This elegant strategy was constructed upon the premise that two historically correlated diverging assets would eventually converge back to each other. The general strategy is to short the assets with an overly inflated price and long the assets that underperform suppose that a certain threshold is met.

Similar to, yet quite different from conventional Pairs Trading, the strategy presented in part b will be trading two nearly identical assets (FNGU and its holdings) at a high frequency with the extreme values in discrepancies being the focus.

b. FNGU Trading Strategy:

Notice in Figure 2 that whenever FNGU outperforms or underperforms its holdings, it would eventually do the reverse within a few days. Thus, using this property, a statistical arbitrage strategy similar to Pairs Trading can be constructed when FNGU either significantly outperforms or underperforms its holdings. As shown in Figure 2, since FNGU would return to the "near-normal" level mostly within two days, the length that we hold onto a position shouldn't exceed two days. Furthermore, since trading decisions are based solely on the discrepancies in percentage without regarding the actual price change, holding onto a position

for even longer than two days could easily jeopardize the gains. Therefore, a short-term trading protocol shown as follows would be desirable:

1. Whenever FNGU's percent change is significantly larger than three times the weighted percent change of its holdings, we would short FNGU and long its holdings. Due to FNGU's 3X leverage, for every dollar used to short FNGU, we have to buy three dollars worth of the ten stocks based on holdings percentage to achieve the same effect magnitude.
2. Whenever FNGU's percent change is significantly lower than three times the weighted percent change of its holdings, we would long FNGU and short its holdings. Due to FNGU's 3X leverage, for every dollar used to long FNGU, we would also have to buy three dollars worth of the ten stocks based on holdings percentage to achieve the same effect magnitude.
3. If FNGU and its holdings' percent change diverge in the same direction and stay in extreme divergence for more than one day, we would liquidate the position due to risks in large price fluctuations.

For instance, on line 1 indicated next to Figure 2, we would short FNGU and long its holdings, and on line 3 we would perform the reverse. Both transactions could be liquidated the next day (line 2 and line 4) depending on the value of the threshold that defines extreme discrepancies.

II. Implementation of Strategy Through LEAN Engine

A. Algorithm and Performance Display:

```
from sklearn import linear_model
import numpy as np
import pandas as pd
from scipy import stats
from math import floor
from datetime import timedelta

class PairsTradingAlgorithm(QCAlgorithm):
    def Initialize(self):
        self.SetStartDate(2018,1,22)
        self.SetEndDate(2020,9,18)
        self.SetCash(10000000) #set large amount of cash to avoid rounding error when adjusting the weight of each stock
        self.numdays = 250  # set the length of training period
        tickers = ["FNGU", "TWTR", "TSLA", "AAPL", "FB", "GOOGL", "BABA", "NFLX", "AMZN", "BIDU", "NVDA"] #underlying stocks of FNGU
        self.holdingsWeight = [1.0, 0.1288, 0.1217, 0.099, 0.0982, 0.0974, 0.0955, 0.0918, 0.0915, 0.0897, 0.0855] #FNGU stock holdings percentage
        self.symbols = []
        self.threshold = 1. #threshold for which the algorithm executes buy/sell
        for i in tickers:
            self.symbols.append(self.AddSecurity(SecurityType.Equity, i, Resolution.Daily).Symbol) #add the securities to our list
        for i in self.symbols:
            i.hist_window = RollingWindow[TradeBar](self.numdays)

        #Set the transaction fee of each securities to 0
        self.Securities["FNGU"].FeeModel = ConstantFeeModel(0)
        self.Securities["TWTR"].FeeModel = ConstantFeeModel(0)
        self.Securities["TSLA"].FeeModel = ConstantFeeModel(0)
        self.Securities["AAPL"].FeeModel = ConstantFeeModel(0)
        self.Securities["FB"].FeeModel = ConstantFeeModel(0)
        self.Securities["GOOGL"].FeeModel = ConstantFeeModel(0)
        self.Securities["BABA"].FeeModel = ConstantFeeModel(0)
        self.Securities["NFLX"].FeeModel = ConstantFeeModel(0)
        self.Securities["AMZN"].FeeModel = ConstantFeeModel(0)
        self.Securities["BIDU"].FeeModel = ConstantFeeModel(0)
        self.Securities["NVDA"].FeeModel = ConstantFeeModel(0)

    #calculates the percent change of stock price
    def PercentChange(self, pointOne, pointTwo, someArray):
        return float((float(someArray[pointTwo].Close)-float(someArray[pointOne].Close))/float(someArray[pointOne].Close))

    def OnData(self, data):
        #check that we have stock data
        if not (data.ContainsKey("FNGU") and data.ContainsKey("TWTR") and data.ContainsKey("TSLA") and data.ContainsKey("AAPL") and data.ContainsKey("FB") and
            data.ContainsKey("GOOGL") and data.ContainsKey("BABA") and data.ContainsKey("NFLX") and data.ContainsKey("AMZN") and data.ContainsKey("BIDU") and
            data.ContainsKey("NVDA")): return
        #create rolling windows of data
        for symbol in self.symbols:
            symbol.hist_window.Add(data[symbol])
        #price_x denotes the percent change of FNGU each day
        price_x = pd.Series([self.PercentChange(i, i+1, (self.symbols[0].hist_window)) for i in range(len(list(self.symbols[0].hist_window))-1)],
                            index = [self.symbols[0].hist_window[i].Time for i in range(1, len(list(self.symbols[0].hist_window)))])
        #combine the percent change arrays into one based on holdings weight
        combinedPercent = []
        for i in range(1, len(list(self.symbols[0].hist_window))):
            eachSymbol = 0.00
            for j in range(10):
                eachSymbol += self.holdingsWeight[j+1]*self.PercentChange(i-1, i, (self.symbols[j+1].hist_window))
            combinedPercent.append(eachSymbol)
```

```
        #series created with the weighted change with time index
        price_y = pd.Series([i for i in combinedPercent],
                            index = [self.symbols[0].hist_window[i].Time for i in range(1, len(list(self.symbols[0].hist_window)))])
        #wait until we get enough data
        if len(price_x) < 249: return
        #Calculate the spread, mean and standard deviation
        spread = self.regr(price_x,3*price_y)
        mean = np.mean(spread)
        std = np.std(spread)

        #note that for every $1 of FNGU traded we need to trade $3 of other stocks based on their holdings percentage to achieve the same level of price change
        if spread[-1] > mean + self.threshold * std:
            if not self.Portfolio[self.symbols[0]].Quantity > 0 and not self.Portfolio[self.symbols[0]].Quantity < 0:
            #trade each stock base on the holdings percentage
                self.Sell(self.symbols[1], 6000000*self.holdingsWeight[1]/self.Portfolio[self.symbols[1]].Price)
                self.Sell(self.symbols[2], 6000000*self.holdingsWeight[2]/self.Portfolio[self.symbols[2]].Price)
                self.Sell(self.symbols[3], 6000000*self.holdingsWeight[3]/self.Portfolio[self.symbols[3]].Price)
                self.Sell(self.symbols[4], 6000000*self.holdingsWeight[4]/self.Portfolio[self.symbols[4]].Price)
                self.Sell(self.symbols[5], 6000000*self.holdingsWeight[5]/self.Portfolio[self.symbols[5]].Price)
                self.Sell(self.symbols[6], 6000000*self.holdingsWeight[6]/self.Portfolio[self.symbols[6]].Price)
                self.Sell(self.symbols[7], 6000000*self.holdingsWeight[7]/self.Portfolio[self.symbols[7]].Price)
                self.Sell(self.symbols[8], 6000000*self.holdingsWeight[8]/self.Portfolio[self.symbols[8]].Price)
                self.Sell(self.symbols[9], 6000000*self.holdingsWeight[9]/self.Portfolio[self.symbols[9]].Price)
                self.Sell(self.symbols[10],6000000*self.holdingsWeight[10]/self.Portfolio[self.symbols[10]].Price)
                self.Buy(self.symbols[0],  2000000/self.Portfolio[self.symbols[0]].Price)
            else:
                #if the holdings maintain the same but the extreme conditions remain (above threshold), liquidate the portfolio (cut loss)
                self.Liquidate()
        elif spread[-1] < mean - self.threshold * std:
            if not self.Portfolio[self.symbols[0]].Quantity < 0 and not self.Portfolio[self.symbols[0]].Quantity > 0:
            #trade each stock base on the holdings percentage
                self.Buy(self.symbols[1], 6000000*self.holdingsWeight[1]/self.Portfolio[self.symbols[1]].Price)
                self.Buy(self.symbols[2], 6000000*self.holdingsWeight[2]/self.Portfolio[self.symbols[2]].Price)
                self.Buy(self.symbols[3], 6000000*self.holdingsWeight[3]/self.Portfolio[self.symbols[3]].Price)
                self.Buy(self.symbols[4], 6000000*self.holdingsWeight[4]/self.Portfolio[self.symbols[4]].Price)
                self.Buy(self.symbols[5], 6000000*self.holdingsWeight[5]/self.Portfolio[self.symbols[5]].Price)
                self.Buy(self.symbols[6], 6000000*self.holdingsWeight[6]/self.Portfolio[self.symbols[6]].Price)
                self.Buy(self.symbols[7], 6000000*self.holdingsWeight[7]/self.Portfolio[self.symbols[7]].Price)
                self.Buy(self.symbols[8], 6000000*self.holdingsWeight[8]/self.Portfolio[self.symbols[8]].Price)
                self.Buy(self.symbols[9], 6000000*self.holdingsWeight[9]/self.Portfolio[self.symbols[9]].Price)
                self.Buy(self.symbols[10],6000000*self.holdingsWeight[10]/self.Portfolio[self.symbols[10]].Price)
                self.Sell(self.symbols[0],2000000/self.Portfolio[self.symbols[0]].Price)
            else:
                #if the holdings maintain the same but the extreme conditions remain (above threshold), liquidate the portfolio (cut loss)
                self.Liquidate()
        #if the spread just went from extreme value towards the other direction, liquidate the portfolio
        elif (spread[-1] > mean and self.Portfolio[self.symbols[0]].Quantity<0) or (spread[-1] < mean and self.Portfolio[self.symbols[0]].Quantity>0):
            self.Liquidate()
    def regr(self,x,y):
        regr = linear_model.LinearRegression()
        x_constant = np.column_stack([np.ones(len(x)), x])
        regr.fit(x_constant, y)
        beta = regr.coef_[0]
        alpha = regr.intercept_
        spread = y - x*beta - alpha
        return spread
```

Figure 3. Trading algorithm created through the LEAN Engine on QuantConnect Algorithm Lab

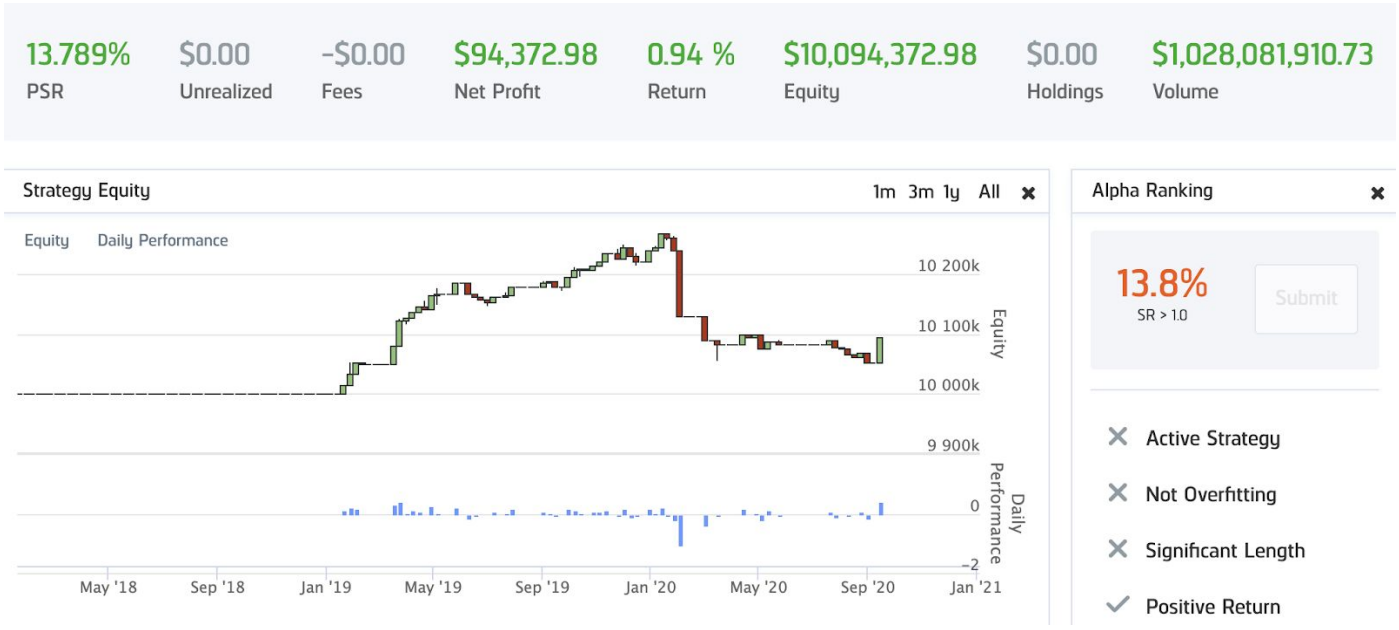

Figure 4. Screenshot of the strategy's performance starting from approximately Jan, 2019 through 09/14/20

Figure 5. Screenshot of the strategy's performance starting from approximately Jan, 2019 through 01/20/20

B. How the Algorithm Works:

a. Algorithm (Figure 2):

Some minor properties of the program include Initialize(), a constructor that sets the parameters such as trading time frame, monetary amount, securities considered in the model, and the rolling window that would be used in OnData() to assist in trading, and regr(), a helper function that uses regression to calculate spreads of the discrepancies. Also note that this algorithm is set to be trading on a zero-commission model. Since OnData() contains the main trading strategy, this is what we will be focusing on.

After storing 249 days of FNGU percent change data (250 days is needed to obtain 249 percent change data) into a series (denoted by price_x), OnData() then performs the weighted percent change calculations on the ten stocks in FNGU's holdings over the same time frame and stores the value into another series (denoted by price_y). With both sets of data, we're able to perform regressions and calculate the mean and standard deviations of the spread. Using exactly the same protocol as the proposed strategy from part I, whenever FNGU outperforms its holdings, we take a short position and long its holdings by three times the monetary amount, and vice versa. The monetary amount to trade for each stock is calculated using FNGU's percent holdings (denoted by holdingsWeight[]) multiplied by the total monetary amount allocated for all ten stocks. Since the regression is set to train on 249 days of data, and FNGU was only recently created on 01/22/18, January of 2019 is when this program actually starts to trade.

b. Performance (Figure 3 and 4):

Starting from January of 2019 through late January of 2020, we have stable and consistent gains with only a few minor dips (Figure 4). However, as mentioned in part I, since only discrepancies in percent change are incorporated into the model, most of the gains are eliminated during times of extreme volatility in February and March. Thereon, because the data used for performing the regression are significantly skewed by the market crash, growth after March is unimpressive.

Until January, the return is mediocre but has steadily increased over the past year, reaching 2.7% (Figure 5) — note that since only 80% of the capital in the portfolio is utilized to perform transactions at any time, the actual return would be closer to 3.5%. After the market crash, the return drops to a mere 1% (Figure 4).

C. Weaknesses of the Strategy and Algorithm:

a. Transaction Rounding Errors:

As noted in strategy description, the number of stocks to be traded should ideally follow their respective weight in FNGU's holdings, and combined they should also achieve the perfect monetary ratio with respect to the amount utilized to trade FNGU. However, since LEAN Engine only allows stock to be traded in whole numbers, there would be rounding errors that could cause the proportions to be slightly off. In order to minimize the effect of rounding errors, the total monetary amount in the portfolio is set to be a large number (ten million dollars).

b. FNGU Establish Date:

It has not even been 3 years since FNGU began trading, so the database that could be utilized for training and backtesting is quite small — if this program is designed to perform intraday trades, even having one year of data would be more than enough. However, this is not the case. Furthermore, to achieve accuracy in regression calculation, the number of days used for calculation has to be large. Thus, using 250 days of data would also shorten the time frame for actual trading.

c. Profitability

Most of the pricing discrepancies between FNGU and its holdings are insignificant that even the more extreme values with several standard deviations away from the mean could only result in extremely small gains. Therefore, not only do the positions can't be held for a longer time frame, significant market movements could be catastrophic and easily incur huge losses (compared to the gains).

III. Conclusion

Although the return of this algorithm is unimpressive, I believe the value of this statistical arbitrage strategy lies within its ability to ensure stability and guarantee highly consistent yields. This strategy does lack practicality in day-to-day trading, but it's definitely interesting to see that gaps in pricing discrepancy could lead to some profit. For future studies, I would like to implement alternative pairs trading strategies such as trading only one of the ten underlying stocks that makes the biggest move during the particular day then FNGU significantly diverges from its underlying holdings.