Component Stock Selection Report

Group Members: Sharan Arora, Cate Dombrowski, Udit Dhand, Anurag Sahu

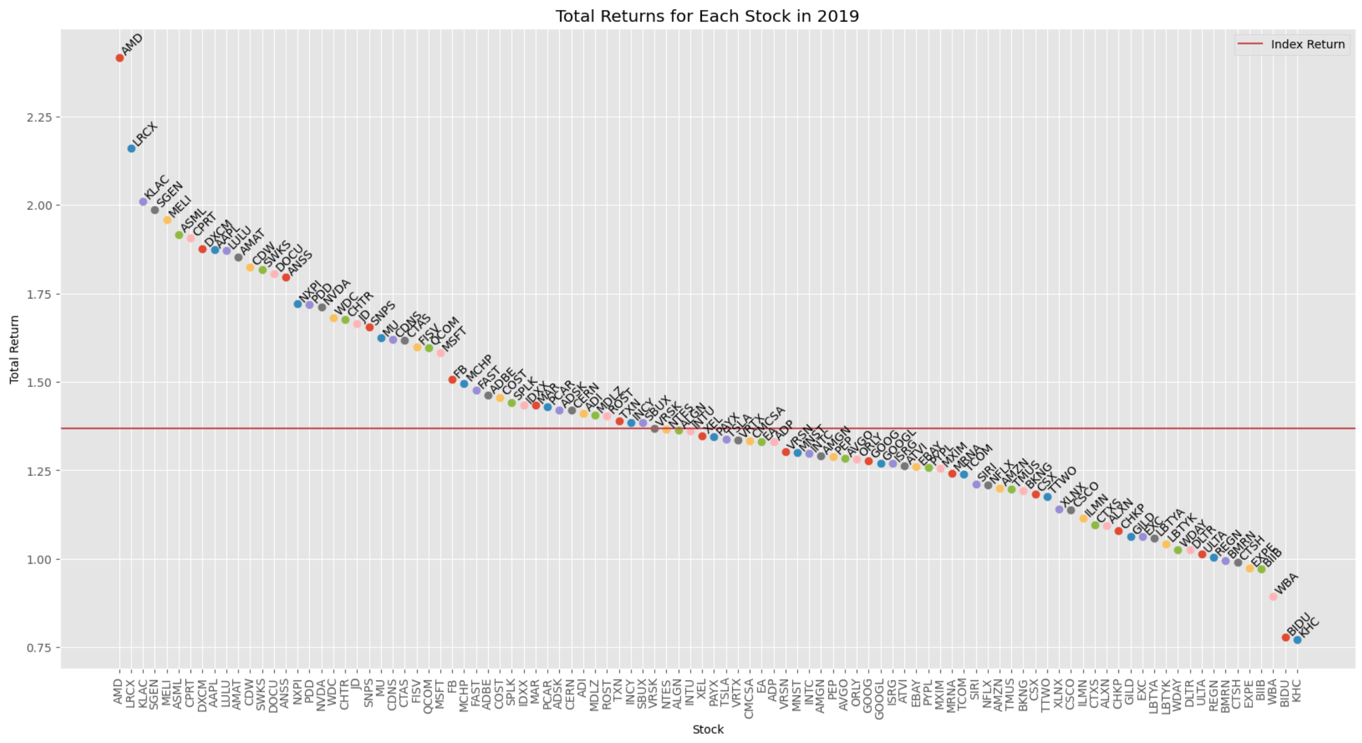
Introduction

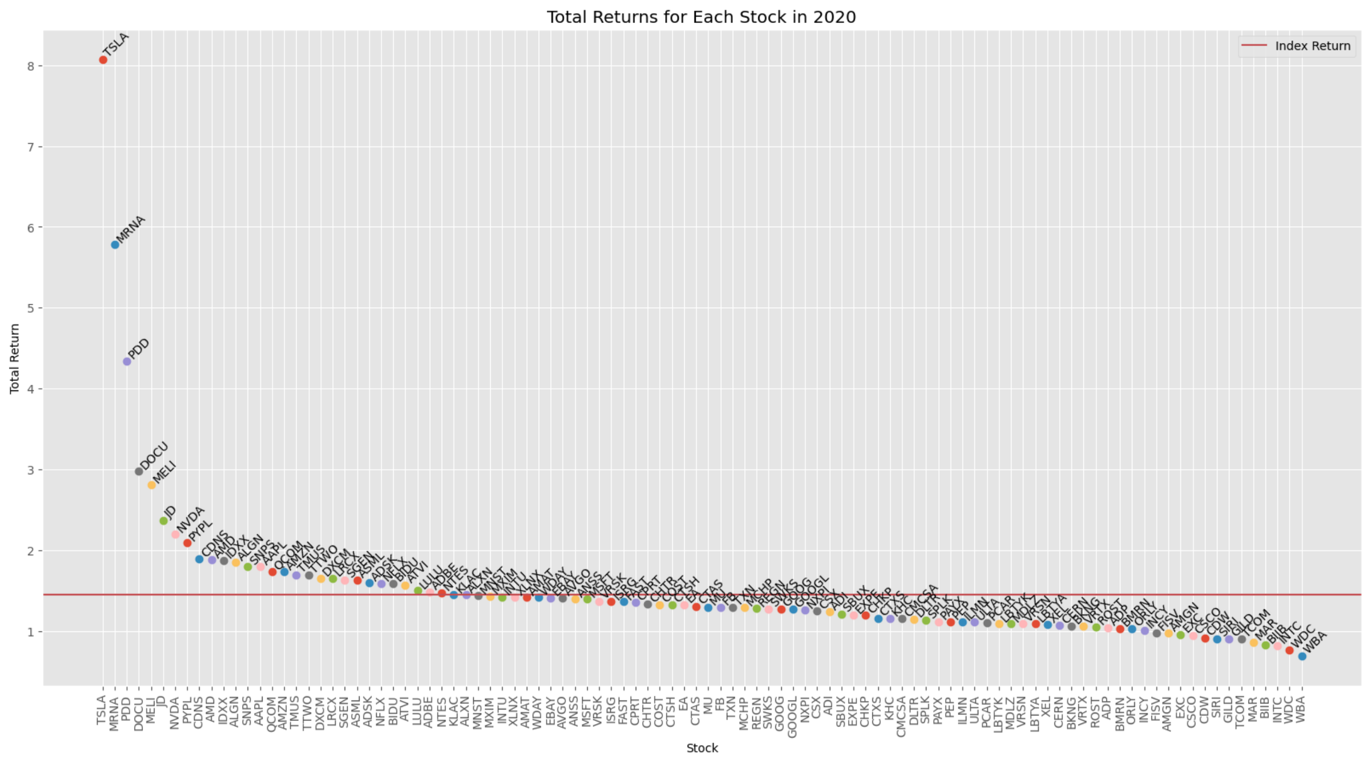
Having received your request, our team conducted an analysis of the 2019 and 2020 NASDAQ-100 index returns and component stocks. Considering our constraints, we have prepared the analysis, methodology, and recommendations of the number of component stocks to select as the corresponding portfolio weights. This report appropriately selects the component stocks for the optimal Index Fund that will closely track the NASDAQ-100. Attached in the report is our code, plots, and additional insights.

Problem Description

Replication of any index fund using all of its components can be costly as there would be a need for constant rebalancing as and when the index fund itself changes. We can instead choose to carefully construct a fund containing a smaller number of stocks compared to the index, as there would be many stocks within the index that would have high historical returns correlation to each other, we can pick one stock as representative of these group of stocks and assign appropriate weightage to simulate the effect of holding all of them in our portfolio.

Below are the individual and index returns for 2019 and 2020, we can see there is a spread in the individual returns and there are many stocks that end up giving similar returns, and co-move on a daily basis too (highly correlated).





Methodology

1. **Portfolio Construction**

   We utilized two approaches to constructing the portfolios:

   1. **Method 1 (IP + LP):**
      a. We first select *m* (less than 100) number of stocks that represent every stock in NASDAQ-100 by maximizing the correlation between these m stocks and the rest.
      b. For these *m* stocks we then calculate the weights by minimizing the absolute difference between index returns and portfolio return for each day of 2019.
   2. **Method 2 (MIP):**
      a. We construct a mixed integer programming model by restricting the number of weights that are more than 0 to *m* (IP) and minimizing the absolute difference between index returns and portfolio returns for given weight for each of the 100 stocks.

2. **Portfolio Performance**

   We create 20 sets of portfolios (10 for each method) by using *m* = 10, 20..100 and solving the optimization models for 2019 stock and index returns. We then check each portfolio on 2 metrics; how the portfolio tracked the index each day, and how the return of the portfolio differed from the index returns at the end of year.

Method 1:

1. **Decision Variables & Constraints**
   To be able to perform this analysis, it is important for us to consider our decision variables and constraints when using Gurobi for optimization models.

   For selection of *m* stocks, our decision variables and constraints are as follows:

   1. **Decision Variables (IP)**
      a. y_j is a binary variable (1 or 0), representing if stock *j* is selected from the fund.
      b. Binary variable x_{ij} indicates which stock j in the index is the best representative of stock i ($x_{ij} = 1$ if stock j in index is the most similar stock i, 0 otherwise)

   2. **Constraints (IP)**
      a. Out of n stocks, exactly m stocks must be selected $\sum_{j=1}^{n} y_j = 1$
      b. For each stock *i* $\{i = 1, 2, \dots n\}$, there must be exactly one representative $\sum_{j=1}^{n} x_{ij} = 1$
      c. Stock *i* is best represented by stock *j* only if *j* is in the fund $x_{ij} <= y_j$

   3. **Objective (IP)**
      a. Maximize $\sum_{j=1}^{n} \sum_{i=1}^{n} \rho_{ij} x_{ij}$ where ρ is the correlation of stock i with j.

   To find the weights of the chosen *m* stocks, our decision variables and constraints are as follows:

   1. **Decision Variables (LP)**
      a. Weights w_i for the stock *i* in the portfolio based on the results from the above IP problem
      b. Variables *r* which are substitutes for the original objective function which is the absolute difference between index and portfolio returns. This means it is nonlinear and we will convert it into a linear optimization program.

2. **Constraints (LP)**

    a. Sum of weights must equal one $\sum_{i=1}^{m} w_i = 1$

    b. r must be greater than or equal to the difference of q_t and $w_i r_{it}$ for all time t where q_t is the return index at time t and r_{it} return of stock i at time t

    c. r must be greater than or equal to the negative difference of q_t and $w_i r_{it}$

3. **Objective (IP)**

    b. Minimize $\sum_{t=1}^{T} = |q_t - \sum_{i=1}^{m} w_i r_{ij}|$

Below is the code representing optimization problem for method 1 (IP), here *m* is 5.

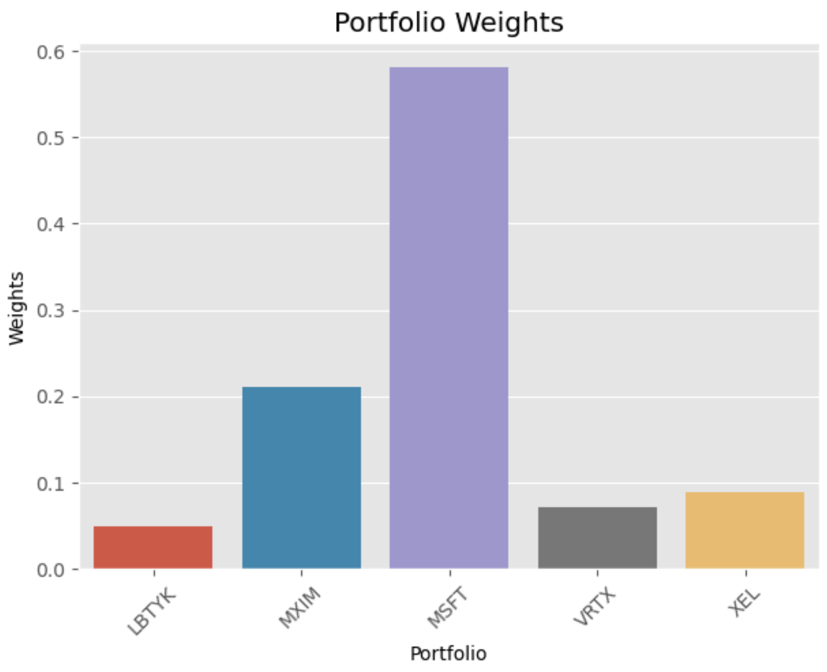
```
n_index = len(corr_2019)  # number of stocks in index

m = 5 # number of stocks in fund

# Model to find best m representative stocks
model = gp.Model()

x = model.addMVar(shape=(n_index, n_index), vtype='B')
y = model.addMVar(shape=n_index, vtype='B')

# Constraints
model.addConstr(gp.quicksum(y[j] for j in range(n_index)) == m)  # m stocks allowed in fund
model.addConstrs((gp.quicksum(x[i, j] for j in range(n_index)) == 1 for i in range(n_index))) # only 1 representative for each stock
model.addConstrs((x[i, j] <= y[j] for i in range(n_index) for j in range(n_index))) # stocks not picked cannot be representative

# Objective
model.setObjective(gp.quicksum(gp.quicksum(corr_2019.iloc[i, j] * x[i, j] for j in range(n_index)) for i in range(n_index)), sense=gp.GRB.MAXIMIZE)

# Solve model
model.Params.outputflag = 0
model.optimize()
```

Code for method 1 (LP):

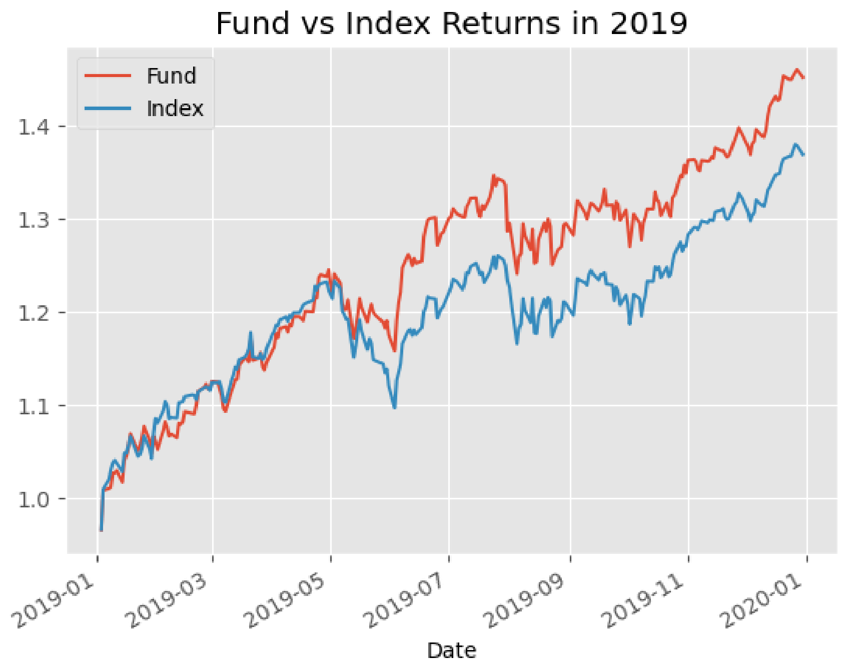
```
# Model to find weights for portfolio stocks
T = len(stock_returns_2019) # no of trading days

model = gp.Model()
w = model.addMVar(m, ub = 1, lb = 0) #weights
r = model.addMVar(T) #residue

# Constraints
model.addConstr( gp.quicksum(w[i] for i in range(m)) == 1 ) # weights add up to 1
model.addConstrs(  index_returns_2019.iloc[t] - gp.quicksum( w[i] * portfolio_returns.iloc[t, i ] for i in range(m)) <= r[t] for t in range(T))
model.addConstrs( -index_returns_2019.iloc[t] + gp.quicksum( w[i] * portfolio_returns.iloc[t, i ] for i in range(m)) <= r[t] for t in range(T))

#Objective and solution
model.setObjective(gp.quicksum(r[t] for t in range(T)))
model.Params.outputflag = 0
model.Params.warningflag = 0
model.optimize()
```

Initial Analysis:
We first tested for $m = 5$, and obtained weights for stocks using the 2019 data. Then, we tracked its performance using 2020 returns and the weights from 2019.

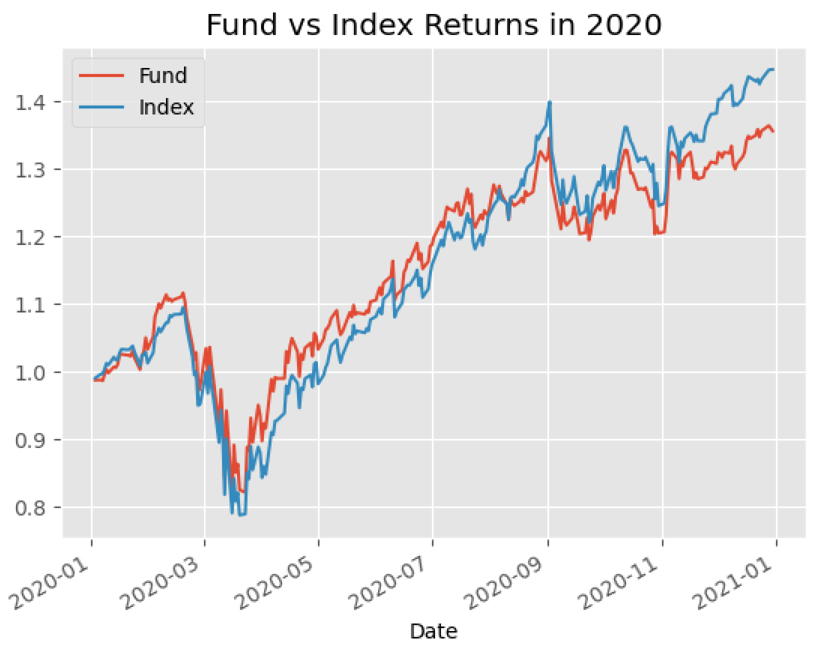
The weights obtained are as following:



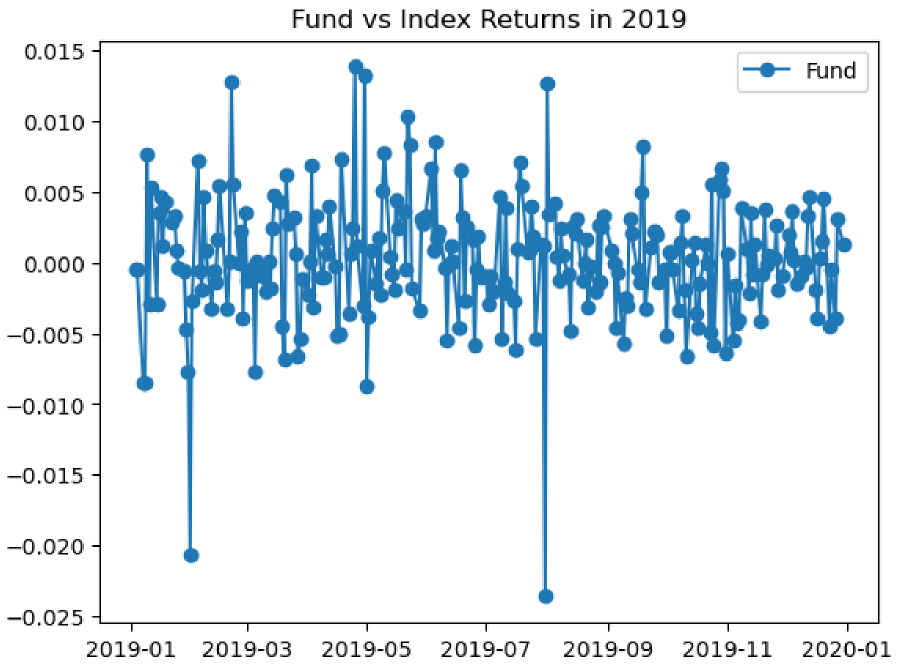
For 2019, this portfolio ended up returning 45.2%, which was 8.2% more than the index return of 37%.



We then tracked how this portfolio would track the index using 2020 stock returns. The fund returned 35.7% which was 9.1% less than index returns of 44.8%.



On average our fund of 5 stocks deviated from the index 0.3% everyday. All of the deviations are shown below.



Final Analysis (Method 1)

Finally, we computed the weights and returns when we changed the number of stocks selected. We iterated from 10 to 100, in increments of 10 and additionally compared in-sample and out-sample performance i.e. using 2019 weights to compare 2019 performance vs that of 2020.

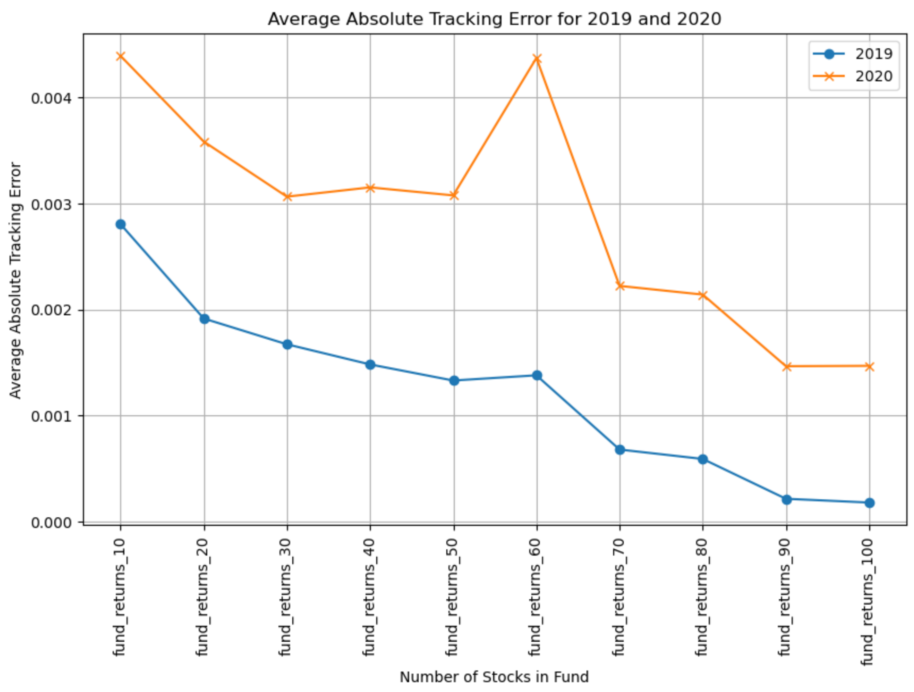
Code for same is below:

```python
# Iterate over different portfolio sizes
for m in range(10, 101, 10):

    # Model to find best m representative stocks
    model = gp.Model()

    x = model.addMVar(shape=(n_index, n_index), vtype='B')
    y = model.addMVar(shape=n_index, vtype='B')

    # Constraints
    model.addConstr(gp.quicksum(y[j] for j in range(n_index)) == m)  # m stocks allowed in fund
    model.addConstrs((gp.quicksum(x[i, j] for j in range(n_index)) == 1 for i in range(n_index))) # only 1 representative for each stock
    model.addConstrs((x[i, j] <= y[j] for i in range(n_index) for j in range(n_index))) # stocks not picked cannot be representative

    # Objective
    model.setObjective(gp.quicksum(gp.quicksum(corr_2019.iloc[i, j] * x[i, j] for j in range(n_index)) for i in range(n_index)), sense=gp.GRB.MAXIMIZE)

    # Solve model
    model.Params.outputflag = 0
    model.optimize()
```

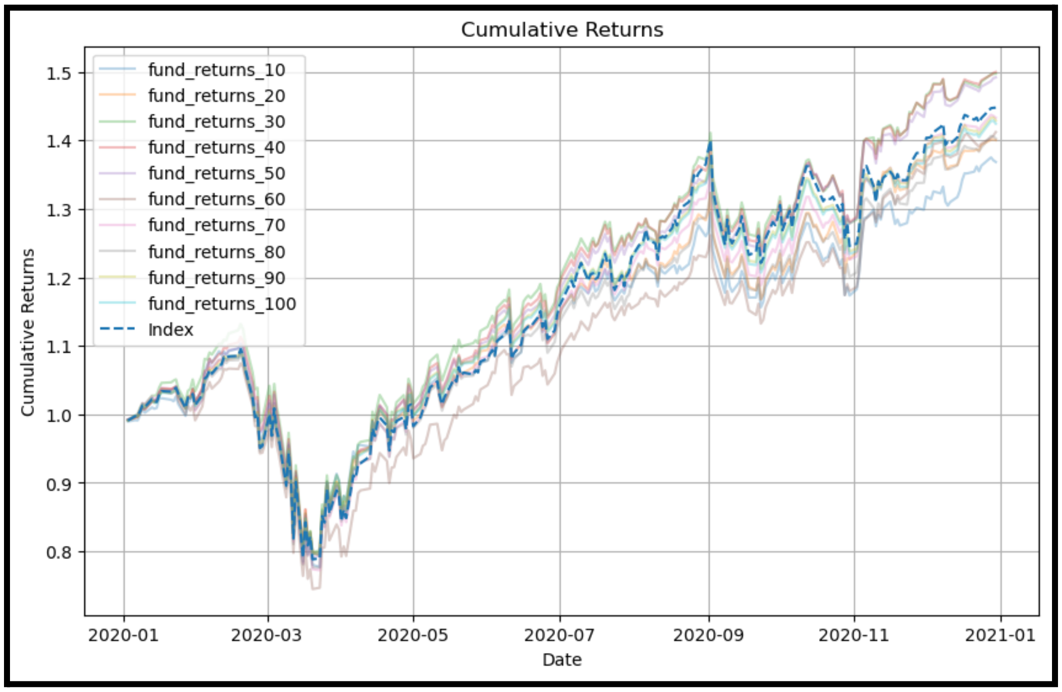
```python
    # Model to find weights for portfolio stocks
    T = len(stock_returns_2019) # no of trading days

    model = gp.Model()
    w = model.addMVar(m, ub = 1, lb = 0) #weights
    r = model.addMVar(T) #residue

    # Constraints
    model.addConstr( gp.quicksum(w[i] for i in range(m)) == 1 ) # weights add up to 1
    model.addConstrs(  index_returns_2019.iloc[t] - gp.quicksum( w[i] * portfolio_returns.iloc[t, i ] for i in range(m)) <= r[t] for t in range(T))
    model.addConstrs( -index_returns_2019.iloc[t] + gp.quicksum( w[i] * portfolio_returns.iloc[t, i ] for i in range(m)) <= r[t] for t in range(T))

    #Objective and solution
    model.setObjective(gp.quicksum(r[t] for t in range(T)))
    model.Params.outputflag = 0
    model.Params.warningflag = 0
    model.optimize()

    # Calculate and store daily portfolio returns in dataframe

    returns[f'fund_returns_{m}'] = (portfolio_returns[:]*w.x).sum(axis = 1)
    returns_2020[f'fund_returns_{m}'] = (portfolio_returns_2020[:]*w.x).sum(axis = 1)
    weights_IP = pd.concat([weights_IP, pd.DataFrame({'m': m, 'Stock': portfolio_returns.columns, 'Weight': w.x})], ignore_index=True)
```

We then plotted the tracking error for both years i.e., measured how well the *m* stocks predict the entire index performance in-sample and out-sample.

For 2019, we can see there is a clear downward trend in absolute tracking error, with a slight spike in error when we select 60 stocks followed by a drop between 60 and 70 selected stocks.



Overall error for 2020 is higher than in 2019, as expected since our model wasn't trained on this data. The trend for out-sample performance is similar to that of 2019, however, between 30-50 selected stocks, the error flattens. There is a huge spike in error with 60 selected stocks, then error decreases as *m* increases past 60. This is also reflected in our cumulative returns chart where 60 selected stocks are consistently underestimating the true index performance for 2020.



In this instance, we can clearly see a diminishing return of value by incrementing the number of stocks in our portfolio from 50 to 60, even if after 60 stocks our error continues to decline. Our in-sample performance is also better than out-sample performance as our model was trained on in-sample (2019) data, meaning we should expect a consistent decline in our error as our selected data increases. Out-sample prediction error, on the other hand, declines at first followed by an increase afterwards.
In order to determine if this performance can be improved, we devised an alternative strategy by rewriting the original problem.

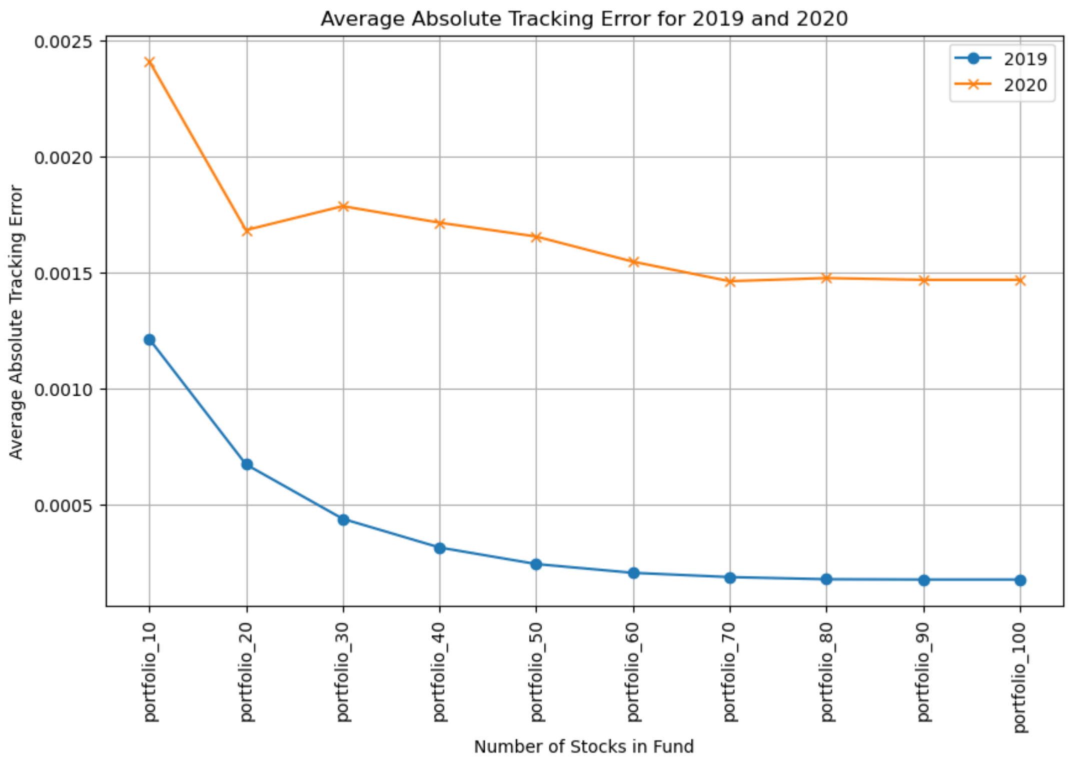
Method 2 (MIP)
Instead of utilizing the first integer program where we selected *m* stocks, we calculated optimal component stocks and weights using a Mixed Integer Program model. We included an additional constraint; the number of non-zero weights must be integers. We did this by creating additional binary decision variables for each stock in the index and replaced *m* with *n*. Additionally, now the weight of any stock can be non-zero. To solve $y_i = 0$, we added an *M* constraint, where weight w_i has to be less than or equal to $M*y_i$ where $M = 1$. The total sum of weights must be 1, therefore an individual weight cannot be greater than 1.

```
# Variables
y = model.addVars(N, vtype='B') # indicating if a stock has a non-zero weight
w = model.addVars(N, vtype='C') # weights
r = model.addVars(T, vtype='C') # residuals

# Objective
model.setObjective(gp.quicksum(r[t] for t in range(T)), sense=gp.GRB.MINIMIZE)

# Constraints
model.addConstrs(r[t] >= index_returns_2019[t] - gp.quicksum(w[i] * stock_returns_2019.iloc[t, i] for i in range(N)) for t in range(T))
model.addConstrs(r[t] >= -index_returns_2019[t] + gp.quicksum(w[i] * stock_returns_2019.iloc[t, i] for i in range(N)) for t in range(T))
model.addConstrs(w[i] <= M * y[i] for i in range(N))
model.addConstr(gp.quicksum(y[i] for i in range(N)) == m)
model.addConstr(gp.quicksum(w[i] for i in range(N)) == 1)
```

Beyond the new decision variables and additional constraints, we have also now made parameter adjustments for time limit as Mixed Integer Programming is much harder for Gurobi to solve, as it uses Branch and Bound. In our instance, we set the time limit to 1 hour for each value of *m* to improve programming efficiency and results.

```
# Gurobi parameters
# Parameter adjustments
model.setParam('TimeLimit', 3600)  # Limit to 1 hour
model.setParam('Heuristics', 1)  # Increase time spent in heuristics
model.setParam('NodeMethod', 1)  # Use dual simplex for node relaxations
model.setParam('VarBranch', 0)  # Use pseudo reduced cost branching
model.setParam('Cuts', 3)  # Use aggressive cut generation
model.setParam('MIPFocus', 2)  # Focus on finding feasible solutions (can change to 2 for optimality or 3 for improving the best bound)
```

With this, we see the following performance.



Compared to model 1, the absolute tracking error peak has gone down from 0.0045 to 0.0025 for m = 10. We also don't see a high error rate at m = 60. Overall, the model error declines as m increases. However, we see that at higher values of m, the error flattens out, with very little variability after m = 70 in-sample and out of sample.
Comparing this to the performance in the previous model, MIP provides a far better result. In our first analysis, m = 80 had a higher error rate than anything from m = 50 and higher in our final analysis.

Conclusion

Based on the comprehensive analysis conducted on the 2019 and 2020 NASDAQ-100 index returns, including the component stocks, several critical insights and findings have been derived. The analysis involved two distinct programming components, with a focus on selecting a specific number of stocks for the portfolio and determining the quantities of each selected stock to be included.

In the initial phase, the team devised a rigorous methodology employing Gurobi for optimization, with a clear delineation of decision variables and constraints. The approach facilitated the selection of an optimal set of stocks, followed by the determination of their weights in the portfolio. The assessment of the index fund's performance relative to the NASDAQ-100 index was conducted across different values of m, with an emphasis on tracking errors for both in-sample and out-sample data.

The first analysis, centered around a portfolio of five stocks, yielded significant insights into the weights and their subsequent impact on the weighted sum of returns. Subsequently, the team expanded their evaluation by varying the number of stocks selected from 10 to 100, revealing an intriguing pattern of tracking errors for both 2019 and 2020. Notably, the study highlighted a diminishing return of value beyond 50 to 60 stocks, with a gradual decline in tracking error until a certain threshold.

Furthermore, the team ingeniously revised their initial approach, transforming the second component program into a Mixed Integer Program (MIP) with specific additional constraints. By enforcing the integer requirement for the number of non-zero weights, the analysis was able to attain a higher degree of precision and performance. Notably, this revised model demonstrated a noticeable reduction in tracking error across different values of m, with a more consistent and controlled error rate compared to the earlier analysis.

Conclusively, the extensive analysis provides valuable insights into the construction of an optimal NASDAQ-100 index, emphasizing the importance of a balanced selection of component stocks and their corresponding weights in the portfolio. The findings underscore the significance of the MIP approach in achieving superior results, with a notable improvement in tracking error reduction and more stable performance, particularly for larger values of m. This comprehensive approach ensures the creation of an efficient and effective index fund strategy that aligns with the desired market performance and delivers promising returns for investors.

Recommendation

We recommend that the index fund should include 70 stocks. Looking at the out of sample error, at $m = 70$, the error rate is at a minimum. After that, the error rate flatlines, meaning there is no substantive benefit received from increasing the number of stocks in the fund. To select the corresponding weights moving forward, we advise that the MIP method be used, as it results in higher performance and a reduction in tracking error.