Project 2 Optimization Report: Equity Money Management Strategies Using Integer Programming

Submitted to:

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

Prepared by:

Aditya Chawla (ac86589)
Biyun Yuan (by3764)
Ian McIntosh (iwm243)
Mayank Gupta (mg66426)

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

Fall 2023

1. Background

The goal of passive equity money management strategies is to create a portfolio that will mirror the market population or a certain market index. This type of portfolio is known as an index fund. Purchasing all the stocks in an index with the same weights is impractical, therefore our goal is to design an index fund with a number of stocks that is substantially smaller than the number of stocks in the index. Our goal is to create an integer program to decide which stocks from the index we should buy, then create a linear program to decide how much to buy of each stock we choose. The integer programming will use a similarity matrix as its input, that tells us the similarity between any two stocks in the index.

2. Stock Selection and Constraints

We will select our stocks by looking at Set I which contains all the stocks in the index, with set F being the stocks in our fund. We will create a link in order to map elements of set I to set F. Our first constraint will be the amount of stocks we want to hold in the fund, which we will set equal to m. We can not map more than m elements to set F from set I. Our second constraint will impose that each stock i in the index has exactly one stock j that best represents it within the index.

This means that each element in set I must map to one element in set F. Our third constraint states that stock i will be best represented by stock j if and only if stock j is in our fund. This suggests that an element of set F must be represented in the fund if an element from set I gets mapped to it. From here we will maximize the similarity between the stocks in the index and the representatives of these stocks that we choose for our fund. This will give us the best mapping from the index to our fund.

3. Reading Datasets and Calculating Correlation Matrix

In order to calculate the correlation matrix, we first had to look at the prices of the component stocks of the NASDAQ 100 in 2019 and 2020. The data sets we were provided with contained daily price data for 100 of the stocks within the index. From here we were able to calculate daily index returns across both data sets. Using the daily index returns we calculated the correlation matrices for both data sets based on the percent changes. The code we ran to do this is shown below.

```
#storing daily index values in both datasets
index_2019 = data_2019[['X','NDX']]
index_2020 = data_2020[['X','NDX']]

#Storing daily index returns in both datasets
q_2019 = data_2019['NDX'].pct_change().reset_index().set_index('index')
q_2020 = data_2020['NDX'].pct_change().reset_index().set_index('index')

#dropping index column from both the datasets
data_2019.drop('NDX',axis=1,inplace=True)
data_2020.drop('NDX',axis=1,inplace=True)


#Calculating corr matrices for both dataframes

#2019
data_2019.set_index('X', inplace=True)
data_2019_pct = data_2019.pct_change()
data_corr_2019 = data_2019_pct.corr()

#2020
data_2020.set_index('X', inplace=True)
data_2020_pct = data_2020.pct_change()
data_corr_2020 = data_2020_pct.corr()

print(data_corr_2019.head())
print(data_corr_2020.head())
```

Figure 1. Code used to calculate correlation matrix.

```
          ATVI      ADBE       AMD      ALXN      ALGN     GOOGL      GOOG
ATVI  1.000000  0.399939  0.365376  0.223162  0.216280  0.433097  0.426777  \
ADBE  0.399939  1.000000  0.452848  0.368928  0.363370  0.552125  0.540404
AMD   0.365376  0.452848  1.000000  0.301831  0.344252  0.418861  0.417254
ALXN  0.223162  0.368928  0.301831  1.000000  0.332433  0.315993  0.307698
ALGN  0.216280  0.363370  0.344252  0.332433  1.000000  0.248747  0.250316

          AMZN      AMGN       ADI  ...      TCOM      ULTA      VRSN
ATVI  0.467076  0.203956  0.329355  ...  0.322906  0.128241  0.464850  \
ADBE  0.598237  0.291978  0.473815  ...  0.360392  0.201151  0.711339
AMD   0.549302  0.151452  0.503733  ...  0.332776  0.210623  0.498342
ALXN  0.363170  0.342022  0.317040  ...  0.257143  0.408936  0.350581
ALGN  0.399281  0.264599  0.328280  ...  0.175957  0.128559  0.360886

          VRSK      VRTX       WBA      WDAY       WDC       XEL      XLNX
ATVI  0.316549  0.259679  0.218149  0.311659  0.303077  0.043389  0.249667
ADBE  0.541243  0.402171  0.228106  0.650430  0.361516  0.207403  0.289497
AMD   0.330900  0.272983  0.281950  0.407626  0.438892  0.017283  0.478010
ALXN  0.191489  0.522423  0.192720  0.416396  0.289908  0.047947  0.200356
ALGN  0.251855  0.334978  0.219595  0.308968  0.284407  0.088059  0.253934

[5 rows x 100 columns]
          ATVI      ADBE       AMD      ALXN      ALGN     GOOGL      GOOG
ATVI  1.000000  0.730174  0.579314  0.403073  0.237541  0.567828  0.566060  \
ADBE  0.730174  1.000000  0.659260  0.491232  0.437418  0.798263  0.795778
...
ALXN  0.473451  0.457318  0.330271  0.353305  0.348578  0.373230  0.475345
ALGN  0.506896  0.307963  0.372854  0.397577  0.543233  0.464390  0.469649

[5 rows x 100 columns]
```

Figure 2. Snapshot of results of correlation matrix.

4. Approach 1

4.1 Choosing the 5 Best Stocks and Calculating Their Weights

After we calculated the correlation matrix, the next step was to start with m = 5 and find the 5 best stocks to include in the portfolio, along with their weights using the 2019 data. To do so, we used gurobi to create and solve a linear programming problem. The decision variables we created included 100x100 binary variables to indicate whether there is a link between element i in Set I and element j in Set F. We also included 100 decision variables for the stocks we choose. After this, we added in our objective function and constraints and had gurobi optimize our stock selections. The objective function and code we used are shown below.

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

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

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

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

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

Figure 3: Objective function used to decide which 5 stocks to buy.

```
#stock selection
m=5
model = gp.Model()

#Creating 100*100 xij decision variables for
xij = model.addMVar((records_2019,records_2019),vtype = 'b')
y = model.addMVar(records_2019,vtype = 'b')

#objective for choosing best stocks (maximizing sum of correlation)
model.setObjective(gp.quicksum(xij[i,j]*data_corr_2019.iloc[i,j] for j in range(records_2019) for i in range(records_2019)), sense = gp.GRB.MAXIMIZE)

#constraint 1 (ensuring only one xij in each column is chosen)
model.addConstrs((gp.quicksum(xij[i,j] for j in range(records_2019)) == 1) for i in range(records_2019))

#constraint 2 (Ensuring that only 'm' stocks are picked)
model.addConstr(gp.quicksum(y[i] for i in range(records_2019)) == m)

#constraint 3 (logical constraint)
model.addConstrs(xij[i,j] <= y[j] for j in range(records_2019) for i in range(records_2019))

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

Figure 4. Code used to select 5 stocks.

The 5 stocks the model told us to select were *LBTYK, MXIM, MSFT, VRTX and XEL*. These stocks yielded an objective value of about 54.84.

From here we needed to calculate the optimal weights of the 5 stocks. We did this using another linear programming problem in gurobi. In order to do so we had to solve the following problem:

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

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

$$w_i \geq 0.$$

Figure 5: Optimization problem to be solved to calculate portfolio weights

Since the absolute value in this objective function creates non-linear constraints, we had to find a way to convert them into linear constraints in order to solve the problem. The methodology we used to do so is shown below.

Suppose I want to solve

$$\min_{x} \quad |x-1| + |x-2| + |x-3|$$

It should be apparent that the answer to this problem is x=2. However, this is a non-linear program; here's how you could formulate it as an LP.

$$\min_{x,y_1,y_2,y_3} \quad y_1 + y_2 + y_3$$

$$s.t. \quad y_1 \geq x - 1$$

$$y_1 \geq -(x-1)$$

$$y_2 \geq x - 2$$

$$y_2 \geq 2 - x$$

$$y_3 \geq x - 3$$

$$y_3 \geq 3 - x.$$

Figure 6: Methodology used to convert non linear constraints into linear constraints.

Finally we were able to calculate the portfolio weights. The figures below will show the code we used to do this as well as the results we obtained.

```
#weights selection
model= gp.Model()
w=model.addMVar(m, vtype='C')
ys= model.addMVar(T, vtype='C')

#setting objective
model.setObjective(gp.quicksum(ys[i] for i in range(T)), sense=gp.GRB.MINIMIZE)

#adding constraints to convert non-linear absolute objective to linear
model.addConstrs((q_2019[t]-gp.quicksum(w[i]*rit_2019[t,i] for i in range(m))) <= ys[t]for t in range(T))
model.addConstrs((gp.quicksum(w[i]*rit_2019[t,i] for i in range(m))-q_2019[t]) <= ys[t] for t in range(T))


model.addConstr(gp.quicksum(w[i] for i in range(m))==1)

model.Params.OutputFlag = 0
model.optimize()
```

Figure 7: Code used to calculate portfolio weights

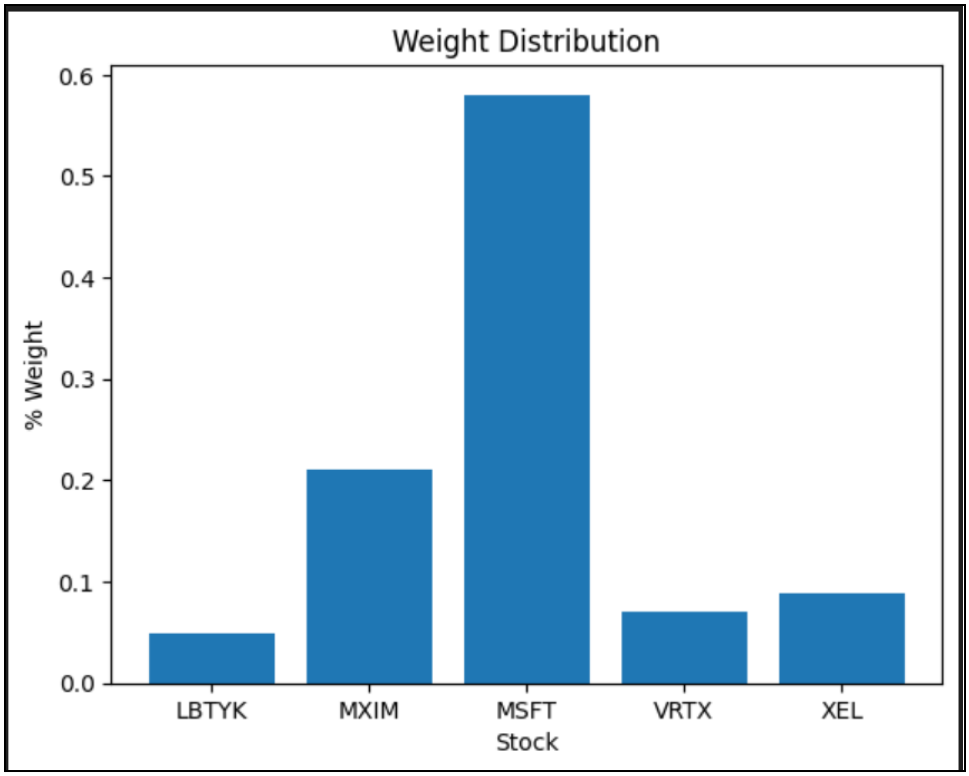

Figure 8: Portfolio weights

The optimal weights for each selected stock is as follows:

Selected Stocks	Weights
LBTYK	0.05
MXIM	0.21
MSFT	0.58
VRT	0.07
XEL	0.09

We wanted to see how these stocks and weights would perform in 2020 given that they were calculated in 2019. After evaluating the performance we plotted the daily return comparison graph of the index and our funds, and then computed the weighted returns in 2020 for the selected stocks and the NASDAQ. The code we used to do so and the results we obtained are shown below.

```
#evaluating performance on 2020 data using above weights
rit_2020 = data_2020_pct[selected_stocks].values
abs_dev_2020 = 0

for i in range(len(data_2020_pct)):
    wr = 0
    for j in range(len(selected_stocks)):
        wr = wr + (w.x[j]*rit_2020[i][j])
    s = abs(q_2020[i] - wr)
    abs_dev_2020 = abs_dev_2020 + s

print ("\nAbsolute Deviation for year 2020 is:", abs_dev_2020)
```

```
#plotting daily return comparison graph of index and our funds
returns_timeline = data_2020_pct.index
weighted_returns = []

#Computing weighted returns
for i in range(len(data_2020_pct)):
    wr = 0
    for j in range(len(selected_stocks)):
        wr = wr + (w.x[j]*rit_2020[i][j])
    weighted_returns.append(wr)

plt.figure(figsize = (14,8))

#Plotting weighted returns from 2020 for selected stocks
sns.lineplot(x = returns_timeline, y = np.array(weighted_returns), marker = 'o',
             linestyle = 'dashed', color = 'green', label = 'Portfolio Fund')

#Plotting returns from 2020 for NASDAQ
sns.lineplot(x = returns_timeline, y = q_2020.ravel(), marker = 'x',
             color = 'blue', label = 'NASDAQ')

plt.title("Performance of 5-Fund Portfolio in tracking 2020 NASDAQ returns")
plt.xlabel('Time Period')
plt.ylabel('Returns')
```

Figures 9 and 10: Code used to evaluate the performance of our 5-fund portfolio compared to NASDAQ returns

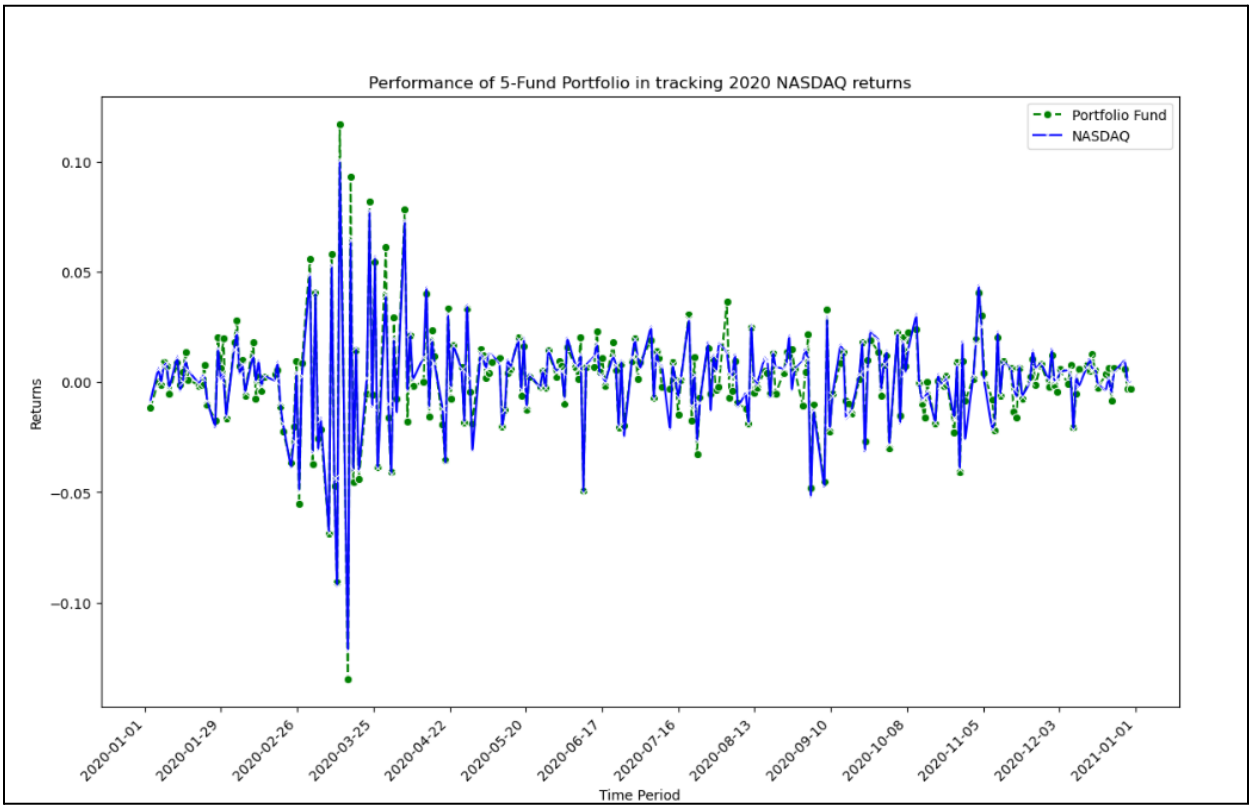

Figure 11: Performance of 5-fund portfolio compared to 2020 NASDAQ returns.

4.2 Iterating over Different Numbers of Stocks (m) and Evaluating Performance

Since 5 was an arbitrary number of stocks to choose, we wanted to vary the value of m and analyze the performance of the fund based on these values. In this section, we delve into a meticulous evaluation of the portfolio's performance as we alter the number of stocks it comprises. This analysis involves:

- Varying the Portfolio Size: We incrementally expand the portfolio, assessing configurations containing 10 to 100 stocks (in steps of 10).
- Model Initialization & Constraints: With each portfolio expansion, we adapt our linear optimization model and its constraints to accommodate the new portfolio size, maintaining our goal to maximize stock-representative similarity.
- Optimizing Stock Weights for 2019: We utilize a secondary optimization process to determine the optimal stock allocation within each portfolio configuration, aiming to minimize the variance between our portfolio returns and the index returns for 2019.
- Evaluating Performance for 2020: Using the 2019-established weights, we assess the 2020 portfolio performance by measuring its tracking error relative to the index.

Below is the code the team implemented to iterate through different values of *m*.

```
perf_2019 = []
perf_2020 = []

no_of_stocks = [5,10,20,30,40,50,60,70,80,90,100]

for m in no_of_stocks:
    if m <= data_2019.shape[1]:



        print(f'For m = {m}')

        #stock selection
        model = gp.Model()

        #Creating 100*100 xij decision variables for
        xij = model.addMVar((records_2019,records_2019),vtype = 'b')
        y = model.addMVar(records_2019,vtype = 'b')

        #objective for choosing best stocks (maximizing sum of correlation)
        model.setObjective(gp.quicksum(xij[i,j]*data_corr_2019.iloc[i,j] for j in range(records_2019) for i in range(records_2019)), sense = gp.GRB.MAXIMIZE)

        #constraint 1 (ensuring only one xij in each column is chosen)
        model.addConstrs((gp.quicksum(xij[i,j] for j in range(records_2019)) == 1) for i in range(records_2019))

        #constraint 2 (Ensuring that only 'm' stocks are picked)
        model.addConstr(gp.quicksum(y[i] for i in range(records_2019)) == m)

        #constraint 3 (logical constraint)
        model.addConstrs(xij[i,j] <= y[j] for j in range(records_2019) for i in range(records_2019))

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

        stocks = data_2019_pct.columns.tolist()
        selected_stocks = [stocks[i] for i in range(records_2019) if y.x[i] == 1]
        print(f'Selected stocks are:', selected_stocks)

        rit_2019 = data_2019_pct[selected_stocks].values
        rit_2020 = data_2020_pct[selected_stocks].values
        T = len(q_2019)

        #weights selection
        model= gp.Model()
        w=model.addMVar(m, vtype='C')
        ys= model.addMVar(T, vtype='C')

        #setting objective
        model.setObjective(gp.quicksum(ys[i] for i in range(T)), sense=gp.GRB.MINIMIZE)

        #adding constraints to convert non-linear absolute objective to linear
        model.addConstrs((q_2019[t]-gp.quicksum(w[i]*rit_2019[t,i] for i in range(m))) <= ys[t]for t in range(T))
        model.addConstrs((gp.quicksum(w[i]*rit_2019[t,i] for i in range(m))-q_2019[t]) <= ys[t] for t in range(T))


        model.addConstr(gp.quicksum(w[i] for i in range(m))==1)

        model.Params.OutputFlag = 0
        model.optimize()

        print(f'Weights of selected stocks are: {w.x}')

        #evaluating performance on 2019 data using above weights
        abs_dev_2019 = 0

        for i in range(len(data_2019_pct)):
            wr = 0
            for j in range(len(selected_stocks)):
                wr = wr + (w.x[j]*rit_2019[i][j])
            s = abs(q_2019[i] - wr)
            abs_dev_2019 = abs_dev_2019 + s

        print ("\nAbsolute Deviation for year 2019 is:", abs_dev_2019)
        perf_2019.append(abs_dev_2019)

        #evaluating performance on 2020 data using above weights
        abs_dev_2020 = 0

        for i in range(len(data_2020_pct)):
            wr = 0
            for j in range(len(selected_stocks)):
                wr = wr + (w.x[j]*rit_2020[i][j])
            s = abs(q_2020[i] - wr)
            abs_dev_2020 = abs_dev_2020 + s

        print ("\nAbsolute Deviation for year 2020 is:", abs_dev_2020)
        perf_2020.append(abs_dev_2020)

    else:
        print("No of stocks chosen exceeded the maximum available stocks!!")
```

Figure 12: Code used to run the model with different values of m and evaluating performance using absolute deviations.

Beginning with a base of 10 stocks, we incrementally increased our portfolio in multiples of 10, culminating at 100 stocks. Each increment was an opportunity to gauge the sensitivity of the portfolio's performance to diversification, providing a granular view of the optimal portfolio composition.

Each 'm' increment necessitated a recalibration of our model constraints to accommodate the broader portfolio while steadfastly pursuing our goal: maximizing alignment with the benchmark. This adaptive approach ensured our model's sustained relevance and accuracy.

Since the output contained a lot of information we created a dataframe to store total absolute deviations with NASDAQ for various values of m. The code used to create the dataframe and the resulting data frame are shown below.

```
#Creating a dataframe to store Total Absolute Deviations with NASDAQ for different portfolios for both 2019 and 2020
summary_df = pd.DataFrame(data=[np.append(np.array([5]),np.array(range(10,110,10))),perf_2019, perf_2020]).T
summary_df.rename(columns={0: 'm', 1: 'Performance on 2019 Data', 2: 'Performance on 2020 Data'}, inplace=True)
summary_df = summary_df.set_index('m')
print("Absolute Deviation of Portfolio Fund with NASDAQ for 2019 and 2020 for various values of m:\n")

summary_df.dropna(axis=0,how='all',inplace=True) #removing any NaN rows in case no. of stocks in file is <100
summary_df
```

```
Absolute Deviation of Portfolio Fund with NASDAQ for 2019 and 2020 for various values of m:
```

m	Performance on 2019 Data	Performance on 2020 Data
5	[0.7891782824631473]	[1.1124373455076468]
10	[0.7012177959266302]	[1.1024044007151048]
20	[0.4662681416517702]	[0.8554460015414783]
30	[0.4097923638636839]	[0.7673668495630364]
40	[0.3632810268101053]	[0.7678907459578618]
50	[0.3325400929154758]	[0.7720996592107847]
60	[0.3520564065645237]	[1.1649319541645191]
70	[0.23314276300706674]	[0.8618932719653669]
80	[0.14768252800096157]	[0.5373227700587154]
90	[0.0538273680921955]	[0.3705057762495029]
100	[0.04491115168422142]	[0.36867085361031604]

Figure 13: Code used to create a dataframe to store total absolute deviations with NASDAQ for various values of m and results.

After this we plotted the total absolute deviation between the index fund portfolio and the NASDAQ for 2019 and 2020. This plot is shown below.

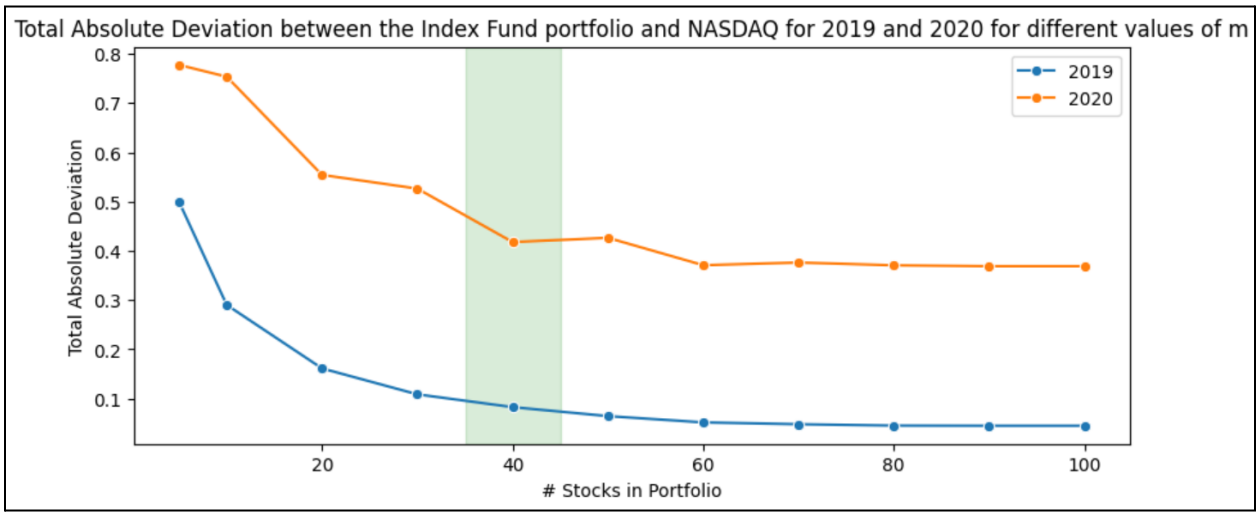

Figure 14: Total absolute deviation between Index Fund Portfolio and NASDAQ.

For each 'm' configuration in 2019, we recalibrated stock weights to mirror that year's index returns, laying the groundwork for 2020's performance evaluation. It was here that we first observed the diminishing returns on portfolio diversification.

In 2019, each 'm' increment presented a unique challenge: optimizing the stock weights to minimize the Total Absolute Deviation from the index's returns. Using the stock weights optimized for 2019, we proceeded to evaluate the portfolio's 2020 performance, focusing on the Total Absolute Deviation as our primary metric. The volatile market conditions of 2020 highlighted the stark contrast to 2019 and underscored the necessity for portfolio strategies that can adapt to changing market environments.

4.3 Analyzing Performance and Observing Diminishing Returns

In 2019, we began with a Total Absolute Deviation of 0.79, which improved significantly with increased diversification. However, 2020 was markedly different, starting with a deviation of 1.11. While expanding the portfolio did enhance performance, the gains were not as substantial, as changing to 90 stocks only reduces to 0.37, much higher than the TAD in 2019 of 0.045.

Interestingly, there's a sharp rise in TAD for both years when the portfolio size goes from 55 to 60 stocks. This suggests that the added stocks during this range might not be well-correlated with the NASDAQ index, leading to higher deviations.

Our analysis unveils a nuanced trajectory of Total Absolute Deviation as we expand our portfolio, affirming the concept of diminishing returns in investment diversification. We observe some trends of diminishing returns from the graph above (Figure 14) and here is the break down:

In-Sample Performance: 2019

- m = 5 to m = 20: A marked contraction in TDA as our equity basket expands. The delta between m = 5 and m = 10 is 0.09, while the leap from m = 10 to m = 20 shows a 0.22 decrement. This underscores the potency of diversification in mirroring our benchmark more closely.
- m = 20 to m = 40: The vigor of tracking error reduction starts diminishing. The shift from m = 20 to m = 30 docks by 0.06, and from m = 30 to m = 40, it recedes by 0.05. This insinuates that while augmentation still hones performance, the gradient is slackening.
- m = 40 to m = 60: Tracking error deceleration becomes more pronounced before increasing a large amount from m = 50 to m = 60. The dip from m = 40 to m = 50 stands at a mere 0.04, and tracking error reaches a maximum at m = 60.
- m = 60 to m = 80: The intensity of tracking error reduction picks up again. The decrease in tracking error from m = 60 to m = 70 and from m = 70 to m = 80 are 2 of the largest decreases we have seen so far.
- m = 80 and beyond: The tracking error reduction diminishes again, stagnating further to a 0.01 reduction from m = 90 to m = 100, spotlighting the diminishing utility of each subsequent stock addition.

Out-of-Sample Performance: 2020

- m = 5 to m = 30: We witness a consistent dip in TAD, signifying optimized performance.
- m = 30 to m = 50: TAD remains largely stagnant, insinuating that stock addition yields minimal traction.
- m = 50 to m = 60: An unexpected surge in TAD emerges, hinting at performance setbacks.
- m = 60 to m = 90: A dramatic plunge in TAD manifests, indicative of recuperating performance.

In our 2019 in-sample analysis, the principle of diminishing returns stands out prominently as we amplify m, especially evident post m=40. While the early inclusion of stocks amplifies alignment with our benchmark, subsequent additions contribute incrementally less, plateauing our performance.

The 2020 out-of-sample narrative sketches a more intricate contour, with tracking error experiencing ebbs and flows across varying m thresholds. Stagnation is evident between m=30−50, underlining the diminished returns principle.

This divergence in annual performances underscores the intricacies of crafting a portfolio that's resilient across temporal shifts. Given that the 2019 backdrop anchored our portfolio, 2020's market dynamism led to fluctuating alignment degrees, underpinning the need for a sagacious portfolio strategy.

Lastly, 2019's index might have exhibited a more stable daily return trajectory than 2020, potentially rendering the former's portfolio superior due to the prevalent market consistency.

5. Approach 2 (Integrated Weight and Stock Choice Strategy)

In our ongoing efforts to optimize portfolio performance, we have explored an alternative approach that seamlessly integrates weight and stock selection. By focusing on a Mixed-Integer Programming (MIP) model, we've opened up the potential for even finer-tuned portfolio optimizations. An alternate strategy to the stock selection problem involves bypassing the traditional stock selection IP and focusing on an MIP that limits the non-zero weights to a certain integer. This is achieved by replacing 'm' with 'n', thereby allowing for optimization across all weights, represented as:

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

Figure 15: Alternative Strategy Problem Formulation

Formulation and Constraints

- Binary Variables: We've employed a series of binary variables, y1 through yn. Utilizing the widely-respected 'Big M' technique, we ensure that stock weight wi is neutralized if its corresponding binary variable yi is inactive. This sophisticated approach gives us a finer control on stock inclusion.
- Challenges with Advanced Optimization: Despite the advanced capabilities of our tools, the inherent complexity of this model presents rigorous computational challenges. Gurobi, one of the industry's leading optimization tools, has been set with a stringent runtime parameter to ensure efficiency.

Our primary goal remains unwavering - minimize deviations and hone the alignment between our portfolio and the overarching market index. This model ensures that any differences between the returns of the index and our portfolio are within the bounds set by 'z'.

Moreover, the 'Big M' technique, a mainstay in our optimization, establishes a vital link between stock weights and binary variables. Utilizing Gurobi's advanced capabilities, each optimization run is governed by a strict time parameter. This strategic move is designed to pinpoint the most optimal stock portfolio, ensuring minimal deviations and maximizing alignment with the market benchmark.

Below is the code we used to explore this:

```
perf_2019 = []
perf_2020 = []
n = 100
p = len(data_2019_pct)
M = 100
a = [5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

#Creating a pandas dataframe so that we can output it as a CSV file
output_file=pd.DataFrame(columns = ['m', 'Stocks Selected', 'Weights','Absolute Deviation 2019','Absolute Deviation 2020','Execution Time (Sec)'])

for k,m in enumerate(a):
    print(f'For m = {m}')
    # Defining the model - Mixed Integer Programming
    model = gp.Model()
    # Defining the variables
    y = model.addMVar(n, vtype = 'B')
    w = model.addMVar(n)
    z = model.addMVar(p)
    model.setObjective(gp.quicksum(z[i] for i in range(p)), gp.GRB.MINIMIZE)
    # Adding Constraints
    model.addConstr(gp.quicksum(w[i] for i in range(n)) == 1)
    model.addConstr(gp.quicksum(y[i] for i in range(n)) == m)
    model.addConstrs(w[i] <= M*y[i]  for i in range(n))
    # Adding two constraints to make absolute value function as an LP
    model.addConstrs(z[i] >= q_2019[i] -
                     gp.quicksum(w[j]*data_2019_pct.iloc[i,j] for j in range(n))
                     for i in range(p))
    model.addConstrs(z[i] >= gp.quicksum(w[j]*data_2019_pct.iloc[i,j] for j in range(n))
                     - q_2019[i] for i in range(p))
    model.Params.OutputFlag = 0
    model.Params.TimeLimit = stop_time
    model.optimize()

    runtime = model.Runtime

    itemindex = np.where(y.x == 1) #storing index value for selected stocks
    list_of_selected_stocks=data_corr_2019.index.values[itemindex].tolist()
    print("Selected Stocks are: ",list_of_selected_stocks)
    weights = w.x[itemindex]
    print("\nWeights are:",weights)

    #Storing the returns of only these stocks in a new dataframe
    selected_stocks_returns = data_2019_pct[list_of_selected_stocks]
    selected_stocks_returns_array = selected_stocks_returns.to_numpy()

    #Storing the returns of only these stocks in a new dataframe for 2020
    selected_stocks_returns_2020 = data_2020_pct[list_of_selected_stocks]
    selected_stocks_returns_array_2020 = selected_stocks_returns_2020.to_numpy()

    #Now we will calculate absolute deviation of index fund from NASDAQ for 2019
    absolute_deviation_2019 = 0
    for i in range(len(data_2019_pct)):
        wr = 0
        for j in range(len(list_of_selected_stocks)):
            wr = wr + (weights[j]*selected_stocks_returns_array[i][j])
        s = abs(q_2019[i] - wr)
        absolute_deviation_2019 = absolute_deviation_2019 + s

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

    #Now we will calculate absolute deviation of index fund from NASDAQ for 2020
    absolute_deviation_2020 = 0
    for i in range(len(data_2020_pct)):
        wr = 0
        for j in range(len(list_of_selected_stocks)):
            wr = wr + (weights[j]*selected_stocks_returns_array_2020[i][j])
        s = abs(q_2020[i] - wr)
        absolute_deviation_2020 = absolute_deviation_2020 + s

    print ("\nAbsolute Deviation for year 2020 is:", absolute_deviation_2020)

    perf_2019.append(absolute_deviation_2019)
    perf_2020.append(absolute_deviation_2020)


    output_file.loc[k] = [m, list_of_selected_stocks, weights,absolute_deviation_2019,absolute_deviation_2020,runtime]
```

Figure 16: Code For Alternative Big M Method Strategy

Here is the TDA for different number of stocks for years 2019 and 2020, gotten after running the model for ten hours:

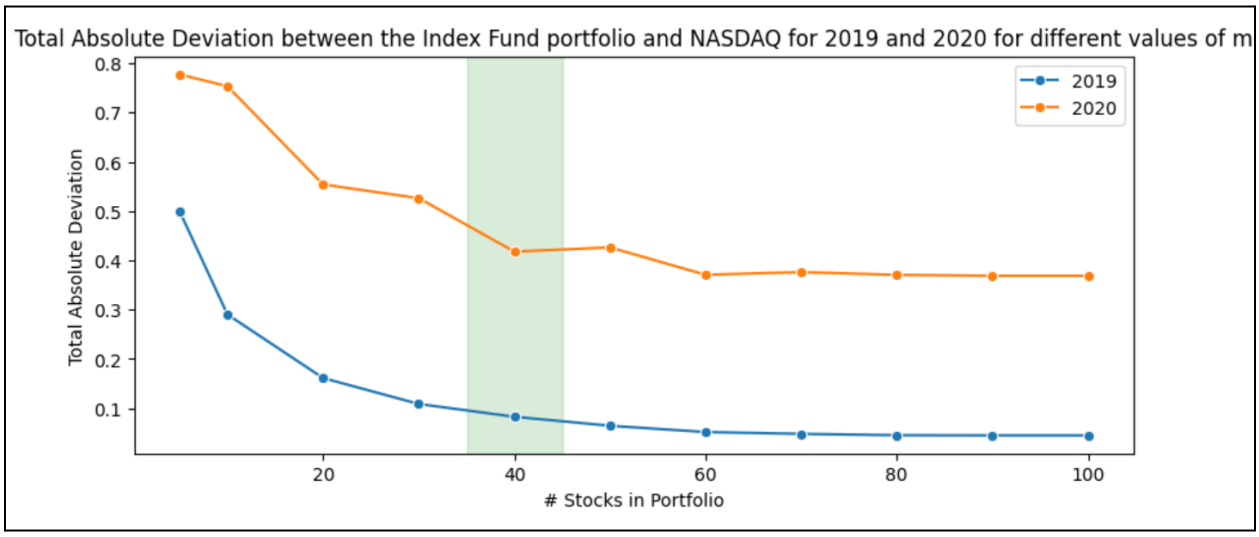

Figure 17: TAD for Big M Method

The Big M technique unveils the undeniable advantages of diversifying stock count. In the 2019 cycle, we noted an impressive decline in TAD, plummeting from 0.50 to 0.08. The 2020 iteration echoed this trend, revealing a drop from 0.78 to 0.36. This reinforces our long-held belief in the strategic merits of diversification, emphasizing that a more comprehensive stock selection can more accurately emulate benchmark performance.

Beyond the 60-stock threshold, the pace of tracking error enhancement begins to moderate. By the 70-stock mark, 2019's tracking error stabilizes at 0.05, indicating an inflection point where augmenting stock count yields marginal benefits. The 2020 spectrum aligns, steadying around 0.4. This suggests that there's an optimum point in portfolio expansion beyond which returns start to plateau. Beyond a certain threshold, mere expansion doesn't necessarily amplify the fidelity of our portfolio to its benchmark.

The Big M model illuminates a pivotal threshold around the 40-stock mark. Prior to this, each stock addition notably enhances the tracking error. Post this threshold, the portfolio starts exhibiting diminishing returns, suggesting a more streamlined portfolio could potentially strike an optimal balance between diversification and efficiency. Below is the figure for weight distribution of the 40-stocks chosen.

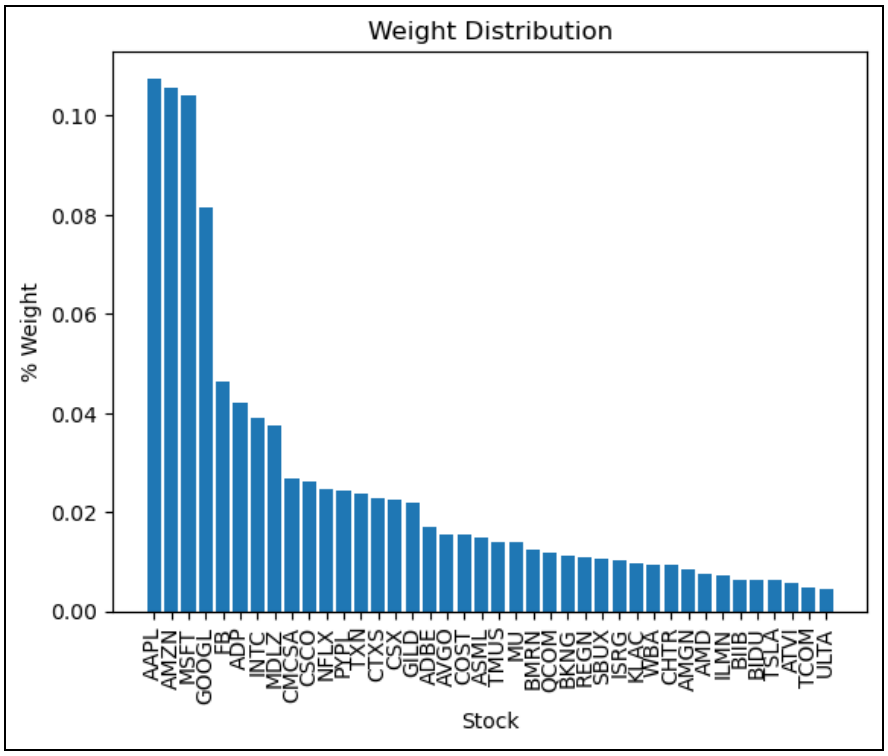

Figure 18: Weight distribution of selected stocks with m=40

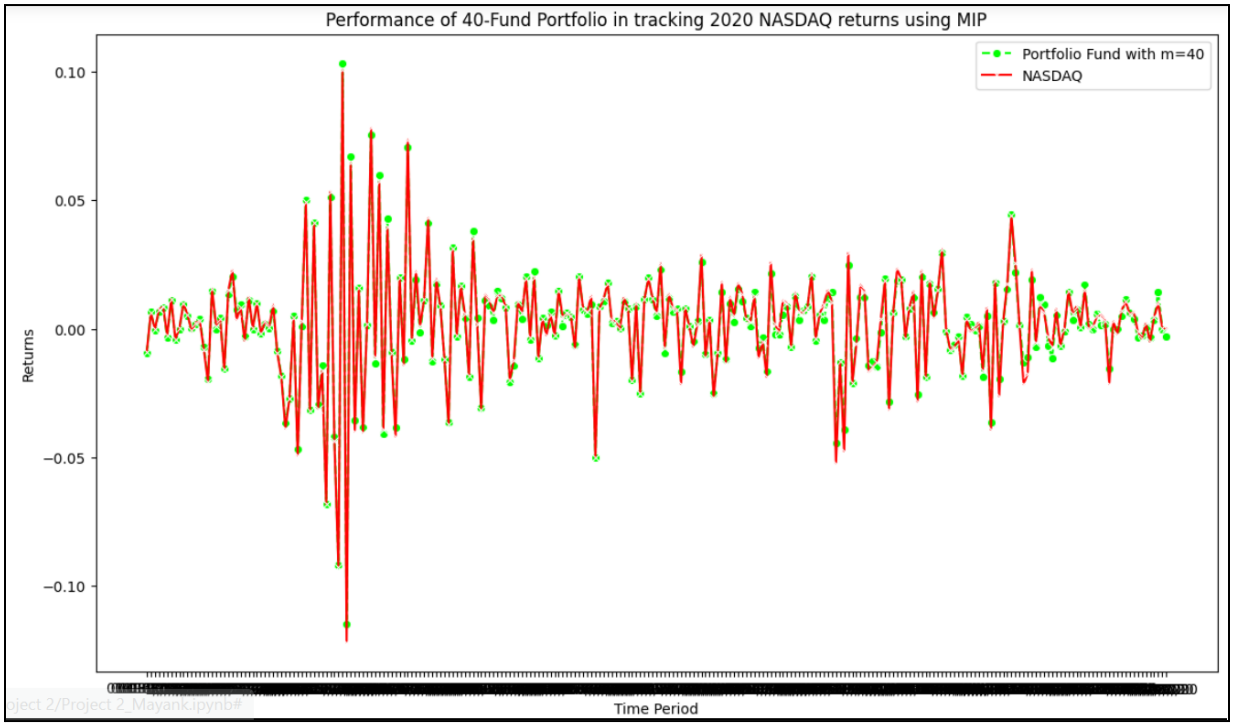

Figure 19: Comparison of our portfolio returns with m=40 vs NASDAQ returns

In-Sample vs. Out-of-Sample Dynamics

As our stock count swells, the in-sample tracking error exhibits consistent reduction, especially prominent in initial diversification stages. This underlines the Big M method's proficiency in dynamically improving portfolio performance as it diversifies.

Our 2020 metrics closely shadow 2019's performance trends, albeit with a gentler gradient of tracking error reduction. Nevertheless, the stability achieved around the 40-stock mark reaffirms the Big M approach's enduring adaptability across varying temporal and market scenarios.

6. Final Recommendations

Based on our research, we strongly endorse the adaptation of the **second methodology** that configures the weight selection challenge as a **Mixed Integer Program (MIP)**, emphasizing the constraint on the number of non-zero stock weights. A deep dive into our empirical findings, particularly comparing **figures 14 and 17**, suggests the superiority of this approach. The first methodology, albeit systematic in its two-step process, showcased a lag, presenting deviations between **0.8 and 1.2**. In contrast, the MIP-based method offered a sharper, more efficient trajectory, beginning with deviations between **0.5 and 0.8** and exhibiting a steeper corrective path.

Furthermore, we believe that **40 component stocks** is the right amount of stocks to include in our fund. Figure 17 shows total absolute deviation for different numbers of component stocks for both the training set from 2019 and the testing set from 2020. Both the total absolute deviations decrease rapidly as the number of component stocks increases from 10 to 40 but then decreases less rapidly as it increases from 40 to 100. The total absolute deviation from the 2020 data set drops below 0.4 when we have 40 component stocks, and then stays around the same even as the number of stocks increases close to 100. Therefore, we believe the mutual fund should select 40 component stocks using the second method to achieve the best results in mirror the movement of the NASDAQ 100.

Leveraging the MIP-based methodology will not only optimize our stock weights but also streamline our operational mechanics, reducing computational overhead and facilitating quicker responsiveness to market dynamics. The tapering off of benefits post the 40-stock mark underscores a critical risk management principle – overdiversification. By capping our portfolio at 40 stocks, we effectively mitigate the risk of diluting returns and veering off the market trajectory, ensuring a lean yet robust portfolio.

While our immediate strategy pivots around these findings, it is imperative to maintain an adaptive stance. Regular recalibration and data refreshes will be integral to ensure our strategy remains agile and aligned with evolving market contours.