Shreyas Vivekanandan

Dr. Dickerson

ECE1895

December 16th, 2022

Backtested MACD/Fibonacci Trading Algorithm

Design Overview

For my final design project, I proposed creating a Python trading algorithm and backtest it to take my interest in finance and coding a little bit further as I have never done a project like this before. The purpose of this design project is to create a profitable trading algorithm using two indicators used in the industry today, Moving Averages Convergence/Divergence(MACD) and Fibonacci Levels, and backtesting this strategy against historical data of a given stock. When deciding what indicators to use, I thought of using a common indicator as well as an indicator that is not very common to create a unique algorithm. From there, I decided to use MACD as it already integrates a powerful indicator, Exponential Moving Average(EMA), within its calculations as well as one of my personal favorite indicators, Fibonacci Extension/Retracements. My project was split into two languages/parts: Python using the Alpaca API for easy data extrapolation and creation of the strategy and PineScript to implement and backtest my strategy in TradingView to generate a report on the strategy..

Preliminary Design Verification

To start creating this algorithm, I wanted to use Python to gather data and plot an easy graph based on my strategy. I researched a lot of IDEs(VSCode, JetBrains, Anaconda) to use to create it and ended up using Jupyter Notebooks hosted in Google Colab. The reason for using

Google Collab was that it was easy to use and I can create and execute parts of code instead of the whole script itself while keeping clean documentation. Since I used Google Colab, I was able to omit hosting in AWS as it is already a cloud-hosted application. Next, I needed a way to extrapolate market data and ended up choosing the Alpaca API as it was a very common trading API that I came across and was much better than loading a CSV file or web scraping data from Yahoo finance. The API allowed me to get data about a certain stock based on some default and user inputs such as start date, end date, and timeframe. After choosing the Alpaca API, I had to research some libraries I had to use in order to manipulate the new data I had just gotten. The three main libraries I ended up needing to use were Matplotlib for graphing and Pandas and NumPy for data manipulation. The first step in creating this algorithm was to use my Alpaca personal ID and secret key to create a web socket connection to my account for market data and receive user input on the stock they want to see. After a secure connection was ensured, I plotted a simple line graph of the stock using the Matplotlib library for the user to see.

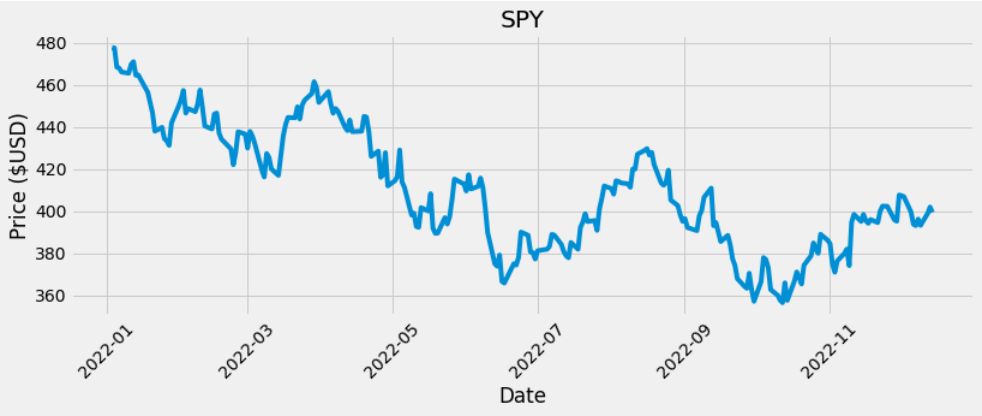

Figure 1: Year to Date(YTD) graph of ticker($SPY)

The next step was to create the two indicators I plan on using in my strategy. The Fibonacci levels are created by searching for the max and min of the data, subtracting the two, and subtracting the max from the distance time the ratios I wanted to use (23.6%, 38.2%,50%,

61.8%, 78.6%). The .618 percent ratio is often referred to as the "golden ratio" and is considered an important level to key an eye on for buying and selling.

```
[ ]  #Calculate Fibonacci Retracement Levels

     max_price = df['close'].max()
     min_price = df['close'].min()

     #diffenece between the max and min to create ratio levels calculated in the next step
     difference = max_price - min_price

     l1 = max_price- difference *0.236
     l2 = max_price - difference * 0.382
     l3 = max_price - difference * 0.5
     l4 = max_price - difference * 0.618
     l5 = max_price - difference * 0.786
```

Figure 2: Fibonacci levels calculation

To calculate the MACD, I first calculated the short EMA using a 12-day average and then calculated the long EMA using the 26-day average as this is a common default value. From there I subtract the short from the long to get the EMA which is now used to calculate the signal line. The signal line is the moving average of the MACD and I used a 9-day time frame. I then added the signal and MACD lines into the dataset for later use. Using these new indicators I created, I plotted these onto the graph.

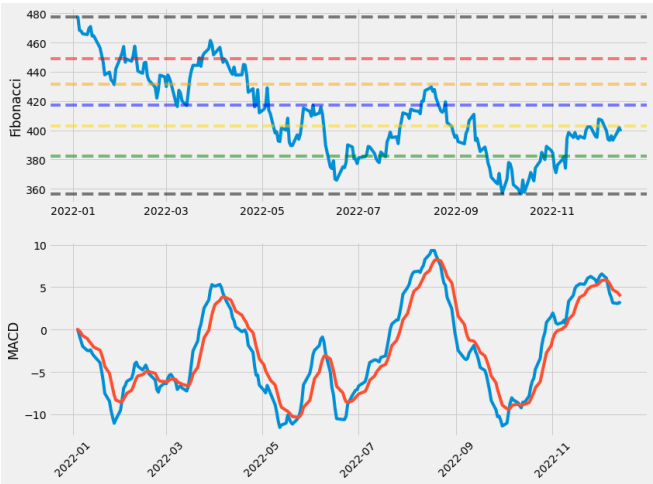

Figure 3: Added Fibonacci and MACD/Signal Lines to graph

Before creating my strategy, I defined a couple of rules I wanted to follow: 1) When the signal line crosses above the MACD Line and the current price crosses above or below the last fibs level then buy 2) When the signal line crosses below the MACD Line and then-current price crossed above or below the last fibs level then sell 3) Never sell at a price that's lower than what I bought it at 4) Function returns buy and sell arrays for each trade. To do this, I first needed to create a helper function that determines the lower and upper fib levels of the current price of the stock to create a boolean to see if it hits either of those levels later in the strategy.

```
def getFibLevels(price):
  if price >= l1:
    return(max_price, l1)
  elif price >= l2:
    return(l1,l2)
  elif price >=l3:
    return(l2,l3)
  elif price >=l4:
    return(l3,l4)
  elif price >= l5:
    return(l4,l5)
  else:
    return(l5,min_price)
```

Figure 4: getFibLevels helper function

Now I started to create my algorithm as I now have all the information I need. The function first takes in a data frame as a parameter. Next, I created a buy and sell list to keep track of each buy and sell price that will be used for calculating returns. I also needed to create a last buy price variable to make sure that the price I sell at is higher than where we bought it and a flag to indicate when we buy(1) and sell(0). The flag also represents if we have an order open as well meaning that if it is currently 1, then we have an open order we haven't closed yet so that we can have two orders open at the same time. The function then loops through the date frame and sets the upper and lower Fibonacci levels of that current price in the interaction. We then get into our

first conditional if else where I see if the current price is greater than or equal to the upper level or less than or equal to the lower level to see if a Fibonacci level was hit. If it is not hit then we update the lower and upper levels for the price. If the Fibonacci level was hit in the outside elif statement, we enter another if/elif statement comparing the signal and MACD line as well as checking and updating the flag. Going back to my rules, if the signal line is above the MACD line then we enter that trade. Once we have an open trade it repeats through the loop again. My sell condition is two parts in which we see if the signal line is below the MACD line and the last sold price is above the current price so it is always profitable. If the sell condition is hit, it updates the flag and continues the process. At the end of the dataset, the algorithm will return the sell and buy lists. I then added the two lists to the data frame each with their respective column.

```
def strategy_algo(df):

  buy_list = []
  sell_list = []

  flag = 0 #flag for buy or sell 0 - SELL 1 - BUY

  last_buy_price = 0;

  #loop through the data
  for i in range(0, df.shape[0]):
    price = df['close'][i]

    #if this is the first data point within the data set, then get the level above or below it
    if i == 0:
      upper_lvl, lower_lvl = getFibLevels(price)
      buy_list.append(np.nan)
      sell_list.append(np.nan)

    #else if the current price is greater than or equal to the upper_lvl, or less than or equal to the lower_lvl then we know price has hit/crossed a new fib level
    elif price >= upper_lvl or price <= lower_lvl:

      #Check if MACD line crossed above or below the signal line
      if df['Signal Line'][i] > df['MACD'][i] and flag == 0:
        last_buy_price = price
        buy_list.append(price)
        sell_list.append(np.nan)

        flag = 1

      elif df['Signal Line'][i] < df['MACD'][i] and flag == 1 and price >= last_buy_price:
        buy_list.append(np.nan)
        sell_list.append(price)

        flag = 0

      else:
        buy_list.append(np.nan)
        sell_list.append(np.nan)

    else:
      buy_list.append(np.nan)
      sell_list.append(np.nan)

      #update new levels
      upper_lvl, lower_lvl = getFibLevels(price)

  return buy_list, sell_list
```

Figure 5: My trading algorithm

Finally, I plotted the final graph with all the indicators and buy and sell targets generated from the newly created algorithm. I was very pleased to see that the algorithm was working to my set conditions and was able to generate multiple buys and sell targets.

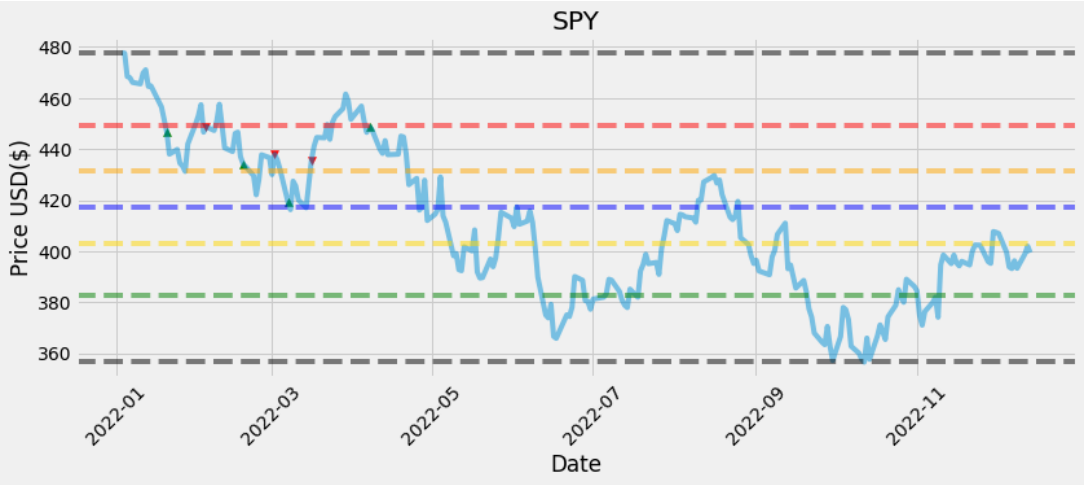

Figure 6: Updated Chart with buy and sell signals

Design Implementation

The next part of my design is to convert this Python code to a Pine script to be backtested on a stock. TradingView is a charting tool used by retail and professional traders utilizing hundreds of built-in and created indicators. One of their most powerful tools is the Pine scripting language which allows users to create a study, indicator, or strategy. For my use, I needed to create a strategy. The strategy returns key metrics such as net profit, trades closed, percent profitable, Sharpe ratio, and average trade return. When creating this script I first initially ran into a couple of understanding issues. Since it was a scripting language and everything ran at run time, I didn't need to import data and could just use the ‘close’ keyword to get the closing price of that stock on that day. The first step in creating a strategy was to define the strategy with some given inputs such as trade balance, chart overlay for plots, and percent of equity to be used to

buy a contract. These can all be changed in the chart settings for a later time. One thing different about the Python script from the Pine script is that I needed to create a timeframe for the strategy to be tested on because by default it does it on all historical data.

```
strategy("MACD and Fibonacci Levels Trading Algorithm", overlay =  true, initial_capital = 1000, default_qty_value = 1, default_qty_type = strategy.percent_of_equity)

// === INPUT BACKTEST RANGE ===
fromMonth = input.int(defval = 1,    title = "From Month",  minval = 1, maxval = 12)
fromDay   = input.int(defval = 1,    title = "From Day",    minval = 1, maxval = 31)
fromYear  = input.int(defval = 2021, title = "From Year",   minval = 1970)
thruMonth = input.int(defval = 1,    title = "Thru Month",  minval = 1, maxval = 12)
thruDay   = input.int(defval = 1,    title = "Thru Day",    minval = 1, maxval = 31)
thruYear  = input.int(defval = 2112, title = "Thru Year",   minval = 1970)


// create function "within window of time"
start     = timestamp(fromYear, fromMonth, fromDay, 00, 00)
finish    = timestamp(thruYear, thruMonth, thruDay, 23, 59)
window()  => time >= start and time <= finish ? true : false
```

Figure 7: Strategy declaration and window() bool created

I then calculated the MACD and signal line from user input for short and long EMA using a built-in function called ‘ta.macd.’ and then plotted those two on the chart. Another change I made to the Pine script compared to the Python one was that I created automatic Fibonacci levels based on user input that creates the levels based on the number of days you input. Before, it created levels based on the timeframe but allowing it to be changed creates tighter levels and therefore more trades can occur.

```
//Moving fibonacci length, input is number of days
fibLength = input.int(30, title = "Fibonacci Length")

// Compute Fibonacci levels
Fib(len,ratio) =>
    difference = ta.highest(len)-ta.lowest(len)
    range_ratio = ta.highest(len) - (difference*ratio)


//fib levels based on ratio
max = Fib(fibLength,0)
l_236 = Fib(fibLength,.236)
l_382= Fib(fibLength,.382)
l_50 = Fib(fibLength,.5)
l_618 = Fib(fibLength,.618)
l_786 = Fib(fibLength,.786)
min = Fib(fibLength, 1)
```

Figure 8: Automatic Fibonacci Levels created based on timeframe

Figure 9: Automatic Fibonacci Levels plotted

The next step was to create the helper function used to determine where the current price lies between the levels to create our higher and lower levels. Since Pine cannot return two items at the same time in a function call, I had to create two functions for the higher and lower levels. Next, I created a boolean variable that checks to see if the price is at or below the fibs level to then be used in my algorithm. From there I was then able to start making my algorithm. It

follows the same trade logic and structure as the Python script as it first starts and sees if a level was hit. When it gets to the buy logic and if it is true, it then opens a trade using the 'stategy.order()' function while making sure it's in the given timeframe. To create a buy order, its parameter must be set to 'strategy.long' to indicate an opening of an order. This then buys 1 share at the close price of that given day. Similarly, to sell that position we use the same function except with the parameter 'strategy.short' to generate a sell order at that close price. If neither condition is met, it then just updates the levels.

```
//my python trading strategy recreated in python using slightly different trade logic as there is no for loops

//create arrays to set and update flag and last buy price because the var is immutable
var last_buy_price = array.new_float(1,0.0)
var flag = array.new_float(1,0.0)


//if price hit the fib level
if hitFibLevel
    //same trade logic as the python script
    if signalLine > macdLine and array.get(flag,0) == 0 and window()
        array.set(last_buy_price, 0, close)
        array.set(flag,0,1)

        //open an order with Long indicator and buy 1 share at market price
        strategy.order("Buy",strategy.long, qty = 1, comment = "Buy 1 Share")
    if signalLine < macdLine and array.get(flag,0) == 1 and close > array.get(last_buy_price,0) and window()
        array.set(flag,0,0)
        //close the order with short indicator and sell 1 share at market price
        strategy.exit("Sell",from_entry = "Buy",qty = 1,stop = close, comment = "Sell 1 Share")

//update lower and higher fib level
else
    upper_lvl := getFibLevels_higher(close)
    lower_lvl := getFibLevels_lower(close)
```

Figure 10: Trade Logic generated in Pine

After creating this function, the script is running in real-time and will generate different results as the inputs change. Some trouble I had was just generally understanding how to write the code as this is my first time writing something in a scripting language. I didn't realize that for loops do not need to be used and that changing an int value in a function is not allowed. One main issue that I really ran into was that I had no debugging console and I would only know the issues line

number with a small description in the compile window. There were many instances where I didn't know what the value of a certain bar or variable was and couldn't access it.

Design Testing

There is a couple of different ways I thought about testing this strategy out. To test this strategy's max profitability, I will keep the MACD inputs constant while changing the bar range(1min, 5min, 30min, 1hr, 2hr, 1D) and Fibonacci length as well(5,10,50,100) on a ten-year time frame. I will test on just the S&P 500 index fund ($SPY) as it provides a lot of historical data. I plan on seeing what time frame and length create the most profitable inputs. When I first started to run the code I ran into an issue as it wasn't displaying any trade history. I looked into my inputs and when I unchecked the test from the backtesting range, it finally pulled data, but not within my backtesting dates. So I suspected and realized there is something wrong with my code involving the backtesting input and the boolean variable checking to see if the bar is within those dates. To fix this issue, I had to create a function that would actively return true or false of the boolean variable if the bar is within that range because my issue was that my variable was not updating as the range was changed. After I changed that, the code was working as expected and I updated the user input settings box. After fixing this, I was then able to start testing my strategy based on different inputs.

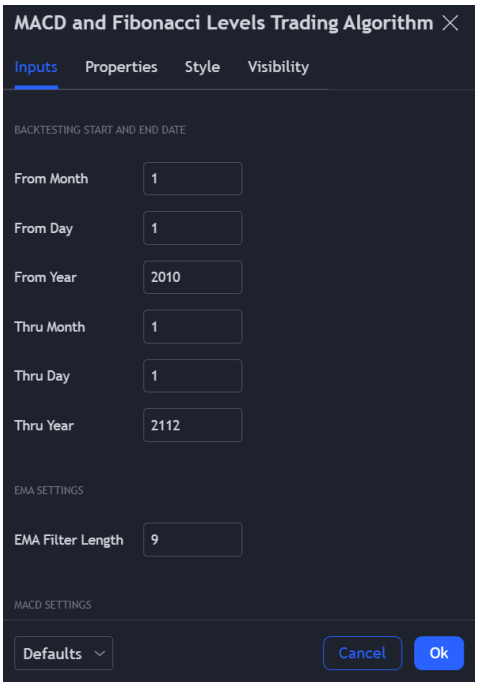

Figure 11: Updated Input Settings Window

$SPY 01/01/2012-12/16/2022 (10 Years)

Time Frame	**1 min**	**5 min**	**30 min**	**1hr**	**2hr**	**1D**	**Average**
Fib Length							
5	2.58%(39)	5.17%(19)	16.82%(34)	16.32%(25)	24.5%(21)	12.54%(6)	**12.99%(24)**
10	3.11%(30)	5.26%(16)	14.51%(22)	16.59%(18)	23.31%(17)	12.41%(6)	12.53(18.16)
20	2.75%(27)	6.54%(15	11.38%(17)	14.49%(13)	17.71%(15)	12.24%(5)	10.85%(14.5)
50	3.81%(18)	5.28%(9)	5.52%(8)	11.29%(9)	16.14%(11)	7.88%(3)	8.32%(9.6)
100	3.35%(10)	5.26%(6)	5.34%(6)	17.34%(8)	12.23%(7)	2.13%(2)	7.61%(6.5)
Average	3.12%(26.8)	5.51%(13)	10.71%(17.4)	15.21(14.6)	**18.78%(14.2)**	9.44%(4.4)	

Figure 12: $SPY 10 Year returns based on Fibonacci Length and timeframe

Summary, Conclusion, and Future Work

From Figure 12, we can see the final testing results of a ten-year time frame of SPY. In the table, I documented the return of the two indicators as well as the number of trades taken in parentheses. I was able to conclude that the two-hour time frame and a Fibonacci length of 5 produced the highest average return. This was interesting to see as a two-hour time frame is used when you are looking at data in terms of a couple of weeks/months rather than years. I also did expect the five and lower Fibonacci length to have higher returns as it's more sensitive and triggers more trades as compared to the longer lengths. These both can be confirmed as the highest return on the chart was on a two-hour time frame and Fibonacci length of 5.

Looking forward, there were many other features I would like to incorporate. First I would like to add two more indicators to my strategies such as linear regression and Bollinger bands as these are some of my favorite indicators to use with each other. The way I would use it is similar to the other two in that if the stock hits the lower regression level, for example, it would then send a buy signal and sell if it hits a higher regression level. This would add some more complexity to my code and would be cool to see how it all works out together. Another thing I would like to do after completing the strategy is to host this fully in AWS. The reason for this is that it can then run by itself and optimize the strategy based on a given input. The end goal is that it would connect to my broker and buy and sell based of the strategy.

In summary, I thoroughly enjoyed this project as I was able to fully create and backtest a trading strategy I use commonly. I was able to learn how to develop code starting from research and learned a lot of new technologies on the way. The skills and tools I learned from this project can be used down the road in my future job.

LINKS:

Google Collab Python Link:
https://colab.research.google.com/drive/1x_63_AxtXqaIhVbzs8SXTYlMDtZoSHpL?usp=sharing

TradingView Pine Script Link:
https://www.tradingview.com/script/1cIO5SxH-MACD-and-Fibonacci-Levels-Trading-Algorithm/

Video Presentation:
https://www.youtube.com/watch?v=Pmu4GjYWOvQ