Optimizing Portfolio Selection for NASDAQ-100 Index Tracking: A Study in Equity Money Management Strategies

Date : 23rd October 2023

Team Members : Group 1

Simran Kaur
Bin(Melody)Yang
Meenal Gaba
Shrishty Mishra

Table of Contents

S No.	Topic	Page
1	Problem Context	3
2	Stock Selection Using Integer Programming	4
3	Portfolio Weights Calculation Using Linear Programming	6
4	Performance Analysis	11
5	Alternative Approach: Mixed Integer Programming (MIP) for Weight Selection	14
6	Conclusion	18

Introduction

The financial market is filled with countless investment strategies. Among these, there lies a delineation between ‘active’ and ‘passive’ strategies. This report delves into the realm of passive investing, particularly focusing on the construction of an index fund.

What is an Index Fund?

At its simplest, an index fund is a type of investment fund constructed to mirror or track the components of a market index, such as the NASDAQ-100 or the S&P 500. This allows investors to gain broad market exposure, without needing to buy individual stocks. Index funds have become increasingly popular due to their lower costs and historically competitive performance against actively managed portfolios.

Problem Context

In the world of equity fund management, investors often use passive investment strategies such as index funds to track the performance of broad market indexes such as the NASDAQ-100. Creating an index fund that perfectly mirrors an index by buying all n stocks of the same weight may seem simple, but the number of positions and the associated rebalancing and trading costs make it impractical and expensive.

Why Not Just Replicate the Entire Index?

While it might seem intuitive to buy every single stock in an index in order to track it, this approach poses logistical and financial challenges. Specifically:

- Volume of Stocks: The sheer number of stocks in major indices can make it cumbersome to manage.
- Rebalancing Costs: As stocks in the index change in value relative to each other, frequent rebalancing may be necessary, leading to transaction costs.
- Price Response to Trading: Buying or selling large quantities of stocks can inadvertently affect their prices.

Given these challenges, our objective is to create an efficient index fund for the NASDAQ-100 using a subset of its stocks, while still closely simulating its performance.

Methodology

Using historical stock prices from 2019, we aim to select the most representative stocks from the NASDAQ-100 and assign appropriate portfolio weights. This constructed portfolio will then be evaluated against 2020 data to determine its efficacy in tracking the index.

This project attempts to solve this challenge by creating an index fund with a smaller number of m shares, where m is significantly less than the total number of shares in the NASDAQ-100 index, n. To achieve this, we can follow one of the below mentioned approaches :

Approach 1 : Stock Selection followed by Adding Portfolio Weights

- Create an Integer Program for selecting a portfolio of exactly m stocks out of n, based on a similarity matrix ϱ, which gauges stock similarities using metrics like correlation.
- Use a Linear Program to determine the weight of each selected stock in the fund

Approach 2 : Calculating Stock Weights only using Mixed Integer Programming

- Instead of relying solely on the integer programming (IP) model for stock selection, an alternative approach involves the use of mixed integer programming (MIP) to determine portfolio weights directly.
- By leveraging the MIP approach, we look to optimize over ALL weights while incorporating certain constraints that ensure the final portfolio size (number of stocks with non-zero weights) aligns with our intended value of m.

Approach 1 : Stock Selection followed by Adding Portfolio Weights

❖ Write an integer program that selects exactly m out of n stocks for a portfolio. The program takes as input a 'similarity matrix', denoted as ϱ, where each element of ϱij represents the similarity of stocks i and j. This similarity can be measured by various metrics, such as the correlation between stock returns. However, other similarity metrics can also be considered.

❖ Solve the linear program to determine the value allocation for each selected stock in the portfolio.

❖ Evaluate the performance of the created index fund by comparing it to the NASDAQ-100 index using out-of-sample data. This evaluation is performed for different values of m to analyze the effect of stock selection on tracking performance.

1. A. Stock Selection

Objective

Select a subset (m stocks out of n) that most closely represents the movements of the entire NASDAQ-100 index.

Methodology

Calculating the correlation matrix for returns using the 2019 dataset. In order to construct the index fund, our task is to identify the top m stocks from the NASDAQ-100 that can effectively replicate the index's performance. The selection process relies on a similarity matrix denoted as ϱ, which is derived from the correlation matrix of stock returns obtained from the 2019 dataset.

Integer Programming

To solve this, we have considered Binary Decision Variables:

- y_j : Indicates if stock j is included in the fund (1 if included, 0 otherwise).
- x_{ij} : Indicates if stock j is the best representative for stock i (1 if so, 0 otherwise).

The goal is to select the optimal 5 stocks for a portfolio with weights w_i, determined using the 2019 data, in order to minimize the tracking error in 2020, i.e, to minimize the absolute differences between the actual 2020 index returns and the portfolio's weighted returns using the 2019 data. The tracking error is calculated as:

$\sum_j |q_j^{2020} - \sum_i w_i * r_{ij}^{2020}|$ for all j.

Here, q_j^{2020} represents the index returns in 2020, w_{ij} represents the weight of stock i in the portfolio, and r_{ij}^{2020} represents the stock returns for stock i in the time period j of 2020.

For Calculating Returns we have applied the following formula :

$$\text{Return (day}_t) = \frac{\text{Price (day}_{t-1}) - \text{Price (day}_t)}{\text{Price (day}_{t-1})}$$

The Integer program is given by:

$$\text{Maximize: } \sum_{i=1}^{n} \sum_{j=1}^{n} \rho_{ij} x_{ij}$$

- ϱ_{ij} : This term represents the similarity or interaction between item i and item j.
- x_{ij} : These are binary decision variables. Each x_{ij} variable takes a value of 0 or 1, indicating whether item i is selected (1) or not selected (0) to be part of the portfolio.

Objective function

Aims to find the combination of items (represented by the x_{ij} variables) that maximizes the overall similarity or interaction between the selected items, which is often a key objective in portfolio optimization and stock selection problems.

Subject to following constraints:

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

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

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

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

Constraints

- Exactly m stocks must be selected for the fund.
- Every stock i should have precisely one representative stock j.
- Stock i can only be represented by stock j if j is part of the fund.

```
# Create a new Gurobi model
model = gp.Model("StockSelection")

# Add decision variables
y = model.addVars(n, vtype='B', name="y")
x = model.addVars(n, n, vtype='B', name="x")

# Set objective function
model.setObjective(gp.quicksum(correlation_matrix_2019.iloc[i, j] * x[i, j] for i in range(n) for j in range(n)), GRB.MAXIMIZE)

# Add constraints
model.addConstr(y.sum() == m, "select_m_stocks")

for i in range(n):
    model.addConstr(x.sum(i, '*') == 1, f"stock_{i}_has_one_representative")

for i in range(n):
    for j in range(n):
        model.addConstr(x[i, j] <= y[j], f"representative_link_{i}_{j}")
```

Results and Analysis

The optimal five stocks that the Integer Program gave us are :

- LBTYK
- MXIM
- MSFT
- VRTX
- XEL

The code and output for the same is shown below in the attached screenshot :

```
# Extract the selected stocks
selected_stock_indices = [i for i, var in enumerate(y.values()) if var.x > 0.5]

# Stock tickers for the selected stocks
selected_stock_tickers = correlation_matrix_2019.index[selected_stock_indices].tolist()

selected_stock_tickers
```

```
Set parameter Username
Academic license - for non-commercial use only - expires 2024-08-20
['LBTYK', 'MXIM', 'MSFT', 'VRTX', 'XEL']
```

Below is the snapshot of the returns obtained :

	NDX	ATVI	ADBE	AMD	ALXN	ALGN	GOOGL	GOOG	AMZN	AMGN	...	TCOM	ULTA	VRSN	VRSK	VRTX	WBA	WDAY	WDC	
1	-0.033602	-0.035509	-0.039498	-0.094530	0.022030	-0.085791	-0.027696	-0.028484	-0.025242	-0.015216	...	-0.022834	-0.018591	-0.034989	-0.030557	-0.002133	-0.010435	-0.035808	-0.068217	-0.0(
2	0.044824	0.039903	0.048632	0.114370	0.057779	0.010445	0.051294	0.053786	0.050064	0.034184	...	0.058976	0.047954	0.044744	0.044147	0.054785	0.033269	0.060577	0.042356	0.0(
3	0.010211	0.028196	0.013573	0.082632	0.018302	0.017192	-0.001994	-0.002167	0.034353	0.013457	...	0.022067	0.062620	0.016312	0.001000	0.018240	0.005749	0.000918	0.035522	-0.0(
4	0.009802	0.030309	0.014918	0.008751	0.006207	0.015954	0.008783	0.007385	0.016612	0.012824	...	0.010281	0.018450	0.036460	0.008902	0.013307	0.020009	0.021101	-0.009615	0.0
5	0.007454	0.017210	0.011819	-0.026988	0.012430	0.038196	-0.003427	-0.001505	0.001714	-0.001196	...	0.023745	0.018804	-0.008157	0.003781	0.023065	0.012050	-0.004612	0.044083	-0.0(

Below is the snapshot of the correlation matrix obtained :

	ATVI	ADBE	AMD	ALXN	ALGN	GOOGL	GOOG	AMZN	AMGN	ADI	...	TCOM	ULTA	VRSN	VRSK	VRTX	WBA	WDAY	WDC	XEL
ATVI	1.000000	0.399939	0.365376	0.223162	0.216280	0.433097	0.426777	0.467076	0.203956	0.329355	...	0.322906	0.128241	0.464850	0.316549	0.259679	0.218149	0.311659	0.303077	0.043389
ADBE	0.399939	1.000000	0.452848	0.368928	0.363370	0.552125	0.540404	0.598237	0.291978	0.473815	...	0.360392	0.201151	0.711339	0.541243	0.402171	0.228106	0.650430	0.361516	0.207403
AMD	0.365376	0.452848	1.000000	0.301831	0.344252	0.418861	0.417254	0.549302	0.151452	0.503733	...	0.332776	0.210623	0.498342	0.330900	0.272983	0.281950	0.407626	0.438892	0.017283
ALXN	0.223162	0.368928	0.301831	1.000000	0.332433	0.315993	0.307698	0.363170	0.342022	0.317040	...	0.257143	0.408936	0.350581	0.191489	0.522423	0.192720	0.416396	0.289908	0.047947
ALGN	0.216280	0.363370	0.344252	0.332433	1.000000	0.248747	0.250316	0.399281	0.264599	0.328280	...	0.175957	0.128559	0.360886	0.251855	0.334978	0.219595	0.308968	0.284407	0.088059

1. B. Calculating Portfolio Weights Calculation Using Linear Programming

Objective

Select optimal Portfolio weights so as to minimize the difference in returns between the index and fund.

Methodology

The goal is to minimize the absolute differences between the returns of the index (q_t) and the weighted sum of the returns of the selected stocks in the portfolio ($\sum_{i=1}^{m} w_i r_{it}$) for each time period t in the range of 1 to T_m

$$\sum_{t=1}^{T_m} |q_t - \sum_{i=1}^{m} w_i r_{it}|$$

Subject to :

($\sum_{i=1}^{n} w_i$ =1) : This constraint ensures that the weights of the selected stocks in the portfolio sum up to 1.

Notation:

- r_{it} : Return of stock i at time t.
- q_t : Return of the NASDAQ-100 index at time t.
- w_i : Weight of stock i in the portfolio.

Objective Function

We can rewrite our objective as below :

$$\text{Minimize} \quad \sum_{i=1}^{n} \sum_{t=1}^{T_m} z_{it}$$

Constraints

$$z_{it} \geq q_t - \sum_{i=1}^{n} w_i r_{it} \quad \forall i = 1, 2, \dots, n, \quad t = 1, 2, \dots, T_m$$

$$z_{it} \geq -\left(q_t - \sum_{i=1}^{n} w_i r_{it} \right) \quad \forall i = 1, 2, \dots, n, \quad t = 1, 2, \dots, T_m$$

$$\sum_{i=1}^{n} w_i = 1$$

$$w_i \geq 0 \quad \forall i = 1, 2, \dots, n$$

$$z_{it} \geq 0 \quad \forall i = 1, 2, \dots, n, \quad t = 1, 2, \dots, T_m$$

- $z_{it} \geq q_t - \sum_{i=1}^{n} w_i r_{it}$ =1 : This constraint ensures that the absolute value difference $q_t - \sum_{i=1}^{n} w_i r_{it}$ is greater than or equal to the auxiliary variable z_{it} for all i and t.

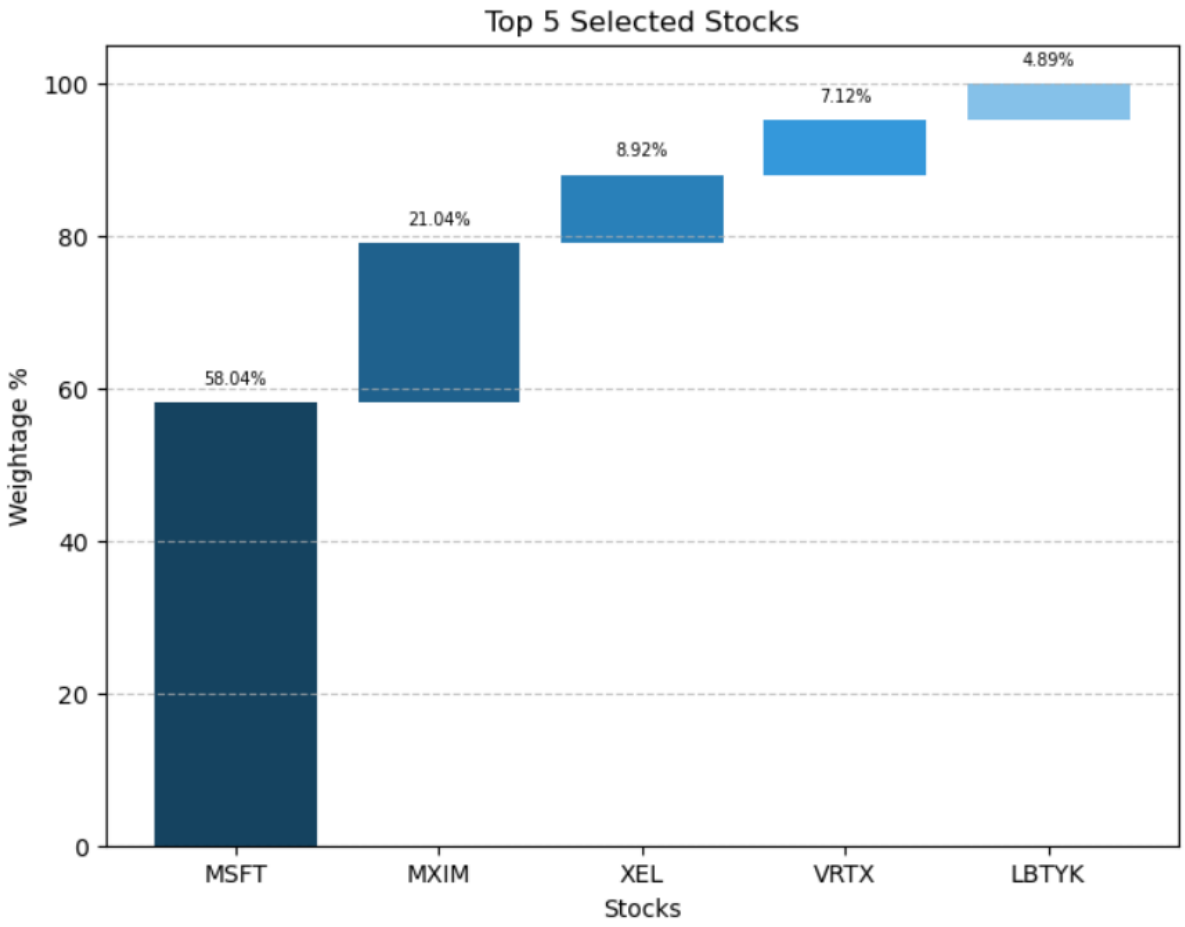
- $z_{it} \geq -(q_t - \sum_{i=1}^{n} w_i r_{it} = 1)$: This constraint enforces that the auxiliary variable zit is greater than or equal to the negation of the difference between q_t and $\sum_{i=1}^{n} w_i r_{it}$ for all i and t.

```
# Extract returns for the selected stocks and the index
selected_returns_2019 = returns_2019.drop('NDX',axis=1).iloc[:, selected_stock_indices]
index_returns_2019 = returns_2019['NDX']

# Create a new Gurobi model for portfolio weights
model_weights = gp.Model("PortfolioWeights")

# Number of time periods
T = selected_returns_2019.shape[0]

# Add decision variables
w = model_weights.addVars(m, vtype=GRB.CONTINUOUS, name="w", lb=0)  # Portfolio weights
z = model_weights.addVars(T, vtype=GRB.CONTINUOUS, name="z", lb=0)  # Absolute differences

# Set objective function
model_weights.setObjective(z.sum(), GRB.MINIMIZE)

# Add constraints: sum of weights should be 1
model_weights.addConstr(w.sum() == 1, "sum_weights")

# Add constraints: absolute differences between portfolio return and index return for each time t
for t in range(T):
    portfolio_return = gp.quicksum(w[i] * selected_returns_2019.iloc[t, i] for i in range(m))
    model_weights.addConstr(z[t] >= index_returns_2019.iloc[t] - portfolio_return, f"abs_diff_1_{t}")
    model_weights.addConstr(z[t] >= portfolio_return - index_returns_2019.iloc[t], f"abs_diff_2_{t}")
```

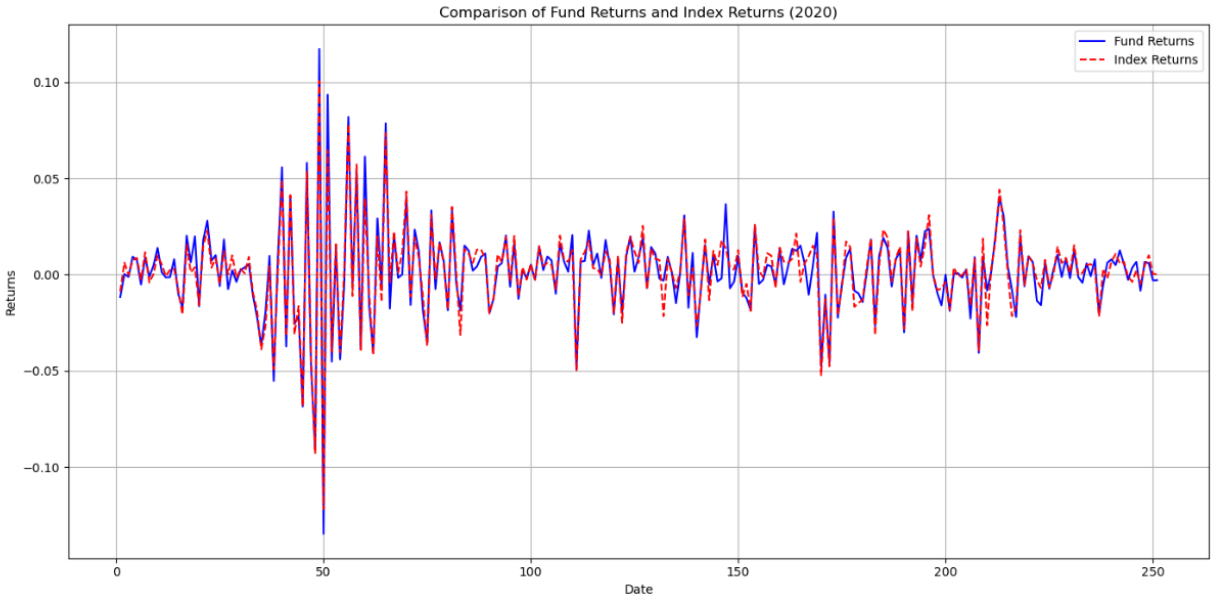
Results and Analysis

The below chart displays the out top 5 Selected stocks with their respective weight percentages in the portfolio.

The vertical axis (Y-axis) represents the weightage percentage, totaling from 0% to 100%. The horizontal axis (X-axis) lists the stock names . The size of each blue bar corresponds to the weightage of each stock. The larger the bar, the greater the weightage of that stock in the selection.



The below graph displays a "Comparison of Fund Returns and Index Returns (2020).



Key Observations from the Above charts :

1. Vertical Y axis represents the returns and horizontal X axis represents the trading days
2. Both the fund and the index have shown fluctuations over the period, indicating the volatile nature of the market in 2020. The graph has several peaks and troughs, which suggest changes in the returns.
3. The two lines tend to move in a somewhat similar pattern, suggesting some level of correlation between the fund returns and the index returns. This is common when a fund tries to mimic or outperform a specific index.
4. Both the fund and the index have shown fluctuations over the period, indicating the volatile nature of the market in 2020. The graph has several peaks and troughs, which suggest changes in the returns

By analyzing the relative movements and differences between the two lines, we can gauge the fund's performance against the broader market.

1. C. Varying number of Stocks Selected for Performance Analysis

Objective

- Constructing a portfolio for various values of "m," representing the number of stocks within the range of 10 to 100. The specific procedure for creating the portfolio is not explicitly stated in the question.
- Evaluating the performance of each portfolio by computing metrics such as returns, risk, and Sharpe ratio, among others and to examine how portfolio performance evolves with changing values of "m."

We need to ascertain whether there is a threshold at which including additional stocks in the portfolio no longer yields a significant improvement in performance, indicating the presence of diminishing returns.

We will also contrast the portfolio's performance in 2019 using 2019 data with its performance in 2020 using 2020 data. Identify and provide explanations for any disparities in performance.

Methodology

- Calculate the daily returns for both the index and the stocks in 2020.
- Compute the absolute tracking error, which quantifies how closely the constructed portfolio's returns match the index's returns.
- Analyze the performance for varying values of m (10, 20,30..., 100).

In the provided code and output, we are performing a portfolio optimization and evaluation exercise for different portfolio sizes (m) using historical financial data from 2019 and 2020. Here's a breakdown of what's happening:

```
# Create a function to select stocks and compute portfolio weights
def compute_portfolio(m, returns_2019, index_returns_2019, correlation_matrix_2019):
    # Model for stock selection
    model = gp.Model("StockSelection")
    y = model.addVars(n, vtype='B', name="y")
    x = model.addVars(n, n, vtype='B', name="x")
    model.setObjective(gp.quicksum(correlation_matrix_2019.iloc[i, j] * x[i, j]
                                   for i in range(n) for j in range(n)), GRB.MAXIMIZE)
    model.addConstr(y.sum() == m, "select_m_stocks")
    for i in range(n):
        model.addConstr(x.sum(i, '*') == 1, f"stock_{i}_has_one_representative")
    for i in range(n):
        for j in range(n):
            model.addConstr(x[i, j] <= y[j], f"representative_link_{i}_{j}")
    model.Params.OutputFlag = 0
    model.optimize()
    selected_stock_indices = [i for i, var in enumerate(y.values()) if var.x > 0.5]
    selected_stock_tickers = correlation_matrix_2019.index[selected_stock_indices].tolist()

    # Model for portfolio weights
    selected_returns = returns_2019.drop('NDX',axis=1).iloc[:, selected_stock_indices]
    T = selected_returns.shape[0]
    model_weights = gp.Model("PortfolioWeights")
    w = model_weights.addVars(m, vtype=GRB.CONTINUOUS, name="w", lb=0)
    z = model_weights.addVars(T, vtype=GRB.CONTINUOUS, name="z", lb=0)
    model_weights.setObjective(z.sum(), GRB.MINIMIZE)
    model_weights.addConstr(w.sum() == 1, "sum_weights")
    for t in range(T):
        portfolio_return = gp.quicksum(w[i] * selected_returns.iloc[t, i] for i in range(m))
        model_weights.addConstr(z[t] >= index_returns_2019.iloc[t] - portfolio_return, f"abs_diff_1_{t}")
        model_weights.addConstr(z[t] >= portfolio_return - index_returns_2019.iloc[t], f"abs_diff_2_{t}")
    model_weights.Params.OutputFlag = 0
    model_weights.optimize()
    portfolio_weights = [var.x for var in w.values()]

    return selected_stock_tickers, portfolio_weights
```

```
# Evaluate portfolio performance
def evaluate_portfolio_performance(selected_stock_tickers, portfolio_weights, returns, index_returns):
    selected_returns = returns[selected_stock_tickers]
    tracking_error = sum(abs(index_returns - selected_returns @ portfolio_weights))
    return tracking_error
```

Code Explanation:

- The code defines a function `compute_portfolio(m, returns_2019, index_returns_2019, correlation_matrix_2019)` to select a portfolio of stocks and compute the portfolio weights for a given value of m. It uses mathematical optimization to select the stocks that maximize the correlation within the portfolio.

- Another function: *evaluate_portfolio_performance(selected_stock_tickers,portfolio_weights,returns, index_returns)* is defined to evaluate the portfolio's tracking error. It calculates the tracking error between the portfolio returns and the index returns.

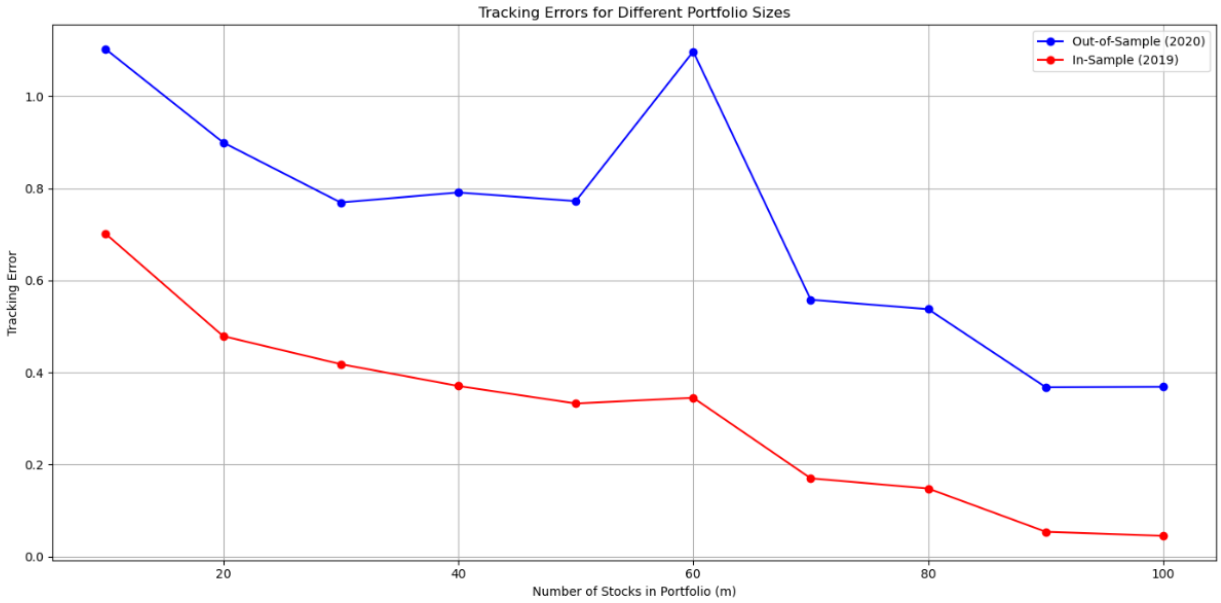
Output Explanation:

- `out_of_sample_performance`: This list contains tracking errors for the out-of-sample performance of the portfolio for each value of m. In other words, it

represents how well the portfolio constructed in 2019 performed when applied to 2020 data.

- `in_sample_performance`: This list contains tracking errors for the in-sample performance of the portfolio for each value of m. It reflects how well the portfolio constructed in 2019 performed when tested against 2019 data.

Results and Analysis



This graph illustrates "Tracking Errors for Different Portfolio Sizes."

Key Points:

- For both years, as the number of stocks in the portfolio increases, the tracking error generally decreases. This suggests that more diversified portfolios (with more stocks) tend to have returns that are more closely aligned with the benchmark, resulting in a lower tracking error.

- The out-of-sample (2020) data shows a significant spike in tracking error when the portfolio size is around 60 stocks. This could be due to various external factors such as coronavirus pandemic side effects

In conclusion, the graph shows how the tracking error of a portfolio varies with its size for two different years. The general trend suggests diversification (adding more stocks to a portfolio) can reduce tracking error.

Approach 2 : Mixed Integer Programming (MIP) for Weight Selection

Overview

Instead of exclusively utilizing the integer programming (IP) model for the purpose of stock selection, we consider an alternative and potentially more direct method. This method employs mixed integer programming (MIP) to ascertain the ideal portfolio weights. The MIP methodology offers a broader scope, as it aims to optimize across the entire spectrum of weights. Moreover, it integrates specific constraints to guarantee that the ultimate portfolio composition—represented by the number of stocks that have non-zero weights—corresponds accurately with our predetermined target of \(m \). This approach not only streamlines the weight determination process but also ensures that our portfolio adheres to the desired structure and size.

Methodology

Here we will employ the use of Mixed Integer Programming to find the optimal solution, the stocks and their weights.

- Weight Optimization: Optimize the weights for all stocks in the index to mirror the NASDAQ-100.
- Binary Variables Introduction: Introduce binary variables y1,y2,y3,...,yn where yi = 1 if stock i is selected and yi = 0 otherwise.
- 'Big M' Constraints: Using the 'big M' technique, enforce constraints such that wi = 0 if yi = 0 Portfolio Size Constraint: Ensure that the sum of the y's is equal to m, which guarantees that only m stocks are selected in the final portfolio.

Given the complexity of this problem, the computational process could be lengthy. To ensure feasibility, Gurobi was limited to an hour per m value.

Mixed Integer Programming

Objective function

To minimize :

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

Variables : 1. w_i: weight of stock i in the portfolio (continuous variable).
2. y_i: binary variable indicating whether stock i is selected (1) or not (0).

Constraints

1. Stock Selection Constraint :

$$\sum_{i=1}^{n} y_i = m$$

2. The sum of all the weights should be equal to 1

$$\sum_{i=1}^{n} w_i = 1$$

3. Big M constraint

$$w_i \leq M y_i \quad \text{for all } i$$

$$w_i \geq -M y_i \quad \text{for all } i$$

Below is the code snippet :

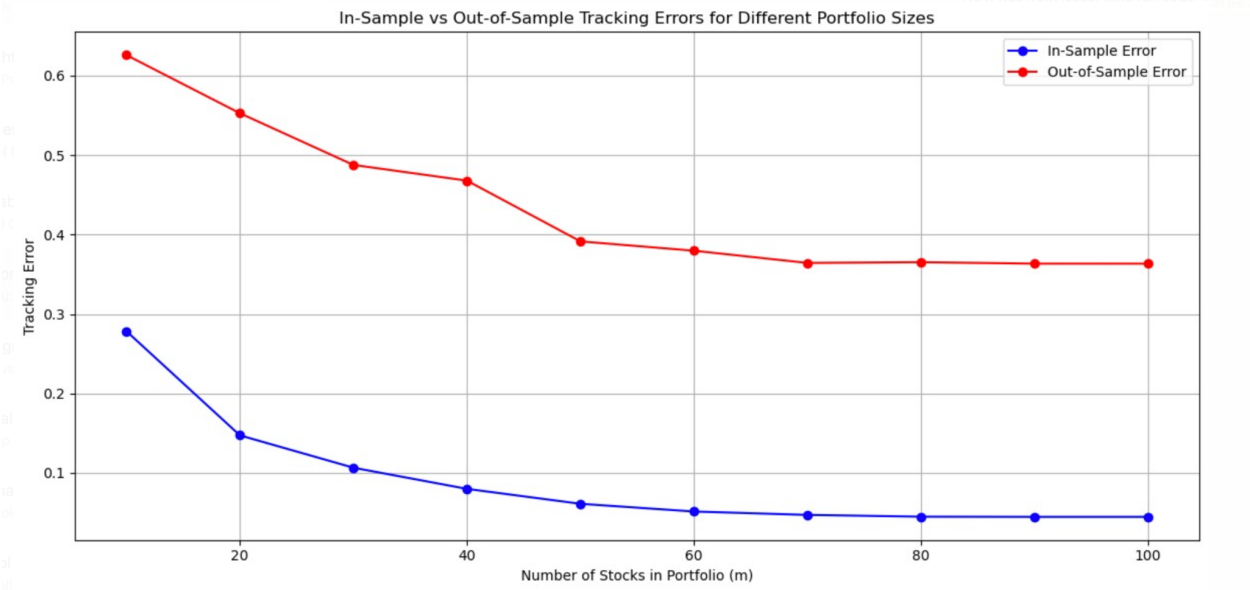
```
for m in ms:
    model = gp.Model("PortfolioWeightsBigM")

    w = model.addVars(n, vtype=GRB.CONTINUOUS, name="w", lb=0)
    y = model.addVars(n, vtype=GRB.BINARY, name="y")
    z = model.addVars(T, vtype=GRB.CONTINUOUS, name="z", lb=0)

    model.setObjective(z.sum(), GRB.MINIMIZE)
    model.addConstr(w.sum() == 1, "sum_weights")
    model.addConstr(y.sum() == m, "select_m_weights")

    for i, stock in enumerate(returns_2019.columns[1:]):
        model.addConstr(w[i] <= M * y[i], f"big_M_constraint_upper_{i}")
        model.addConstr(w[i] >= -M * y[i], f"big_M_constraint_lower_{i}")

    for t in range(T):
        portfolio_return = gp.quicksum(w[i] * returns_2019[stock].iloc[t] for i, stock in enumerate(returns_2019.columns[1:]))
        model.addConstr(z[t] >= returns_2019['NDX'].iloc[t] - portfolio_return, f"abs_diff_1_{t}")
        model.addConstr(z[t] >= portfolio_return - returns_2019['NDX'].iloc[t], f"abs_diff_2_{t}")
```

Results and Analysis

The table below presents tracking errors for different portfolio sizes in two different years: 2019 (In-Sample) and 2020 (Out-of-Sample).

m	Performance_2019 (In-Sample Error)	Performance_2020 (Out-of-Sample Error)
10	0.278571	0.626074
20	0.147351	0.552829
30	0.106611	0.487730
40	0.079984	0.467973
50	0.061195	0.391497
60	0.051471	0.379741
70	0.047221	0.364377
80	0.044999	0.365333
90	0.044768	0.363453
100	0.044768	0.363432

The graph above represents the "In-Sample" vs "Out-of-Sample" tracking errors for different portfolio sizes. Here's an analysis of the presented data:



- **In-Sample Error (Blue Line)**: Represents the tracking error based on historical data that was likely used to construct or optimize the portfolio. It consistently decreases as the number of stocks in the portfolio increases. This suggests that diversifying the portfolio (increasing the number of stocks) results in a better fit to the benchmark within this sample.

- **Out-of-Sample Error (Red Line)**: Represents the tracking error based on data that was not used during the portfolio construction or optimization process. It provides a measure of how well the portfolio might perform in the future or under different circumstances. This line is less consistent, showing some volatility, especially around the 40-60 stock range, before stabilizing.

Key Points :

- **Diversification Benefits:** As the number of stocks in the portfolio increases, the in-sample tracking error consistently decreases. This showcases the benefits of diversification, where adding more stocks can help in reducing unsystematic risks and aligning the portfolio closer to its benchmark.

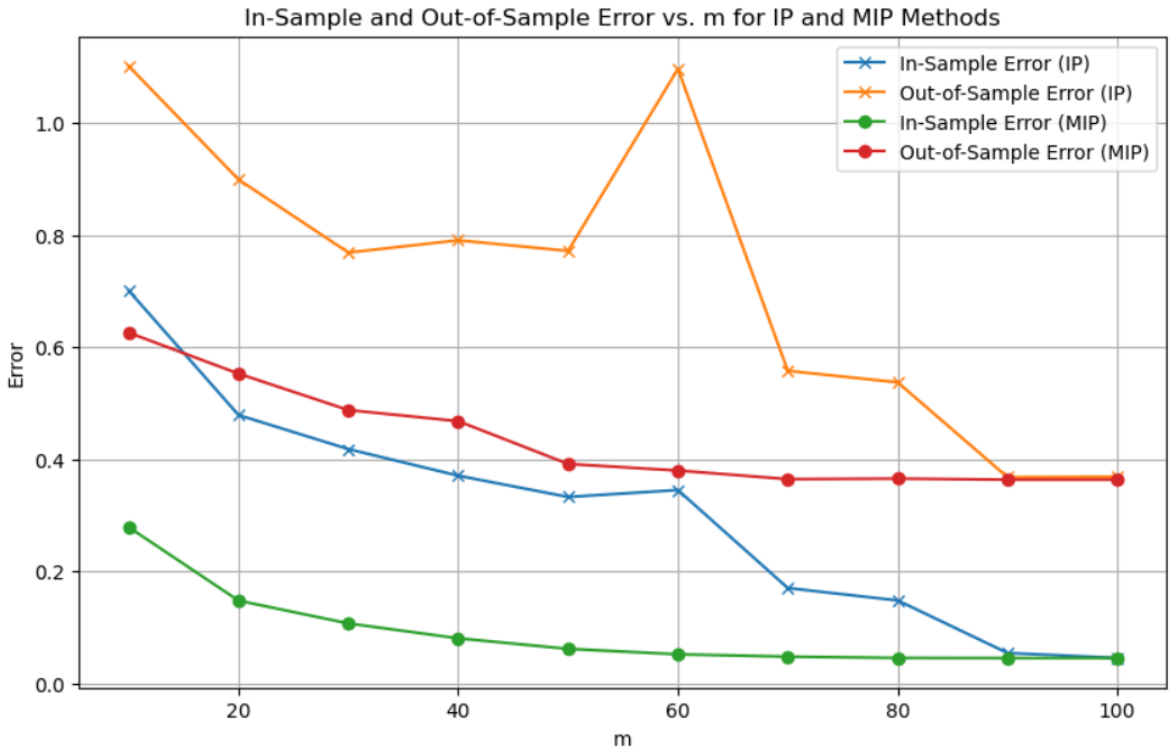
- **Stabilization Point:** After around 60 stocks, both in-sample and out-of-sample tracking errors seem to stabilize. This might indicate a point of diminishing returns in terms of adding more stocks for the purpose of reducing tracking error.

- **Volatile Range:** The out-of-sample error sees a slight increase around 40 stocks and then drops again. This could be due to a variety of factors, such as market anomalies during the out-of-sample period or specific stocks that had significant deviations during that time.

While the in-sample data suggests a consistent benefit to diversifying the portfolio, the out-of-sample data highlights the importance of regular reviews and potential adjustments to ensure the portfolio remains aligned with its benchmark in changing market conditions.

Conclusion

Comparison of both Approaches



Key Observations from the above chart :

- In-Sample Errors: Both the IP and MIP methods see a decline in in-sample error as "m" increases. However, the MIP method consistently has a lower in-sample error than the IP method across the entire range of "m".

- Out-of-Sample Errors: For the IP method, there's a notable spike in out-of-sample error around m=60. This suggests some overfitting or anomaly at that point. On the other hand, the MIP method's out-of-sample error decreases more smoothly and levels off after m=60.

Comparison of IP and MIP Methods:
Generally, the MIP method appears to have a lower error (both in-sample and out-of-sample) compared to the IP method, especially as "m" increases. This could indicate that the MIP method is more robust or better-suited for this particular dataset or problem.

- Approach 1: Stock Selection and Portfolio Evaluation
  Using Integer Programming is relevant if the primary decision is binary (e.g., whether to include a stock in the portfolio or not) and if the problem's constraints and objectives can

be **accurately represented using only the integer decision variables**, which is a major drawback.

- Approach 2: Mixed Integer Programming (MIP) Method
  MIP is **more flexible** and **allows for a combination of integer and continuous decision variables**. This can be useful if the problem **has complex constraints** or if there's a need for both binary decisions and continuous weights (like the proportion of each stock in the portfolio).

Recommendation for Component Stocks and Weights

- Diversifying the portfolio (increasing the number of stocks) results in a better fit to the benchmark within the in-sample data.

- For around 50 stocks, both in-sample and out-of-sample tracking errors seem to stabilize, indicating a point of diminishing returns.

Based on this, a portfolio containing around 30 stocks might be a good starting point as it provides a balance between diversification and manageability. Regular reviews and adjustments are recommended to ensure the portfolio remains aligned with its benchmark in changing market conditions.

Final Verdict

We conducted a comprehensive examination of two methods. In the first approach, stocks were selected before portfolio weights were assigned, whereas in the second approach, stock selection and weight assignment occurred simultaneously within the fund. After our thorough analysis, our team suggests the following:

- To strike the optimal balance between diversification and manageability, we recommend using a value of 'm' corresponding to the selection of 30 highly correlated stocks for the fund.

- Utilize the second approach and employ Mixed Integer Programming to identify the fund configuration that closely aligns with the index, as this minimizes the error.

- Nonetheless, in cases where the user faces time and complexity constraints, opting for Approach 1 might be more suitable, even though it could result in a slight compromise in accuracy.