TEXAS McCombs | MASTER OF SCIENCE IN BUSINESS ANALYTICS

Stochastic Control and Optimization

INTEGER PROGRAMMING REPORT

[Project 2]

as part of **Optimization I**

Prepared By

Group 7:	**EID**
Ian Arzt	iga246
Soumi Basu	sb59982
Soumya Nayak	sn25829
Allie W Touchstone	awt529

Submitted to

Daniel Mitchell

November 14, 2021

Table of Contents

1. Background

Money Management is something which has gained increasing precedence over the past few decades. Among all the means of income, **equity investments** have been by far the hottest way for an individual to make money. Equity represents the value that would be returned to a company's shareholders if all the assets were liquidated and all of the company's debts were paid off. Equity investments are optimized by traders using the concept of Money management which is a strategy for increasing or decreasing the position size to limit risk while achieving the greatest growth possible from a trading account.

The stock market of USA has various stock indexes like S&P-500, FTSE-100, NIFTY-50, etc., which help us segregate and understand the best performing firms based on their stock performance calculated on the basis of certain underlying assumptions of those indexes. The indexes help us track the ups and downs of a chosen group of stocks. These investments are often grouped around a particular industry, like tech stocks, or even the stock market overall. Also, tracking the performance of the indexes can help us understand the overall market performance and provide guidance to financial firms and exchange traded funds (ETFs).

2. Project Overview

Our goal is to construct a portfolio (index fund) and provide recommendations on the number of stocks to include. This will be done through passive portfolio management which is a long-term strategy. This works through indexing, where we will select the stocks that best represent the movements of the broad market population or market index benchmark.

In this project, we are trying to create an Index fund with m stocks to track the NASDAQ-100 index as closely as possible ensuring that the number of component stocks are as minimum as possible. Using less number of stocks will help ensure that the cost of rebalancing is the least as well as there is no need to hold unnecessary positions. This objective will be achieved in two steps. In the first step the optimizer would choose a portfolio which minimizes the absolute difference between the overall index return and the overall portfolio return. In the second step the optimizer will choose the stocks to be included and their respective weights. This will be followed by recommendation of a specific portfolio based on our analysis. We will use the January-December 2019 daily prices of the index and the component stocks of NASDAQ-100 to construct the portfolio and the 2020 data to analyze the performance.

3. Part 1 – Stock Selection & Weight Construction

The first step involving building of the model which includes stock selection and weight construction. This step involves selecting those stocks which are best representative of the overall stocks in the index fund, thus mimicking the overall performance of the index the closest, while also keeping the number of stocks at the minimum.

In order to select the stock-basket, which represents NASDAQ-100 the closest, we will be evaluating the correlation between the returns of the selected stocks and the rest of the stocks followed by maximizing the sum of all the correlation across all stocks.

This is done by the following code:

Return Calculation

```
returns_df = pd.DataFrame()

for stock in stocks_2019.columns[2:]:
    returns_df[stock] = (stocks_2019[stock]) / (stocks_2019[stock].shift()) - 1

returns_df = returns_df.dropna().reset_index().drop('index', axis = 1)
```

Correlation

```
p = returns_df.corr()
stocks = pick_stocks(m, p, returns_df)
```

Constraints to be considered while solving:

a) **Stock Selection:**

1. select the exact number of stocks to be held in the fund as mentioned
2. each stock i can have exactly one representative stock j in the index
3. guarantee that stock i is best represented by stock j only if j is in the fund

b) **Weight Calculation**

1. yi ≥ mod(index_return - weighted stock return)
2. Sum of wi = 1

Where,
yi is the difference between index overall return and weighted stock return and,
wi is the weights for each of the selected stocks.

Stock Selection

Here our main objective is to maximize the correlation of the stocks selected with the overall stocks while also keeping the number of stocks selected to the minimum. The selection process is achieved by the following function:

```
def pick_stocks(m, p, returns):
    num_stocks = len(p)
    obj = np.array(p.values.flatten().tolist() + [0] * num_stocks)
    A = np.zeros(((num_stocks ** 2 + num_stocks + 1), (num_stocks ** 2 + num_stocks)))
    b = np.zeros(num_stocks ** 2 + num_stocks + 1)
    direction = np.array([''] * len(b))

    # Constraint 1
    A[0,-len(p):] = [1] * (num_stocks)
    b[0] = m
    direction[0] = '='
    # Constraint 2
    ind_vec = np.array(range(num_stocks))
    row = 1
    for j in range(num_stocks):
        A[row,j*num_stocks + ind_vec] = 1
        b[row] = 1
        direction[row] = '='
        row+=1
    # Constraint 3
    counter = 0
    while row != A.shape[0]:
        for i in range(num_stocks):
            A[row, [(i * len(p) % len(p)**2) + counter, counter + len(p)**2]] = [1, -1]
            b[row] = 0
            direction[row] = '<'
            row += 1
        counter += 1

    # c1,c2,c3,c4,y1,y2
    #  1  0  0  0  -1 0
    #  0  0  1  0  -1 0
    #  0  1  0  0  0  -1
    #  0  0  0  1  0  -1

    # xij <= yj
    # b = 0

    Model = gp.Model()
    Mod_x = Model.addMVar(len(obj), vtype = 'B')
    Mod_con = Model.addMConstrs(A, Mod_x, direction, b)
    Model.setMObjective(None,obj,0,sense=gp.GRB.MAXIMIZE)
    Model.Params.OutputFlag = 0
    Model.optimize()
    x = Model.x
    temp_df = pd.DataFrame(x)
    stocks = temp_df[(temp_df[0] == 1) & (temp_df.index >= (num_stocks ** 2))].index.values - num_stocks**2

    selected_stocks = returns.iloc[:,stocks]
    return selected_stocks
```

Weight Construct

Further, the portfolio weight needs to be calculated. This is done by allocating the optimal weights to the selected stocks by minimizing the difference between the weighted stock return and the index overall return.

This is achieved by the following function:

```
def get_weights(x, data):
    obj = np.array([0] * x.shape[1] + [1] * x.shape[0])

    A = np.zeros((2*x.shape[0] + 1, len(obj)))
    b = np.zeros(2*x.shape[0] + 1)
    direction = np.array([''] * (2*x.shape[0] + 1))

    # Constraint 1
    # weights must add up to one
    A[0,:x.shape[1]] = 1
    b[0] = 1
    direction[0] = '='


    row = 1
    # Constraint 2
    # get the returns of each stock for each day and set the difference to 1
    for i in range(len(x)):
        A[row, :x.shape[1]] = x.iloc[row - 1].tolist()
        A[row, i+x.shape[1]] = 1
        b[row] = data.iloc[row - 1, 0].tolist()
        direction[row] = '>'
        row += 1

        # y1  y2  d1  d2
# day 1  ry1 ry2  1   0
# day 2  ry1 ry2  0   1

# b[0] <- corresponds to day1 = return of the index on day 1

    # Constraint 3
    for i in range(len(x)):
        return_vals = (x.iloc[row - (x.shape[0] + 1)].values) * -1
        A[row, :x.shape[1]] = return_vals.tolist()
        A[row, i+x.shape[1]] = 1
        index_vals = (data.iloc[row - (x.shape[0] + 1), 0]) * -1
        b[row] = index_vals.tolist()
        direction[row] = '>'
        row += 1

         # y1  y2  d1  d2
# day 1  -ry1 -ry2  1   0
# day 2  -ry1 -ry2  0   1

# b[0] <- corresponds to day1 = negative return of the index on day 1

    Model = gp.Model()
    Mod_x = Model.addMVar(len(obj))
    Mod_con = Model.addMConstrs(A, Mod_x, direction, b)
    Model.setMObjective(None,obj,0,sense=gp.GRB.MINIMIZE)
    Model.Params.OutputFlag = 0
    Model.optimize()
    return Model.x[:x.shape[1]]
```

4. Part 2 - The best 5 stocks and their weights

The objective of this part is to find the best 5 stocks to include in our portfolio and the weights of those 5 stocks, using the 2019 data. This is done by similar approach as discussed in Part 1 above where we derive correlation to determine the best representative stock included in the portfolio of each stock in the index. The objective is to maximize the similarity/correlation between the stocks in the index and the representative stocks in the portfolio. The details of the 5 representative stocks included in the portfolio are as follows:

LBTYK	MXIM	MSFT	VRTX	XEL
0.048862	0.210388	0.580352	0.07119	0.089208

Observation:

MSFT represents 58% of the portfolio followed by MXIM which represents 21% of the portfolio.

The performance of this portfolio in its ability to track the index of 2020 is as follows:

2019	2020
M 5	M 5
0.789178	1.112437

We see that the tracking error for 2019 data is 0.79 which is considerably poor. This is due to less number of stocks being selected (m = 5). This however increases with 2020 data, as the tracking error increases to 1.1. Thus the portfolio with m=5 stocks is unable to adequately capture all the variance in the NASDAQ-100.

5. Part 3 – Iterating for different values

We re-ran part 2 with values of m ranging from 10 to 100.

Calculate Tracking to the Index

This calculates difference in tracking returns with the weights we got for 2019 (apply these weights onto stocks in 2020 and compare to 2020 index) compared to 2020 index

```
def calculate_tracking(weights, data):

    difference = 0
    for t in range(len(data)):
        day_return = 0
        for s in range(len(weights)):
            day_return += data.iloc[t,s+1] * weights[s]
        difference += abs(data.iloc[t,0] - day_return)
    return difference
```

```
f_track_2020 = pd.DataFrame()
f_track_2019 = pd.DataFrame()

for m in range(10,stocks_2020.shape[1], 10):
    stock_2019 = get_entire_performance(m, p, returns_df, returns_df_ndx, returns_df_ndx, return_stocks = 'Y')
    returns_df_ndx_copy = returns_df_ndx[['NDX'] + stock_2019.columns.values.tolist()]
    weights_copy = get_weights(stock_2019, returns_df_ndx_copy)

    returns_df_ndx_2020_copy = returns_df_ndx_2020[['NDX'] + stock_2019.columns.values.tolist()]

    tracking_2019_copy = calculate_tracking(weights_copy, returns_df_ndx_copy)
    tracking_2020_copy = calculate_tracking(weights_copy, returns_df_ndx_2020_copy)


    f_track_2020["M " + str(m)] = [tracking_2020_copy]
    f_track_2019["M " + str(m)] = [tracking_2019_copy]
```

2019 (In-sample) and 2020 (Out-of-sample) performance results:

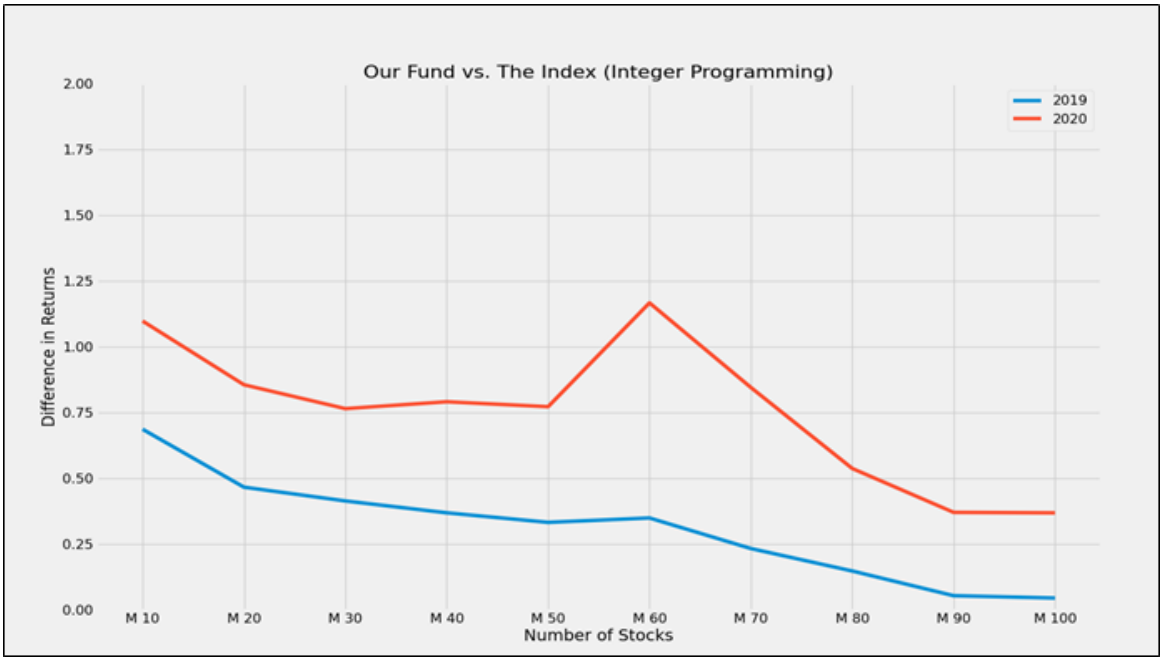
f_track_2019

	M 10	M 20	M 30	M 40	M 50	M 60	M 70	M 80	M 90	M 100
0	0.701218	0.466268	0.409792	0.363281	0.33254	0.352056	0.233143	0.147683	0.053827	0.044911

f_track_2020

	M 10	M 20	M 30	M 40	M 50	M 60	M 70	M 80	M 90	M 100
0	1.102404	0.855446	0.767367	0.767891	0.7721	1.164932	0.861893	0.537323	0.370506	0.368671

Graphical representation:



From the graph, we can see that as m increases from 10 to 100, the difference in return (the difference in the total return from the returns of the selected stocks at each value of m) is decreasing overall. The lowest value is at m=100, which is where all stocks in the portfolio are considered. This held true for both the 2019 and 2020 stocks.

In-sample performance (2019 stocks):

When recommending a portfolio which contains less than 100 stocks, we can see that the difference in return is low at m = 50 stocks after which it starts slightly increasing and before decreasing again. Overall, it does not provide a very clear indication in this regard.

Out-of-sample performance (2020 stocks):

When recommending a portfolio which contains less than 100 stocks, we can see that the smallest difference in return is at m=30 stocks after which it starts increasing until it peaks at m=60 and then declining again. We consider m = 30 stocks to be a good portfolio based off of the 2020 stock data.

6. Part 4 – Method 2 – Mixed Integer Programming

MIP Optimization by adding the "Big M" constraint

We added binary variables (y1, y2, …, yn) and integer variables to force the weights to 0 when yi=0 (based on binary variables 0 or 1). We have taken M as 1 because that is the highest value the weights and the binary variable can take.

```
############################
##### 10 Hour Program #######
############################
temp_df = pd.DataFrame(columns = returns_df.columns.values)
model_x_list = []
for m in range(10, stocks_2020.shape[1], 10):
    weights, model_x = optimize_portfolio(m, p, returns_df, timestop, returns_df_ndx)
    temp_df.loc[m, :] = weights
    temp_df.to_csv('m' + str(m) + ".csv")
    model_x_list.append(model_x)
```

After adding all the constraints, we set the TimeLimit value(timestop) at 1 hour and run part 2 and 3 (considering all the m values from 10 to 100).

```
data3 = pd.DataFrame()

for stock in stocks_2020.columns[1:]:
    data3[stock] = (stocks_2020[stock]) / (stocks_2020[stock].shift()) - 1

data3 = data3.dropna().reset_index().drop('index', axis = 1)
final_tracking_2020 = pd.DataFrame()
final_tracking_2019 = pd.DataFrame()

for row in temp_df.index:
    weight_list = temp_df.loc[row, :].to_list()
    data3_copy = data3[['NDX'] + temp_df.columns.values.tolist()]

    tracking = get_entire_performance(row, p, returns_df, returns_df_ndx, data3_copy, problem = 'MIP', weight_list = weight_list)
    tracking_2019 = get_entire_performance(row, p, returns_df, returns_df_ndx, returns_df_ndx, problem = 'MIP', weight_list = we

    final_tracking_2020['M ' + str(row)] = tracking['M ' + str(row)]
    final_tracking_2019['M ' + str(row)] = tracking_2019['M ' + str(row)]
```

2019 (In-sample) and 2020 (Out-of-sample) performance results:

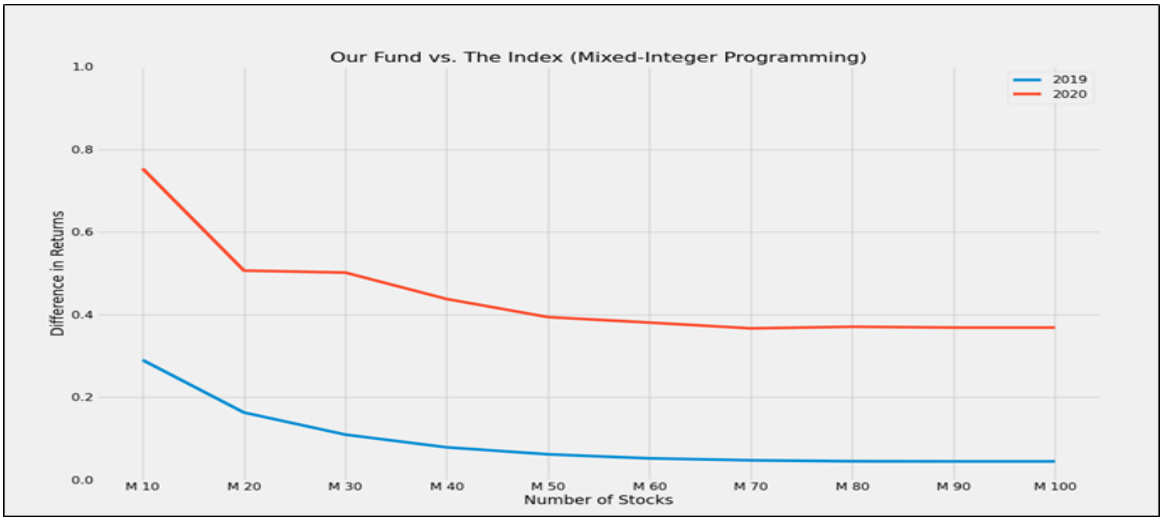
final_tracking_2019

	M 10	M 20	M 30	M 40	M 50	M 60	M 70	M 80	M 90	M 100
0	0.290137	0.163126	0.109681	0.078939	0.062055	0.052246	0.047617	0.045227	0.044911	0.044911

final_tracking_2020

	M 10	M 20	M 30	M 40	M 50	M 60	M 70	M 80	M 90	M 100
0	0.753372	0.50645	0.501655	0.437683	0.39397	0.380664	0.366778	0.370629	0.368682	0.368682

Graphical representation:



From the graph, we can see that as m increases from 10 to 100, the difference in return i.e., the difference in the total return from the return from selected stocks at each value of m is decreasing overall. The lowest value is obtained at m=100, where we are considering all the stocks in the portfolio. This held true in both the 2019 and 2020 stocks. The difference is that what we observe here is unlike the IP solution (part 3) where the decrease in the difference in returns is not consistent, the difference in return is more consistent with the increasing value of m.

7. Recommendations:

The main objective of our recommendation is to minimize both portfolio size and the difference between index return and portfolio return. Both IP and MIP optimizations are indicating that considering all 100 stocks will best mirror the NASDAQ-100 market index trend and fluctuation. But taking all 100 stocks will not be the most efficient and optimized solution. We need to take the best m which provides the smallest difference in return while reducing the cost and risk. For the 2020 stocks, as per IP optimization, we concluded that the balance is achieved at m=30. While in the MIP optimization, the balance is achieved at m between 60 and 70, because post 60 we do see any more reduction in the difference in returns.

Further, although the method of IP for determining stocks and their weights is computationally fast, it leads to poorer results when compared with Mixed Integer Programming. Thus our recommendation is to use the Mixed Integer Programming approach for determining the stocks to be included in the portfolio, with 70 stocks in the portfolio as the Tracking error is the least for m = 70.