Project 2 Optimization Report: Indexing Portfolio Strategy

Submitted to:

Dr. Daniel Mitchell, Professor
Optimization I, R M 294
05240

Prepared by:

Bindu Hebbar (br28722)
Deepti Hariyani (dh42224)
Khyati Jariwala (kkj448)
Spandan Pal (sp54969)
Teja Sirigina (ts37662)

Master of Science Business Analytics Program
The University of Texas at Austin
Austin, Texas

Fall 2023

Background

Indexing is a very popular and common passive equity money management strategy, where the goal is to create a portfolio that echoes the movements of the broad market population or a market index. The resulting portfolio is called an index fund. One approach to creating an index fund is to use m stocks, where m is much smaller than the size of the target population, n.

We aim to create an index fund with m stocks to track the NASDAQ-100, a popular market index. The methodology that we followed for this was to formulate an integer program to pick m out of n stocks for the portfolio, which takes a similarity matrix between stocks as an input. Then, we proceeded to solve a linear program to pick how much of each stock to purchase for the portfolio. As a final step, we then evaluated how our index fund performed in comparison to the NASDAQ-100 index and even evaluated the performance of the portfolio for different values of m. In other words, we evaluated the portfolio's performance if the number of stocks (m) included in the portfolio were to change. This report includes details about our methodology, findings, performance evaluations with different criteria, and recommendations as well as our closing remarks.

Stock Selection: Integer Programming (IP)

We treated stock selection as an integer programming problem, where we have a series of binary decision variables and their associated constraints. Our objective for the model is to maximize the similarity, or correlation in this case, between our m-stock portfolio and the entire index. We used a set of binary decision variables called Yj, which indicates which stocks j from the index, NASDAQ-100 in our case, are included in our fund. We also used another set of binary variables called Xij, which determines for each stock in the index, i, which stock j is the most similar to stock i. The objective and constraints for the stock selection problem are represented mathematically in the figure below.

$$\max_{x,y} \sum_{i=1}^{n} \sum_{j=1}^{n} \rho_{ij} x_{ij}$$

$$s.t. \sum_{j=1}^{n} y_j = m.$$

$$\sum_{j=1}^{n} x_{ij} = 1 \quad for\ i = 1,2,\dots,n$$

$$x_{ij} \leq y_j \qquad for\ i,j = 1,2,\dots,n$$

$$x_{ij}, y_j \in \{0,1\}$$

- $$\max_{x,y} \sum_{i=1}^{n} \sum_{j=1}^{n} \rho_{ij} x_{ij}$$

The objective is to maximize the correlation that exists between our m-stock portfolio and the whole index. To achieve this, we consider 100 Yj variables and 100*100 = 10,000 Xij variables. The Yj binary variables represent whether or not a stock is present in our m-stock fund. The Xij binary variables represent whether or not a stock j is the most similar to stock i.

In addition, some constraints must be met:

1. $$\because \sum_{j=1}^{n} y_j = m.$$

   The first constraint ensures that exactly m stocks must be present in our m-stock fund.

2. $$\sum_{j=1}^{n} x_{ij} = 1 \quad for\ i = 1,2, \dots, n$$

   The second constraint states that each stock i can only be “most similar” to only one stock j.

3. $$x_{ij} \leq y_j \quad for\ i, j = 1,2, \dots, n$$

   The third constraint ensures that the similarity between stock i and j can exist only if stock j is selected for the m-stock fund.

4. $$x_{ij}, y_j \in \{0,1\}$$

   The last constraint defines that xij and yj can only take values of 0 or 1,the variable type is set as ‘Binary’ in Gurobi, which is what we used to solve our optimization problems.

Calculating Portfolio Weights: Linear Programming (LP)

To fully set up a fund, the weights of the chosen component stocks need to be determined. The aim of the weight selection model is to minimize discrepancies between the fund return and index return over time. This is achieved by determining the weight for each component stock, represented by "wi", such that the following equation is minimized:

$$\min_{w} \sum_{t=1}^{T} \left| q_t - \sum_{i=1}^{m} w_i r_{it} \right|$$

$$s.t. \sum_{i=1}^{m} w_i = 1$$

$$w_i \geq 0.$$

Here, q represents the index's return on a given day "t", $r$it represents the return of stock "i" on a given day "t", and the sum is taken over all selected component stocks.

To prevent the negative and positive discrepancies from canceling each other out, using the absolute discrepancies between index and fund returns may cause additional issues as there is no direct way to represent absolute values within the objective function. To cope with this issue, one way is to break q up into two parts by defining an arbitrary decision variable yi for each selected stock “i” such that:

$$(1)\ y_i \geq q_t - \sum_{i=1}^{m} w_i r_{it}$$

$$(2)\ y_i \geq -\left(q_t - \sum_{i=1}^{m} w_i r_{it}\right)$$

With the above objectives, a model that optimizes weights for selected stocks can be created with formulations stated below:

Decision Variables:

wi : weights of stock “i” selected for portfolio

yi : max value of the possible difference between index return at t and weighted return of selected indexes

Constraints:

1. Sum of weights wi for m stocks in the fund index equals 1
2. The difference between the index returns and the weighted return of the stocks should be less than yi

Calculating the Correlation Matrix Using Stock Returns

For our purposes, we used daily prices of the index and component stocks of the NASDAQ-100 in 2019 for the creation of our portfolio and then we used data from 2020 to evaluate the performance of our portfolio. First, we began by calculating the returns of the stocks in 2019 in order to calculate the correlation matrix that will be used in future steps of stock selection and weight construction.

```
#Calculating the Returns for the stocks
for stock_name in stocks_available:
    stocks_2019[stock_name+"_return"] = (stocks_2019[stock_name] - stocks_2019[stock_name].shift(1)).fillna(0)/
                                        stocks_2019[stock_name].shift(1).fillna(1)
```

```
#Calculating the correlation between the available individual stocks
corr_matrix = pd.DataFrame(index=stocks_available, columns=stocks_available)

for i in stocks_available:
    for j in stocks_available:
        correlation = stocks_2019[i+"_return"][1:].corr(stocks_2019[j+"_return"][1:])
        corr_matrix[i][j]=correlation

corr_matrix.head()
```

Code 1. *Calculating returns for stocks and creating the correlation matrix between individual stocks.*

The code segment above depicts our approach to calculating the returns that are used to find the correlation matrix between individual stocks. To calculate the returns for each stock, we took the price of the stock on the previous day, subtracted it from the price of the stock on the current day, and then divided that by the price of the stock on the previous day. After calculating the returns for each stock, we then created the correlation matrix in order to quantify the relationships between the returns of the different stocks. A snippet of the 100x100 matrix is shown below.

	ATVI	ADBE	AMD	ALXN	ALGN	GOOGL	GOOG	AMZN	AMGN	ADI	...	TCOM	ULTA	VRSN	VRSK	
ATVI	1.0	0.399939	0.365376	0.223162	0.21628	0.433097	0.426777	0.467076	0.203956	0.329355	...	0.322906	0.128241	0.46485	0.316549	0.2
ADBE	0.399939	1.0	0.452848	0.368928	0.36337	0.552125	0.540404	0.598237	0.291978	0.473815	...	0.360392	0.201151	0.711339	0.541243	0.4
AMD	0.365376	0.452848	1.0	0.301831	0.344252	0.418861	0.417254	0.549302	0.151452	0.503733	...	0.332776	0.210623	0.498342	0.3309	0.2
ALXN	0.223162	0.368928	0.301831	1.0	0.332433	0.315993	0.307698	0.36317	0.342022	0.31704	...	0.257143	0.408936	0.350581	0.191489	0.5
ALGN	0.21628	0.36337	0.344252	0.332433	1.0	0.248747	0.250316	0.399281	0.264599	0.32828	...	0.175957	0.128559	0.360886	0.251855	0.3

5 rows × 100 columns

Table 1. *Correlation matrix that summarizes the relationships between the returns of each stock.*

The calculation of the correlation matrix is essential to the creation of the portfolio because we seek to select stocks for our portfolio that have returns that are highly correlated with the returns of the entire index.

Modeling the Optimization Problem: Stock Selection

For our optimization problem, which is to maximize the correlation between our m-stock portfolio and the entire index, we used 100 Yj binary variables, 10,000 Xij binary variables, and

embedded the necessary constraints within a Gurobi model. The Gurobi model utilized the correlation matrix created previously from 2019 data. Our selection of stocks was picked using the code snippet below, illustrating the model and its use of Gurobi's integer programming features.

```
#Setting the number of stocks for the fund
m = 5
```

```
#initializing the model in Gurobi
stocks_model=gp.Model()

#Creating the decision variables for the model.
stocks_model_present_X = stocks_model.addMVar(len(stocks_available),vtype='B')

best_stock_X = stocks_model.addMVar((len(stocks_available),len(stocks_available)), vtype='B')

#Adding the objective function for the model
stocks_model.setObjective(gp.quicksum(best_stock_X[i,j]*corr_matrix.iloc[i][j] for i in range(len(stocks_available))
                                      for j in range(len(stocks_available))),sense=gp.GRB.MAXIMIZE)

#Adding the constraints for the model
stocks_model_cons=[0]*3

stocks_model_cons[0] = stocks_model.addConstr(gp.quicksum(stocks_model_present_X[i] for i in range(len(stocks_available)))
                                              == m)
stocks_model_cons[1] = stocks_model.addConstrs(gp.quicksum(best_stock_X[i,j] for i in range(len(stocks_available))) == 1
                                               for j in range(len(stocks_available)))
stocks_model_cons[2] = stocks_model.addConstrs(best_stock_X[i,j] <=stocks_model_present_X[i]
                                               for j in range(len(stocks_available))
                                               for i in range(len(stocks_available)))

stocks_model.Params.OutputFlag = 0 # tell gurobi to shut up!!
stocks_model.optimize() #Solving the model

print("Optimzation Status is :",stocks_model.Status)

# print the optimal decision variable as a datafram for easy understanding
stocks_model_present_X.x
```

```
#show the selected stocks
selected_stocks = stocks_available[np.where(stocks_model_present_X.x)]
selected_stocks_position = np.where(stocks_model_present_X.x)
```

```
selected_stocks
```

```
Index(['LBTYK', 'MXIM', 'MSFT', 'VRTX', 'XEL'], dtype='object')
```

Code 2. *Selecting the best 5 stocks after solving the integer programming optimization problem with necessary constraints.*

First, we initialized the model in Gurobi with our decision variables and objective function, which was to maximize the correlation between the m-stock and the entire index. Then, we added the key constraints, which were discussed mathematically in a previous section. Finally, we ran the model to find our selected stocks when m=5, where m is the number of stocks in our portfolio.

The five most significant stocks of the index are Liberty Global PLC (multinational telecommunications company), Maxim Integrated (analog and mixed-signal integrated circuits company), Microsoft (multinational technology corporation), Vertex Pharmaceuticals

Incorporated (a biopharmaceutical company), Xcel Energy Inc (utility holding company), respectively. We initially set 'm' to 5 and utilized the 2019 dataset to identify which stocks our model selected. The selected stocks, along with their respective company and industry, are summarized in the table below.

Stocks Selected	Company	Industry
LBTYK	Liberty Global PLC	Telecommunications
MXIM	Maxim Integrated	Semiconductors
MSFT	Microsoft Corporation	Information Technology
VRTX	Vertex Pharmaceuticals Inc	Healthcare
XEL	Xcel Energy Inc	Utilities

Table 2. *The best 5 stocks selected, their company name, and corresponding industry.*

Notably, when examining the stocks chosen by our Integer program, we observe that each stock belongs to a distinct sector. Given the limitation of having only five stocks in our portfolio, diversification is crucial.

Modeling the Optimization Problem: Portfolio Weight Selection

Once the stocks were selected according to the constraints and objective, we then proceeded to find the optimal weights of each stock within the portfolio using our calculated stock return data for 2019 and selected stocks as the input to the Gurobi model. The calculation of portfolio weights to match the returns of the index is depicted in the code snippet below.

```
#get the number of stock returns
no_of_rows = len(stocks_2019)
```

```
Index_column_name = stocks_2019.columns[1]

stocks_2019[Index_column_name+"_return"] = (stocks_2019[Index_column_name] -
                                            stocks_2019[Index_column_name].shift(1)).fillna(0)/
                                            stocks_2019[Index_column_name].shift(1).fillna(1)
```

```
#initializing the model in Gurobi
stocks_weight_model=gp.Model()

#Creating the decision variables for the model
stocks_return_difference_X = stocks_weight_model.addMVar(no_of_rows,vtype='C')

stocks_weight_X = stocks_weight_model.addMVar(m,vtype='C')

#Adding the objective function for the model
stocks_weight_model.setObjective(gp.quicksum(stocks_return_difference_X[i] for i in range(no_of_rows)),sense=gp.GRB.MINIMIZE)

#Adding the constraints for the model
stocks_weight_cons=[0]*3

stocks_weight_cons[0] = stocks_weight_model.addConstr(gp.quicksum(stocks_weight_X[i] for i in range(m)) == 1)
stocks_weight_cons[1] = stocks_weight_model.addConstrs(stocks_return_difference_X[i]
                                                       >= stocks_2019[Index_column_name+"_return"][i] -
                                                       gp.quicksum(stocks_weight_X[j]*
                                                                   stocks_2019.iloc[i,selected_stocks_position[0][j] +2+
                                                                                    len(stocks_available)] for j in range(m))
                                                       for i in range(no_of_rows))
stocks_weight_cons[2] = stocks_weight_model.addConstrs(stocks_return_difference_X[i] >=
                                                       -stocks_2019[Index_column_name+"_return"][i] +
                                                       gp.quicksum(stocks_weight_X[j]*
                                                                   stocks_2019.iloc[i,selected_stocks_position[0][j] +2+
                                                                                    len(stocks_available)] for j in range(m))
                                                       for i in range(no_of_rows))


stocks_weight_model.Params.OutputFlag = 0 # tell gurobi to shut up!!
stocks_weight_model.optimize() #Solving the model

print("Optimization Status is :",stocks_weight_model.Status)
```

```
#show the optimum weights
stocks_weight_X.x
```

Code 3. *Finding the optimal weights of each stock within the portfolio of selected stocks.*

First, we initialized the Gurobi model with our constraints and objective, which was to minimize the difference in returns between the selected stocks and the index. Then, we added the key constraints discussed mathematically in a previous section. Finally, we ran our model to find the optimal weights of each stock in the portfolio to mirror the index returns as closely as possible. The resulting stock weights can be found in the following table.

Stocks Selected	Weights
LBTYK	0.048862
MXIM	0.210388
MSFT	0.580352
VRTX	0.071190
XEL	0.089208

Table 3. *The selected stocks and their associated weights in the portfolio.*

At a quick glance, we can see that Microsoft has the highest portfolio weight followed by Maxim Integrated. It's important to note that in the NASDAQ-100 index, a significant portion of the weight is attributed to technology giants like Apple, Google, and Microsoft. In practical terms, assigning a considerable weight to Microsoft in our portfolio is a strategic choice. It effectively aligns our portfolio with the composition of the NASDAQ-100 index, as it mirrors the weightings of key constituents, such as Microsoft, within the index. This strategic weighting allows our portfolio to closely mimic the performance and characteristics of the NASDAQ-100, which is desirable for replicating the index's behavior. We have also included the figure below to help visualize the portfolio weight distribution of each of the five stocks in our portfolio.

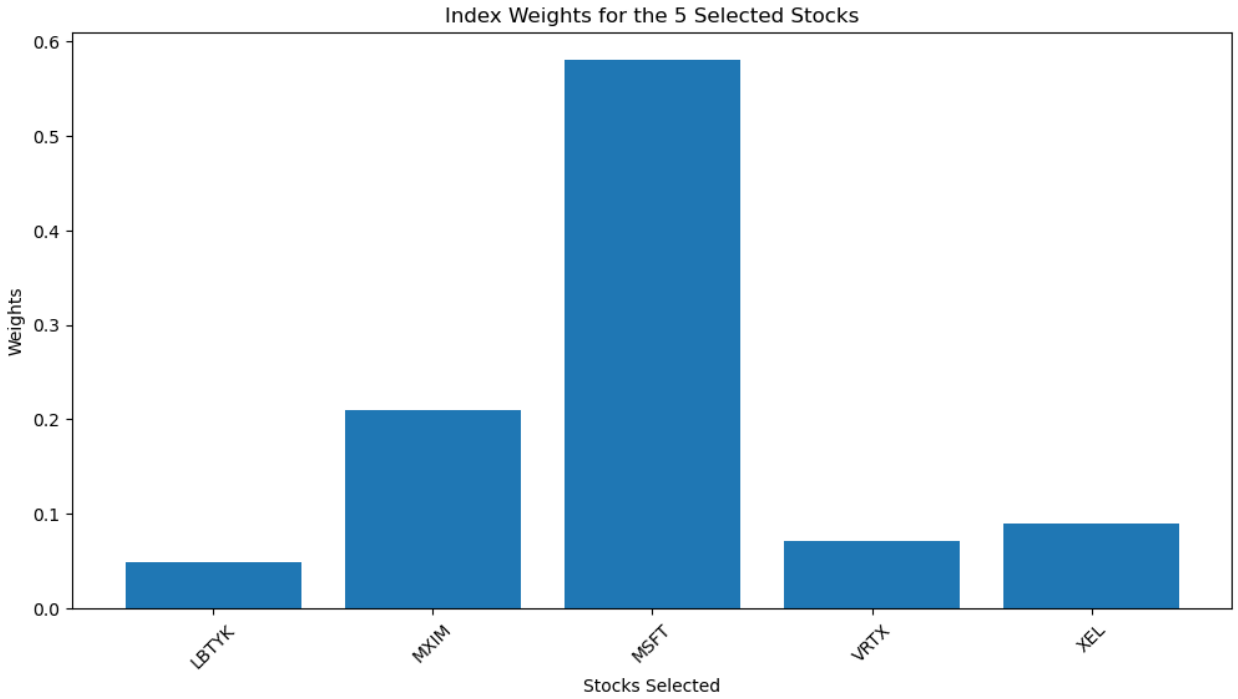

Figure 1. *Bar graph displaying the distribution of weights for each stock in the portfolio.*

Evaluation of Portfolio Performance Using the Best 5 Stocks

In order to evaluate the performance of our portfolio using the best 5 stocks, we needed to compare how the stock returns deviated from the NASDAQ index returns. We used data from 2020 and 2019 for our selected stocks and the index to assess the performance of our portfolio. The code snippet below details the process.

```
stocks_2020 = pd.read_csv("stocks2020.csv")

#Calculating the Returns for the stocks
for stock_name in stocks_available:
    stocks_2020[stock_name+"_return"] = (stocks_2020[stock_name] -
                                         stocks_2020[stock_name].shift(1)).fillna(0)/
                                         stocks_2020[stock_name].shift(1).fillna(1)

stocks_2020[Index_column_name+"_return"] = (stocks_2020[Index_column_name] -
                                            stocks_2020[Index_column_name].shift(1)).fillna(0)/
                                            stocks_2020[Index_column_name].shift(1).fillna(1)

#get the number of stock returns
no_of_rows_2020 = len(stocks_2020)
```

```
# Calculate the absolute deviation of the index fund
absolute_deviation_2019 = sum(abs(stocks_2019.iloc[i][Index_column_name+"_return"] -
                                  sum(stocks_weight_X.x[j] * stocks_2019.iloc[i][selected_stocks_position[0][j]+2+
                                                                                 len(stocks_available)] for j in range(m)))
                              for i in range(no_of_rows))

print("Absolute Deviation for 2019: ", absolute_deviation_2019)



# Calculate the absolute deviation of the index fund from NASDAQ for 2020
absolute_deviation_2020 = sum(abs(stocks_2020.iloc[i][Index_column_name+"_return"] -
                                  sum(stocks_weight_X.x[j] * stocks_2020.iloc[i][selected_stocks_position[0][j]+2+
                                                                                 len(stocks_available)] for j in range(m)))
                              for i in range(no_of_rows_2020))

print("Absolute Deviation for 2019: ", absolute_deviation_2020)
```

Code 4. *Calculating the deviations between our portfolio returns in 2019 and 2020 against the NASDAQ index returns.*

First, we calculated the returns for the stocks using the 2020 data using the same formula that we used to calculate the returns for stocks with 2019 data. Then, we calculated the absolute deviation of our selected stock returns compared to the index returns in years 2019 and 2020 to evaluate performance. The returns for our portfolio were calculated by multiplying stock weights with respective selected stocks and summing the individual returns. The results are summarized in the figure below.

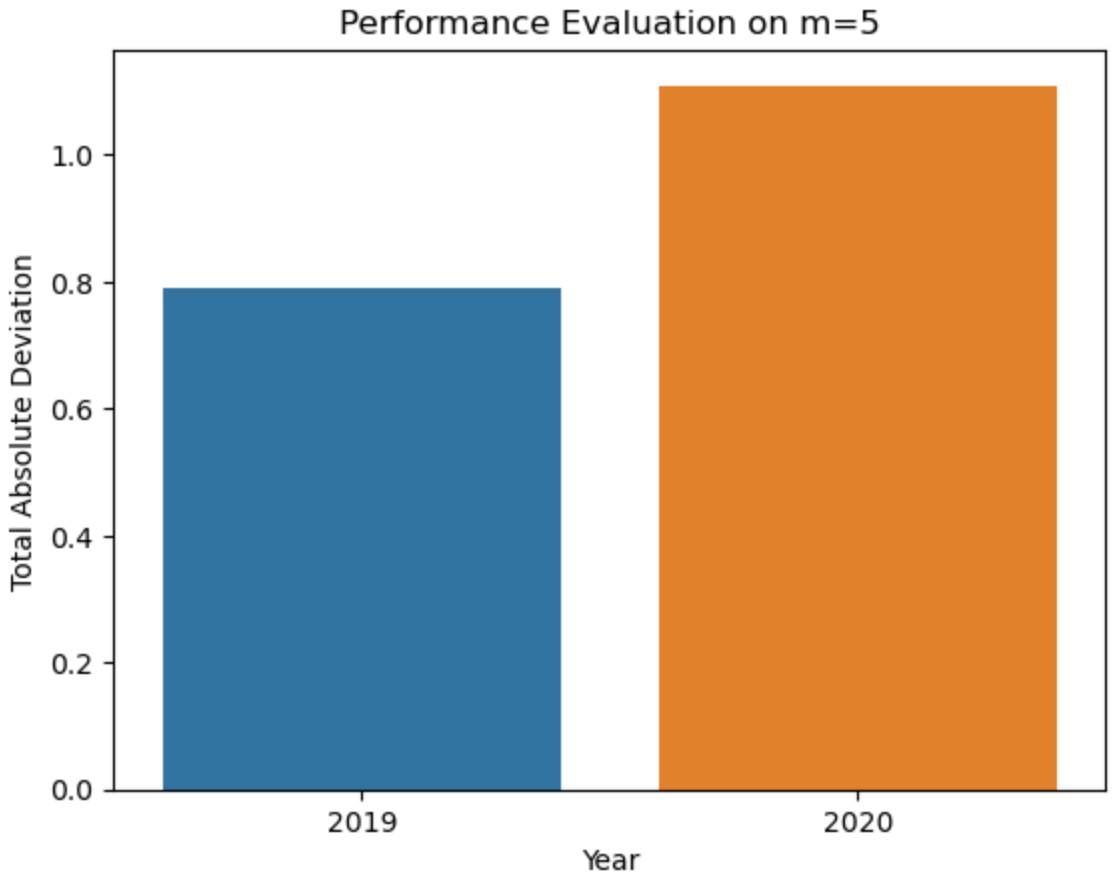

Figure 2. *Bar graph displaying the absolute deviation in returns between the portfolio and the index for the years 2019 and 2020.*

Upon evaluating the performance of the portfolio against the index, the absolute deviation in returns was calculated to be 0.789 for 2019 and 1.112 for 2020. We see that the portfolio has a better performance (lower deviation) in 2019 than in 2020, which makes sense given that portfolio weights are calculated using 2019 data. To allow for a better evaluation of the portfolio's performance, we created the line graph below that displays the performance of our portfolio fund against the NASDAQ index in 2020.

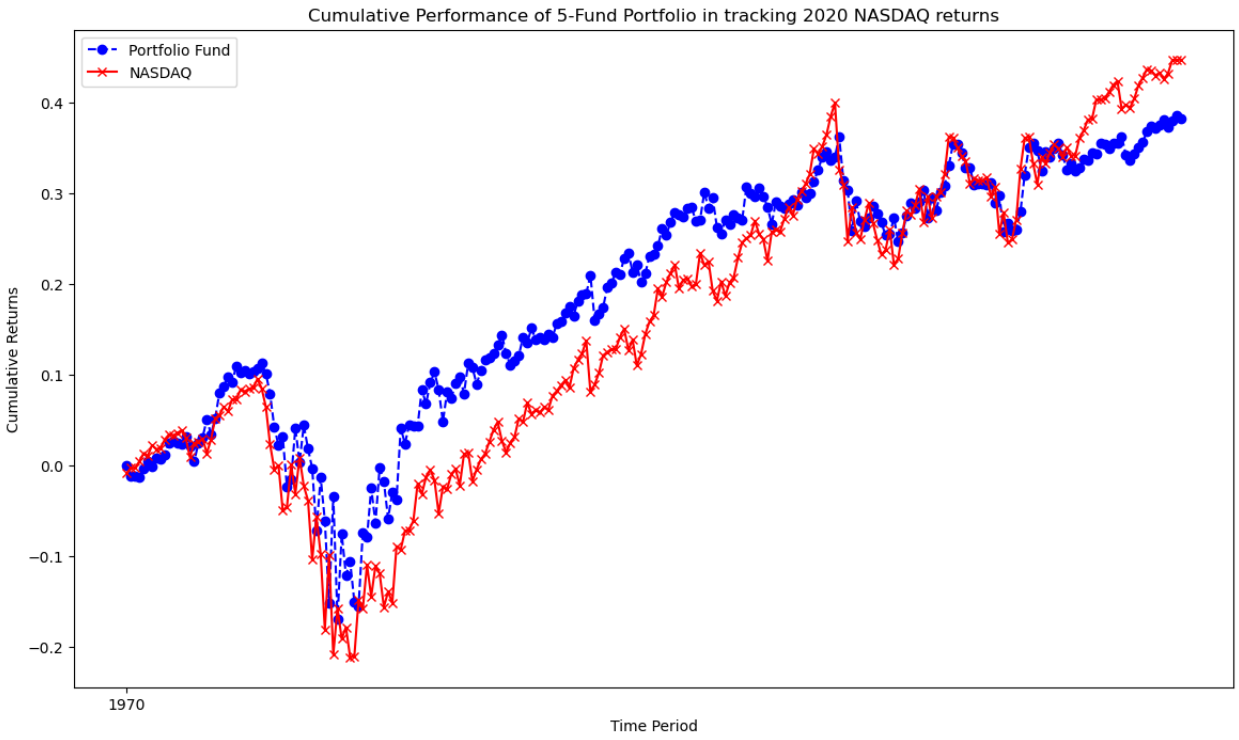

Figure 3. *Line graph depicting the cumulative performance of our 5-stock portfolio in tracking 2020 NASDAQ returns.*

Given that the lines for the portfolio fund and the NASDAQ are quite closely aligned with their cumulative returns over time, we believe that this 5-stock portfolio (m=5) does a fairly reasonable job of tracking the daily returns of the NASDAQ for the year 2020.

Evaluating Portfolio Performance by Changing the Number of Stocks Included

After establishing the workability of the models and optimization problem setup, we wanted to understand how changing the number of stocks affects the portfolio performance. By doing this, we aim to find the values of m that perform better than our current portfolio and to identify if there is a value of m, where there are diminishing returns of including more stocks in the portfolio. As a reminder, our current portfolio of m=5 has an absolute deviation of 0.789 in 2019 and an absolute deviation of 1.112 in 2020. The following code snippets detail the process taken to create portfolios with different numbers of stocks included as well as an evaluation of the performance of the portfolio for 2019 and 2020 using the absolute deviation of returns between the portfolio and the index as the metric.

```
#Setting the number of stocks for the fund
no_of_selected_stocks = [10, 20, 30, 40 ,50, 60, 70, 80, 90, 100]

no_of_stocks=[]
abs_deviation_2019=[]
abs_deviation_2020=[]

for m in no_of_selected_stocks:

    #initializing the model in Gurobi
    stocks_model=gp.Model()

    #Creating the decision variables for the model.
    stocks_model_present_X = stocks_model.addMVar(len(stocks_available),vtype='B')

    best_stock_X = stocks_model.addMVar((len(stocks_available),len(stocks_available)), vtype='B')

    #Adding the objective function for the model
    stocks_model.setObjective(gp.quicksum(best_stock_X[i,j]*corr_matrix.iloc[i][j] for i in range(len(stocks_available))
                                          for j in range(len(stocks_available))),sense=gp.GRB.MAXIMIZE)

    #Adding the constraints for the model
    stocks_model_cons=[0]*3

    stocks_model_cons[0] = stocks_model.addConstr(gp.quicksum(stocks_model_present_X[i] for i in range(len(stocks_available))
                                                  == m)
    stocks_model_cons[1] = stocks_model.addConstrs(gp.quicksum(best_stock_X[i,j] for i in range(len(stocks_available)))== 1
                                                   for j in range(len(stocks_available)) )
    stocks_model_cons[2] = stocks_model.addConstrs(best_stock_X[i,j] <=stocks_model_present_X[i]
                                                   for j in range(len(stocks_available))
                                                   for i in range(len(stocks_available)))

    stocks_model.Params.OutputFlag = 0 # tell gurobi to shut up!!
    stocks_model.optimize() #Solving the model


    #show the selected stocks
    selected_stocks = stocks_available[np.where(stocks_model_present_X.x)]
    selected_stocks_position = np.where(stocks_model_present_X.x)

    #get the number of stock returns
    no_of_rows = len(stocks_2019)
    no_of_rows_2020 = len(stocks_2020)

    Index_column_name = stocks_2019.columns[1]
```

Code 5. *Selection of the stocks for each varying m using 2019 data.*

Essentially, the process of creating the models with the respective objective and constraints remains the same for stock selection as before and the portfolios are still constructed with 2019 data. However, since we have to create a model for each number of m, we have a for loop to increase the speed and reduce redundant code. The following code snippet is also located within the for loop for the same reasons.

```
#initializing the model in Gurobi
stocks_weight_model=gp.Model()

#Creating the decision variables for the model
stocks_return_difference_X = stocks_weight_model.addMVar(no_of_rows,vtype='C')

stocks_weight_X = stocks_weight_model.addMVar(m,vtype='C')

#Adding the objective function for the model
stocks_weight_model.setObjective(gp.quicksum(stocks_return_difference_X[i]
                                        for i in range(no_of_rows)),sense=gp.GRB.MINIMIZE)

#Adding the constraints for the model
stocks_weight_cons=[0]*3

stocks_weight_cons[0] = stocks_weight_model.addConstr(gp.quicksum(stocks_weight_X[i] for i in range(m)) == 1)
stocks_weight_cons[1] = stocks_weight_model.addConstrs(stocks_return_difference_X[i]>=
                                                stocks_2019[Index_column_name+"_return"][i] -
                                                gp.quicksum(stocks_weight_X[j]*
                                                            stocks_2019.iloc[i,selected_stocks_position[0][j]+2+
                                                                             len(stocks_available)]
                                                            for j in range(m)) for i in range(no_of_rows))
stocks_weight_cons[2] = stocks_weight_model.addConstrs(stocks_return_difference_X[i]>=
                                                - stocks_2019[Index_column_name+"_return"][i] +
                                                gp.quicksum(stocks_weight_X[j]*
                                                            stocks_2019.iloc[i,selected_stocks_position[0][j]+2+
                                                                             len(stocks_available)]
                                                            for j in range(m) ) for i in range(no_of_rows))


stocks_weight_model.Params.OutputFlag = 0 # tell gurobi to shut up!!
stocks_weight_model.optimize() #Solving the model

print("Optimization Status is :",stocks_weight_model.Status)

# Calculate the absolute deviation of the index fund
absolute_deviation_2019 = sum(abs(stocks_2019.iloc[i][Index_column_name+"_return"]
                                  - sum(stocks_weight_X.x[j] * stocks_2019.iloc[i][selected_stocks_position[0][j]+2+
                                        len(stocks_available)] for j in range(m)))
                              for i in range(no_of_rows))

print("Absolute Deviation for 2019: ", absolute_deviation_2019)

# Calculate the absolute deviation of the index fund from NASDAQ for 2020
absolute_deviation_2020 = sum(abs(stocks_2020.iloc[i][Index_column_name+"_return"]
                                  - sum(stocks_weight_X.x[j] * stocks_2020.iloc[i][selected_stocks_position[0][j]+2+
                                        len(stocks_available)] for j in range(m)))
                              for i in range(no_of_rows_2020))

print("Absolute Deviation for 2020: ", absolute_deviation_2020)

no_of_stocks.append(m)
abs_deviation_2019.append(absolute_deviation_2019)
abs_deviation_2020.append(absolute_deviation_2020)
```

Code 6. *Portfolio weight calculation of the stocks for each varying m and absolute deviation calculations for performance comparison of the portfolio to the index in years 2019 and 2020.*

After the stocks have been selected for each portfolio with a different m, we have to calculate the weights for each stock within each portfolio. To do this, we again followed much of the same process used to calculate the weights for the portfolio with m=5, where we aim to minimize the difference between the returns of the selected stocks and the index. Then, we calculated the absolute deviation between the returns of each portfolio and the returns of the index for 2019 and 2020. The resulting values are listed in the table below.

	no_of_stocks	abs_deviation_2019	abs_deviation_2020
0	10	0.701218	1.102404
1	20	0.478936	0.898582
2	30	0.418279	0.771210
3	40	0.367439	0.788335
4	50	0.332540	0.772100
5	60	0.346033	1.167648
6	70	0.233002	0.858428
7	80	0.148219	0.545540
8	90	0.053779	0.367790
9	100	0.044911	0.368682

Table 4. *Absolute deviation of portfolio fund with NASDAQ for 2019 and 2020 for various values of m using the first method.*

The portfolio with m=100 had the best performance (lowest deviation) at an absolute deviation of 0.045 when tracking the NASDAQ in 2019. The portfolio with m=90 (followed closely by m=100) had the best performance at an absolute value of 0.368 when tracking the NASDAQ in 2020. This makes sense given that all stocks that are present within the index are included within the portfolio. The only difference in returns stems from the fact that the calculated portfolio weights may be different from the weights of the stocks within the index. We have also plotted the data in the table below to better identify differences in performance between 2019 and 2020.

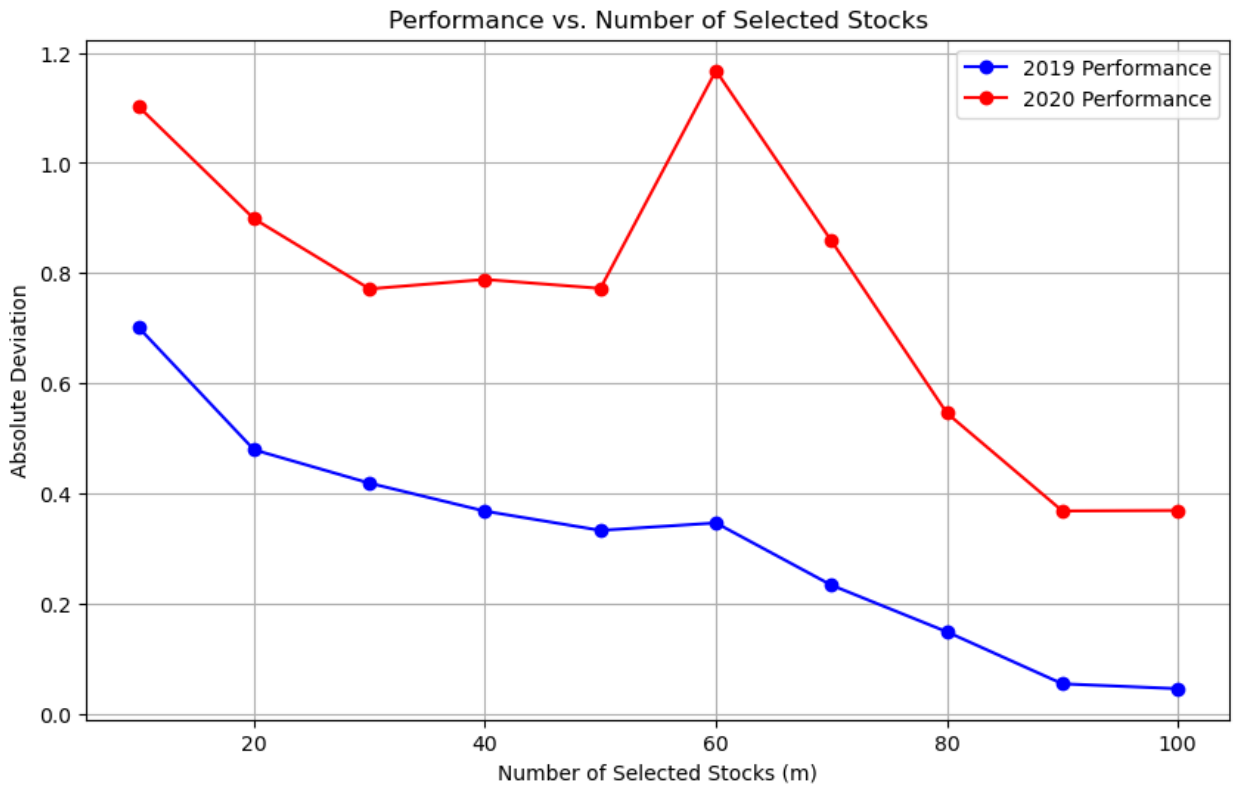

Figure 4. *Line graph comparing the performance of the portfolios in 2019 and 2020 for varying numbers of selected stocks.*

At a quick glance, we can see that when the performance of the portfolios is evaluated in 2019 versus 2020, portfolios evaluated in 2019 consistently perform better (have lesser deviations) than portfolios evaluated in 2020. This is likely due to the fact that the portfolios are constructed using 2019 data, which would lead to fewer deviations and hence better performance. Additionally, we also see that there is a downward trend in the absolute deviations for both 2019 and 2020 data, with an exception at m=60.

Our analysis of the 2020 data reveals some interesting trends. Notably, there is a pronounced 49.23% increase in deviation when transitioning from 30 to 60 stocks. This increase signifies that, in this specific case, a smaller selection of stocks (30) is more favorable. What's particularly intriguing is the 41% spike in deviation from 50 to 60 stocks, reinforcing the appeal of the 30-stock option. This preference is substantiated further by examining the 2019 data, where the deviation of the 60-stock model is only 17.49% lower than that of the 30-stock model.

Another viable alternative to consider is the 70-stock fund, which consistently exhibits a very similar deviation to the 80-stock option. Furthermore, the 90-stock model delivers the lowest deviation for the year 2020, suggesting that this configuration may merit special attention.

In summary, our rigorous analysis of different 'm' values underscores the nuanced relationship between the number of selected stocks and performance. The spike observed at 'm=60' highlights a pivotal point of diminishing returns, emphasizing the significance of optimizing the selection to ensure the best outcomes for our fund.

Reformulating Weight Selection: Mixed Integer Programming (MIP)

The goal of switching to an MIP optimization problem from the linear one is to see if the np hard computational complexity is worth it for more precise results. The MIP objective function does not differ much from the original objective function. The core objective of the weight selection model is to minimize the discrepancies between the fund return and index return over a given time. This is accomplished by determining the weight for each component stock, denoted by "wi", to minimize the equation below:

$$\min_{w} \sum_{t=1}^{T} |q_t - \sum_{i=1}^{n} w_i r_{it}|$$

Where:

- qt indicates the return of the index on day "t".
- rit signifies the return of stock "i" on day "t".
- n represents the total number of stocks available in the system. This is the main difference between the two objective functions.
- T is the total count of days in the considered period.
- wi: Weights of stock “i” chosen for the portfolio.

In the MIP model, the binary decision variables, denoted as Yj, ensure that only a subset of the total stocks are selected for the portfolio. There are 100 Yj variables, one for each potential stock to be included in the portfolio. Furthermore, the Xij variables, which are restricted continuous variables dependent on their binary counterparts using the "Big M" method, account for the weights of the stocks. There are 10,000 of these variables, capturing the intricate relationships between stocks and their respective weights in the portfolio.

The decision variables are as follows:

- D: Continuous variable representing the difference between the portfolio and the entire index for a particular day
- Xij: Continuous variable representing the relationship between stock “i” and its weight “j” in the portfolio.

- Yj: Binary variable where Yj = 1 if stock “j” is selected for the portfolio and Yj = 0 otherwise.

The constraints are as follows:

1. $$\sum_{j=1}^{n} Y_j = m$$

   This guarantees that 'm' stocks, out of the possible 100, are chosen for the portfolio.

2. $$X_{ij} \leq M \times Y_j$$

   With "M" being a sufficiently large number, this constraint connects the binary decision variables Yj to the continuous variables Xij. It ensures a stock has a weight in the portfolio only if it's selected (i.e., if its corresponding Yj is 1).

3. $$\sum_{j=1}^{n} X_{ij} = 1$$

   This ensures that the portfolio weights sum up to 1.

4. $$X_{ij} \geq 0$$

   This ensures non-negative portfolio weights.

5. $$D >= q_t - \sum_{i=1}^{n} w_i r_{it}$$
   $$D >= \sum_{i=1}^{n} w_i r_{it} - q_t$$

   These constraints remove the nonlinearity in the problem caused by using an absolute value.

By employing the updated MIP framework, the optimization problem endeavors to find the optimal weights of a predefined number 'm' of stocks from the available 'N' stocks. The intent is to minimize the discrepancy between the fund's and the index's returns over time. This model offers precise control over the stocks' count in the portfolio and ensures realistic stock weights, though at the cost of greater computational complexity. Below, we have included a code snippet

showing the Gurobi model that includes this new method that utilizes MIP and the necessary constraints to identify the new weights and selected stocks.

```
#get the number of stock returns
m=5

no_of_rows = len(stocks_2019)
no_of_rows_2020 = len(stocks_2020)

Index_column_name = stocks_2019.columns[1]

#initializing the model in Gurobi
stocks_weight_model=gp.Model()

stocks_weight_model.Params.TimeLimit = gurobi_time_limit

#Creating the decision variables for the model
stocks_return_difference_X = stocks_weight_model.addMVar(no_of_rows,vtype='C')

stocks_weight_X = stocks_weight_model.addMVar(len(stocks_available),vtype='C')

stocks_presence_X = stocks_weight_model.addMVar(len(stocks_available),vtype='B')

#Adding the objective function for the model
stocks_weight_model.setObjective(gp.quicksum(stocks_return_difference_X[i] for i in range(no_of_rows)),sense=gp.GRB.MINIMIZE)

#Adding the constraints for the model
stocks_weight_cons=[0]*5

M = 2*len(stocks_available)

stocks_weight_cons[0] = stocks_weight_model.addConstr(gp.quicksum(stocks_presence_X[i]
                                                    for i in range(len(stocks_available))) == m)
stocks_weight_cons[1] = stocks_weight_model.addConstrs(stocks_return_difference_X[i]>=
                                                stocks_2019[Index_column_name+"_return"][i] -
                                                gp.quicksum(stocks_weight_X[j]*stocks_2019.iloc[i,j+ 2+
                                                                                    len(stocks_available)]
                                                            for j in range(len(stocks_available)) )
                                                for i in range(no_of_rows))
stocks_weight_cons[2] = stocks_weight_model.addConstrs(stocks_return_difference_X[i]>=
                                                - stocks_2019[Index_column_name+"_return"][i] +
                                                gp.quicksum(stocks_weight_X[j]*stocks_2019.iloc[i,j+ 2+
                                                                                    len(stocks_available)]
                                                            for j in range(len(stocks_available)) )
                                                for i in range(no_of_rows))
stocks_weight_cons[3] = stocks_weight_model.addConstrs((stocks_weight_X[i] <=M*stocks_presence_X[i])
                                                for i in range(len(stocks_available)) )
stocks_weight_cons[4] = stocks_weight_model.addConstr(gp.quicksum(stocks_weight_X[i] for i in range(len(stocks_available)))
                                                == 1)

stocks_weight_model.Params.OutputFlag = 0 # tell gurobi to shut up!!
stocks_weight_model.optimize() #Solving the model
```

Code 7. *Method 2 of calculating portfolio weights using MIP and ignoring stock selection.*

First, we initialized the model in Gurobi with our decision variables and objective function, which was to minimize the returns between the m-stock and the entire index. Then, we added the key constraints discussed mathematically above. Finally, we ran the model to find our selected stocks when m=5, where m is the number of stocks in our portfolio.

Following our model run with 'm = 5,' we've identified a compelling selection of stocks that have significant relevance within the index. These five stocks are AMZN (Amazon.com Inc), ADI (Analog Devices Inc), AAPL (Apple Inc), MSFT (Microsoft Corporation), and MDLZ (Mondelez

International Inc). These five chosen stocks span a diverse spectrum of industries and companies, each bringing distinctive attributes to our portfolio.

Here is a detailed overview of the five selected stocks, providing insights into the respective companies and the industries they represent:

Selected Stocks	*Company*	*Industry*
AMZN	Amazon.com Inc	E-Commerce and Cloud Computing
ADI	Analog Devices Inc	Semiconductor Manufacturing
AAPL	Apple Inc	Technology and Consumer Electronics
MSFT	Microsoft Corporation	Information Technology
MDLZ	Mondelez International Inc	Food and Snack Manufacturing

Table 5. *The best 5 stocks selected, their company name, and corresponding industry using MIP.*

These selected stocks collectively encapsulate a rich tapestry of industries, from e-commerce and technology to consumer electronics and snack manufacturing. Such diversity in sector representation forms the cornerstone of a well-balanced portfolio.

Our analysis involved the initial exploration of the 2019 dataset, during which our model pinpointed these stocks for inclusion. Subsequently, leveraging linear programming, we precisely determined the weight allocation for each of the selected stocks. The resulting weight distribution for these stocks is as follows:

Selected Stocks	*Weights*
AMZN	0.25
ADI	0.12
AAPL	0.19
MSFT	0.28
MDLZ	0.16

Table 6. *The selected stocks and their associated weights in the portfolio using MIP.*

It's noteworthy that within the NASDAQ-100 index, technology behemoths like Amazon, Apple, and Microsoft hold considerable weight. Our strategic decision to assign significant weight to these stocks within our portfolio aligns it closely with the composition of the NASDAQ-100 index. By mirroring the weightings of key constituents such as Amazon, Apple, and Microsoft, our portfolio closely mirrors the performance and attributes of the NASDAQ-100, making it an effective strategy for replicating the index's behavior. To better visualize the weight distribution

of stocks within the portfolio, we have created the bar graph of the stocks and their respective weights below.

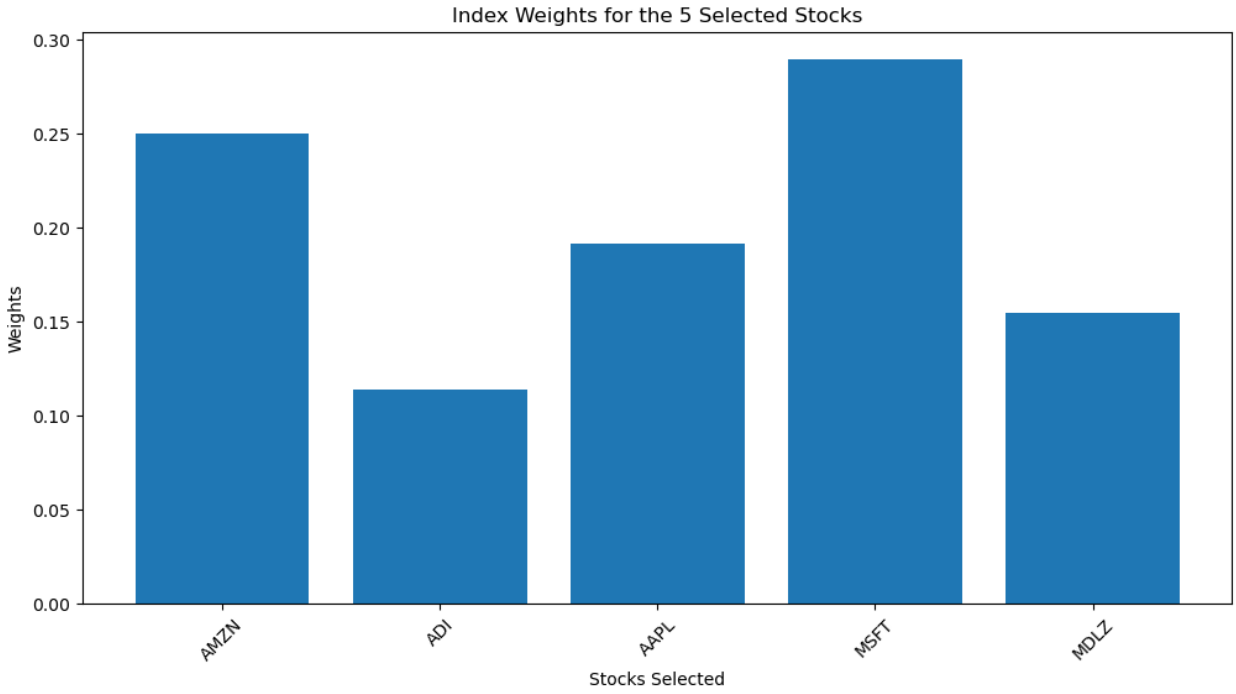

Figure 5. *Selected stocks and associated weights for m=5 portfolio using MIP method.*

To allow for a better evaluation of the portfolio's performance, we created the line graph below that displays the performance of our portfolio fund against the NASDAQ index in 2020. The line graph visually depicts the deviations in return performance between the index and our portfolio, just as we did for the previous method.

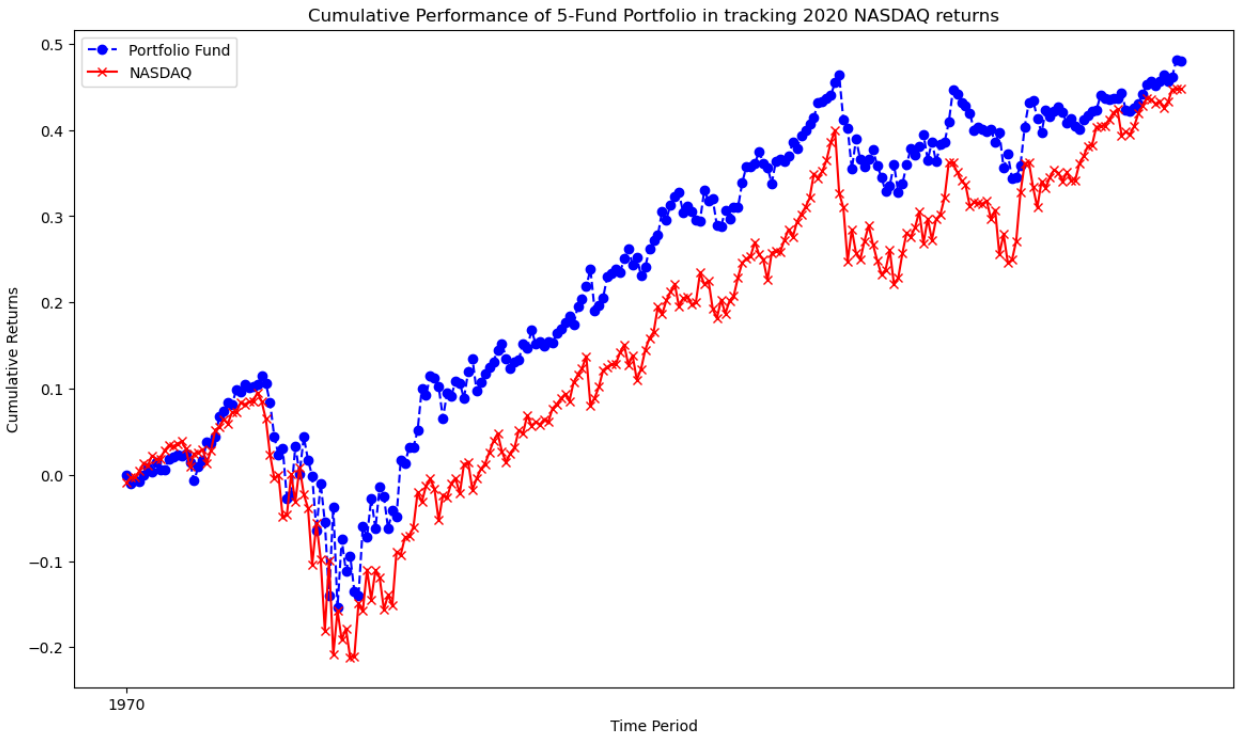

Figure 6. *Line graph depicting the cumulative performance of our 5-stock portfolio in tracking 2020 NASDAQ returns using MIP method 2.*

In conclusion, this thoughtfully selected group of stocks, along with their strategic weight distribution, positions our portfolio to closely mimic the NASDAQ-100's performance, providing us with an optimal foundation for replicating the index's behavior and characteristics.

Since we want to compare how this model performs across different ‘m’ values, we followed a similar process to the previous method to evaluate the performance of the model across varying m. The code snippet to create portfolios at different m values is shown below.

```
#Setting the number of stocks for the fund
no_of_selected_stocks = [5, 10, 20, 30, 40 ,50, 60, 70, 80, 90, 100]

no_of_stocks=[]
abs_deviation_2019=[]
abs_deviation_2020=[]

for m in no_of_selected_stocks:
    #get the number of stock returns
    no_of_rows = len(stocks_2019)
    no_of_rows_2020 = len(stocks_2020)

    Index_column_name = stocks_2019.columns[1]

    #initializing the model in Gurobi
    stocks_weight_model=gp.Model()

    stocks_weight_model.Params.TimeLimit = gurobi_time_limit

    #Creating the decision variables for the model
    stocks_return_difference_X = stocks_weight_model.addMVar(no_of_rows,vtype='C')

    stocks_weight_X = stocks_weight_model.addMVar(len(stocks_available),vtype='C')

    stocks_presence_X = stocks_weight_model.addMVar(len(stocks_available),vtype='B')

    #Adding the objective function for the model
    stocks_weight_model.setObjective(gp.quicksum(stocks_return_difference_X[i] for i in range(no_of_rows)),
                                     sense=gp.GRB.MINIMIZE)

    #Adding the constraints for the model
    stocks_weight_cons=[0]*5

    M = 2*len(stocks_available)

    stocks_weight_cons[0] = stocks_weight_model.addConstr(gp.quicksum(stocks_presence_X[i]
                                                                     for i in range(len(stocks_available))) == m)
    stocks_weight_cons[1] = stocks_weight_model.addConstrs(stocks_return_difference_X[i]>=
                                                           stocks_2019[Index_column_name+"_return"][i] -
                                                           gp.quicksum(stocks_weight_X[j]*stocks_2019.iloc[i,j+ 2+
                                                                       len(stocks_available)]
                                                                       for j in range(len(stocks_available)) )
                                                           for i in range(no_of_rows))
    stocks_weight_cons[2] = stocks_weight_model.addConstrs(stocks_return_difference_X[i]>=
                                                           - stocks_2019[Index_column_name+"_return"][i] +
                                                           gp.quicksum(stocks_weight_X[j]*stocks_2019.iloc[i,j+ 2+
                                                                       len(stocks_available)]
                                                                       for j in range(len(stocks_available)) )
                                                           for i in range(no_of_rows))
    stocks_weight_cons[3] = stocks_weight_model.addConstrs((stocks_weight_X[i] <=M*stocks_presence_X[i])
                                                           for i in range(len(stocks_available)) )
    stocks_weight_cons[4] = stocks_weight_model.addConstr(gp.quicksum(stocks_weight_X[i]
                                                                     for i in range(len(stocks_available))) == 1)

    stocks_weight_model.Params.OutputFlag = 0 # tell gurobi to shut up!!
    stocks_weight_model.optimize() #Solving the model
```

Code 8. *Weight calculation and stock selection for varying m values using the MIP method.*

Using a for loop, we create models to calculate the weights for each of the stocks in the portfolios. The for loop allows us to reduce redundancy within the code. The objective, constraints, and inputs for the Gurobi model remain the same as the previous model we ran for m=5 and are placed inside of the for loop.

We also calculated absolute deviation values between the returns of each portfolio with a varying m and the index. To understand how the performance of the model changes across different m values and the different years (2019 and 2020), we have visually depicted the absolute deviations in returns between each m-portfolio and the NASDAQ in the graph below.

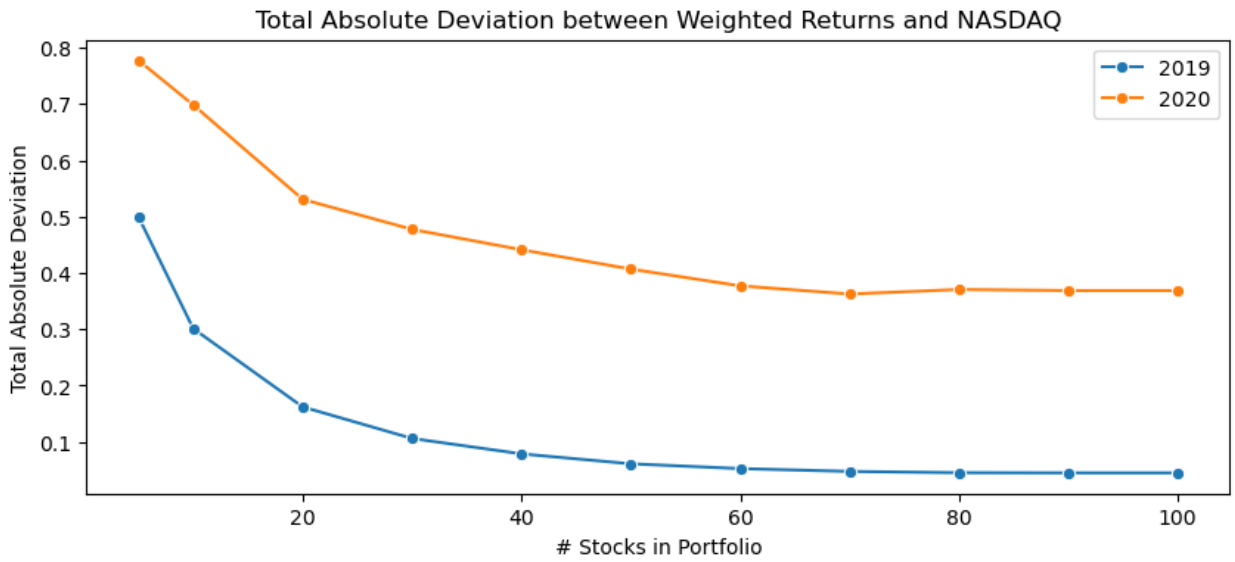

Figure 7. *Line graph comparing the performance of the portfolios in 2019 and 2020 for varying numbers of selected stocks using method 2.*

The second method (MIP) shows that as we increase the number of stocks in the fund (m), performance stabilizes and adding more stocks doesn't significantly improve performance. This behavior is in contrast to the first model, where we observed performance improvements with the addition of more stocks.

Furthermore, the second method results in smaller deviations compared to the first method. Therefore, the second method consistently delivers better and more stable performance. However, it's worth noting that the second method is computationally more intensive.

Around m=50, we observe an optimal selection of stocks, where performance stabilizes. This point represents an "elbow" in the 2020 performance graph. If we prefer a smaller number of stocks in the fund, 30 stocks appear to be a suitable choice, providing a balance between performance and simplicity.

In summary, the second method's performance stability and lower deviation make it a preferable choice, and the choice of the optimal number of stocks depends on specific preferences and trade-offs.

Recommendations & Conclusions

After evaluating the performance of several different portfolios created with different methods or selecting a different number of stocks to include, we have come up with a few recommendations and conclusions on the basis of the results. Our primary recommendation is

to use the Mixed Integer Programming (MIP) method when trying the maximize cumulative returns. When we compare the cumulative returns for the Integer Programming/Linear Programming (IP/LP) and MIP methods, we observe that the MIP method closely tracks the index for a wide range of selected stocks. This trend is supported by the total absolute deviation plots for both 2019 and 2020, where the MIP method's curves remain relatively smooth over the range of 'm' values. In contrast, the IP/LP method exhibits jagged deviations, and it only reaches its lowest deviation after selecting over 80 stocks. The MIP method, on the other hand, begins to level out somewhere between 40 and 60 selected stocks. This superior accuracy comes at the cost of increased computational time, making MIP more time-consuming than IP/LP.

Now that we've recommended the MIP method for selecting weights, let's determine the optimal number of stocks to select using MIP. Analyzing the total absolute deviation for the MIP method, we find that the deviation from the index stabilizes between 40 and 60 selected stocks. Beyond this range, there's little additional benefit gained from adding more stocks to the portfolio. Therefore, our recommendation is to select at most 60 stocks in the MIP model to closely replicate the index with the least computational effort.

In evaluating the two approaches, Method 2 consistently outperforms Method 1 across different values of 'm', delivering significantly reduced errors. The performance of Method 2 appears to peak around m = 50 and then stabilizes, while Method 1 exhibits higher variability and continues to improve as 'm' increases. Hence, we strongly endorse Method 2 for constructing our Index fund to replicate the NASDAQ-100.

Another recommendation we have is to set 'm' at approximately 30, where we observe an inflection or "elbow" in the performance graph. This number of stocks in our fund ensures a reasonable replication of the NASDAQ-100. If the costs associated with adding more stocks to our portfolio, including rebalancing costs and price response to trading, are manageable, we can consider increasing 'm'. The precise value of 'm' should be refined by weighing the trade-offs between portfolio expansion costs and the costs associated with tracking error when mirroring the NASDAQ-100.

It's important to note that Method 2 requires significantly more computational resources and time. Method 1 should be considered only when there are insurmountable resource constraints for constructing our Index fund. To assist your decision-making, we've provided visualizations that illustrate the comparative performance of both methods across different 'm' values. This data-driven analysis empowers you to make an informed choice aligned with our fund's objectives and resource constraints.