Optimization Project 2

Portfolio Construction and Optimization Report

Prepared by: Areeba Shah, Rohan Giri, Sarah Lee, and Shashank Rao

Internal Optimization Team

11/03/2024

1. Introduction

Background on Index Funds and Optimization in Fund Management

Index funds replicate a market index by holding stocks that match its composition, offering diversified, low-cost exposure without active management. Full replication, holding all index stocks proportionally, can be costly due to transaction and rebalancing expenses, especially with indices like the NASDAQ-100. A "subset index fund" approach uses fewer stocks ($m < n$) while maintaining close performance to the index by optimizing stock selection and weights to minimize tracking error. Optimization techniques like Integer Programming, Linear Programming, and Mixed Integer Programming help balance tracking precision with cost and manageability.

Problem Statement

The project aims to build an optimized subset index fund that tracks the NASDAQ-100 with minimal tracking error, using 2023 data for construction and 2024 for evaluation. Key tasks include:

1. **Stock Selection**: Integer Programming (IP) is used to select m stocks out of n, maximizing their correlation to the index for diversified exposure.
2. **Weight Optimization**: Linear Programming (LP) minimizes the deviations between portfolio and index returns, with a Mixed Integer Programming (MIP) approach adding sparsity control.

Portfolios are tested iteratively for different m values (10 to 100), analyzing in-sample (2023) and out-of-sample (2024) tracking errors to evaluate performance stability and robustness.

Importance of Optimized Portfolio Selection

Optimized portfolio selection is crucial for efficient index tracking with lower costs. Traditional full-replication index funds incur high transaction costs, especially in volatile markets. Selecting an optimal stock subset reduces these costs while maintaining performance close to the index. Minimizing tracking errors using IP and LP techniques ensures a near-identical risk-return

profile, enhancing fund appeal by balancing accuracy, cost-efficiency, and manageability. This project will recommend the optimal number of stocks (m) for the fund, based on in-sample and out-of-sample performance analysis, to support robust, cost-effective management.

2. Data Description and Preparation

This section outlines the structure and preparation of the provided NASDAQ-100 data files for 2023 and 2024. These datasets form the foundation for stock selection, weight optimization, and performance evaluation, and careful data preparation is crucial to ensure accurate and consistent results in portfolio construction.

Overview of Data (2023 and 2024 NASDAQ-100)

The data comprises two CSV files containing daily price data for the NASDAQ-100 index and its component stocks: one for 2023, used for portfolio construction, and another for 2024, used to evaluate out-of-sample performance. Each dataset is structured with a date column serving as the index for time series analysis, followed by a column for the daily closing price of the NDX (NASDAQ-100 index). The subsequent columns hold the daily closing prices for each of the 100 component stocks, enabling comprehensive analysis for stock selection and performance tracking.

Data Cleaning and Interpolation Methods

To prepare the data for analysis, several cleaning steps were undertaken to address potential issues such as missing values. Accurate data preparation is essential for reliable portfolio optimization. The following steps were applied:

Handling Missing Values:

Missing Values are handled by manually adding missing data into the 2024 file , as vew data points were missing from the csv file from the internet. While we can also fill Missing values in stock price data were addressed using linear interpolation to estimate gaps based on surrounding data, ensuring smooth continuity. Forward and backward filling methods ('ffill' and 'bfill') were applied to handle any remaining missing values at the dataset's edges, ensuring completeness in all columns . we have also filled NANs for ARM stock with price of it on the day of IPO such

that the return on that stock until the day of IPO is zero

```
# Fill missing values by interpolating between values, and handle edge cases
data_2023 = data_2023.interpolate(method='linear', axis=0).fillna(method='ffill').fillna(method='bfill')
data_2024 = data_2024.interpolate(method='linear', axis=0).fillna(method='ffill').fillna(method='bfill')
```

Calculation of Daily Returns

Daily returns for both the index (NDX) and individual stocks were calculated using the percentage change method, which captures the relative price change between consecutive days. This was done by applying `pct_change()` to all relevant columns and removing any NaN values, providing a basis for correlation calculations and tracking error evaluations.

$R_t = \frac{P_t - P_{t-1}}{P_{t-1}}$, where R_t is the return on day t, and P_t and P_{t-1} are the prices on days t and t-1, respectively.

```
# Compute daily returns
returns_2023 = data_2023.iloc[:, 1:].pct_change().fillna(0)
returns_2024 = data_2024.iloc[:, 1:].pct_change().fillna(0)

index_returns_2023 = returns_2023.iloc[:, 0]
stock_returns_2023 = returns_2023.iloc[:, 1:]

index_returns_2024 = returns_2024.iloc[:, 0]
stock_returns_2024 = returns_2024.iloc[:, 1:]
```

Construction of the Correlation Matrix (ρ)

The correlation matrix(ρ), computed from 2023 stock returns data, captures inter-stock relationships by showing the similarity (or dissimilarity) between each pair of NASDAQ-100 stocks. High correlation values indicate similar behavior, while low or negative values suggest diversification potential. This matrix is essential for the integer programming model, which selects a subset of stocks that maximizes similarity to effectively replicate the index's performance.

```
# Compute correlation matrix (ρ)
rho = stock_returns_2023.corr().values
```

3. Methodology

In this project, we employed a multi-step optimization approach to construct a portfolio that accurately tracks the NASDAQ-100 index using a subset of stocks. The primary goal was to minimize tracking errors between the portfolio and the index over time, focusing on both in-sample (2023) and out-of-sample (2024) data. Our approach involved:

1. **Integer Programming (IP) and Linear Programming (LP) :** Integer Programming was used to select m stocks from the NASDAQ-100 that best represent the index by maximizing similarity through the correlation matrix (ρ). Linear Programming (LP) then determined optimal weights for these stocks to minimize tracking error, ensuring the portfolio closely matched the index returns.
2. **Mixed Integer Programming (MIP)** with a Big-M constraint to impose sparsity by selecting only a limited number of non-zero weights, making the portfolio more practical and cost-effective.

3.1 Integer Programming for Stock Selection & Linear Programming for Weight Optimization

3.1.1 Integer Programming for Stock Selection

The initial phase used Integer Programming (IP) to select m stocks from the NASDAQ-100 index, maximizing similarity to the overall index via the correlation matrix (ρ). This step ensured the chosen stocks could closely replicate the index's performance.

Decision Variables:

- **Binary Variable y_j**: Indicates whether stock j is included in the portfolio (y_j = 1 if selected, otherwise 0).
- **Binary Variable x_ij**: Represents whether stock j is the most similar to stock i in the index (x_ij = 1 if stock j is the best representative of stock i, otherwise 0).

Objective: Maximize the total similarity score across all stocks, formulated as:
$\text{Maximize} \quad \sum_{i=1}^{n} \sum_{j=1}^{n} \rho_{ij} x_{ij}$ where ρ_ij is the correlation coefficient between stocks i and j,

capturing the degree of similarity between their returns

Constraints:

- **Selection Constraint**: Ensures that exactly m stocks are chosen, given by: $\sum_{j=1}^{n} y_j = m$
- **Representation Constraint**: Each stock i in the index is represented by one stock j in the selected portfolio: $\sum_{j=1}^{n} x_{ij} = 1 \quad \forall i$
- **Linkage Constraint**: Ensures that a stock can only represent others if it is selected in the portfolio: $x_{ij} \leq y_j \quad \forall i, j$

Example Calculation with m = 5: For an initial demonstration, we set m=5 to select a portfolio of 5 stocks. The IP model was solved using Gurobi, with the following process:

Input: The correlation matrix (ρ) derived from the 2023 daily returns for the NASDAQ-100 stocks.

Optimization Process: The Gurobi optimizer was used to solve the IP model, selecting 5 stocks that maximize the total correlation with the rest of the index.

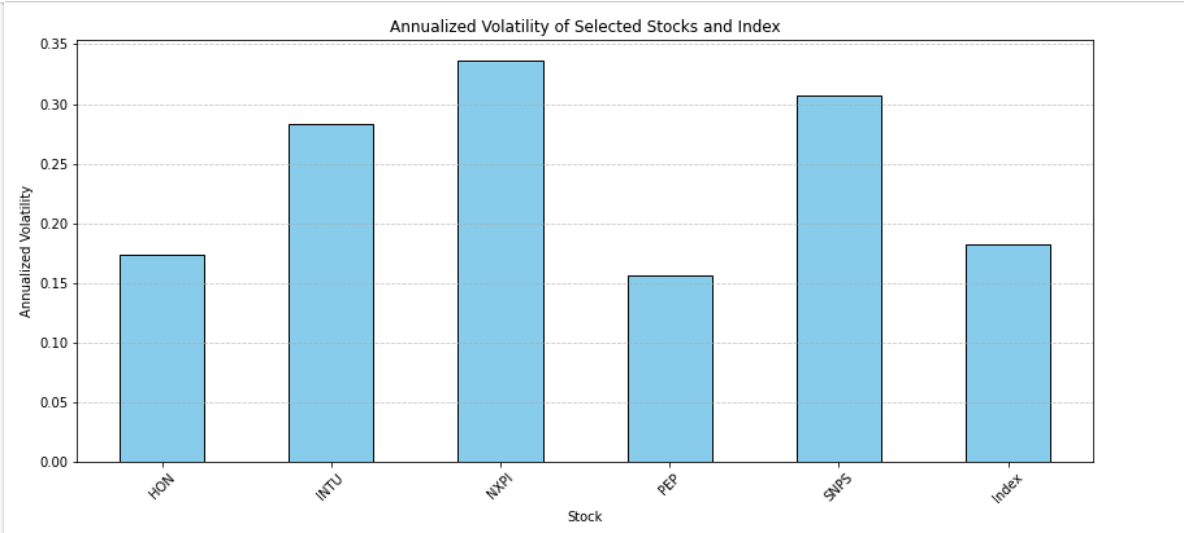
```
# Step 2: Solve the IP for selecting 5 stocks
def select_stocks(m, rho):
    n = rho.shape[0]
    model = Model()

    # Decision variables
    y = model.addVars(n, vtype=GRB.BINARY, name="y")
    x = model.addVars(n, n, vtype=GRB.BINARY, name="x")

    # Objective: Maximize similarity
    model.setObjective(quicksum(rho[i, j] * x[i, j] for i in range(n) for j in range(n)), GRB.MAXIMIZE)

    # Constraints
    model.addConstr(quicksum(y[j] for j in range(n)) == m)  # Select exactly m stocks
    for i in range(n):
        model.addConstr(quicksum(x[i, j] for j in range(n)) == 1)  # Every stock has a representative
        for j in range(n):
            model.addConstr(x[i, j] <= y[j])  # A stock can only be a representative if selected

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

    selected_stocks = [j for j in range(n) if y[j].x > 0.5]

    return selected_stocks
```

The results were as follows:

- **Selected Stock Indices**: [47, 51, 72, 81, 88]

- **Selected Stock Names**: ['HON', 'INTU', 'NXPI', 'PEP', 'SNPS']

These stocks were chosen as the optimal subset of the NASDAQ-100 index for representing the index with minimal tracking error. We can Also visualize how volatile the stocks are compared to the index fund from the volatility graph, obviously this helpful as long as the values of m are small. This does help me giving an intuition of the mimicking of the index fund is just a coincidence or it is actually correlated to index



3.1.2 Linear Programming (LP) for Weight Optimization

Following stock selection, Linear Programming (LP) was applied to assign optimal weights to the selected m stocks, minimizing the tracking error relative to the NASDAQ-100 index. The goal was to match the index returns for the in-sample period (2023) and evaluate performance for the out-of-sample period (2024).

Objective: Minimize the sum of absolute deviations between the portfolio returns and the index returns, formulated as: $\text{Minimize} \quad \sum_{t=1}^{T} \text{dev}_t$

where dev_t represents the deviation between the portfolio return and the index return at time t.

Constraints:

- **Deviation Constraints**: For each time period t, deviations are constrained by:

$$\text{dev}_t \geq q_t - \sum_{i=1}^{m} w_i r_{it}$$

$$\text{dev}_t \geq \sum_{i=1}^{m} w_i r_{it} - q_t$$

where q_t is the index return at time t, and w_i is the weight of stock i in the portfolio.

- **Weight Sum Constraint**: The sum of all weights must equal 1, ensuring the portfolio is fully invested: $\sum_{i=1}^{m} w_i = 1$
- **Non-Negativity Constraint**: All weights must be non-negative, ensuring no short positions.

Weight Calculation Results for Various m Values (5, 10, …, 100)

Using the linear programming setup described above, we solved for optimal weights for portfolio sizes m=5,10,20,…,100. The selected stocks for each m were determined through the integer programming process based on maximizing correlation similarity, and these stocks were then used in the weight optimization model.

```
# Step 3: Construct weights to match the index return with robust indexing
def optimize_weights(selected_stocks, stock_returns, index_returns):
    T = len(index_returns)  # Number of time periods
    m = len(selected_stocks)  # Number of selected stocks

    # Create LP model for weight optimization
    model = Model()
    weights = model.addVars(m, lb=0, name="w")  # Non-negative weights
    deviations = model.addVars(T, lb=0, name="dev")  # Deviations for absolute differences

    # Convert index returns and stock returns to NumPy arrays for easier operations
    index_returns_array = index_returns.to_numpy()
    selected_stock_returns = stock_returns.iloc[:, selected_stocks].to_numpy()

    # Objective: Minimize sum of absolute deviations
    model.setObjective(quicksum(deviations[t] for t in range(T)), GRB.MINIMIZE)

    # Constraints for each time period
    for t in range(T):
        portfolio_return_t = quicksum(weights[i] * selected_stock_returns[t, i] for i in range(m))
        model.addConstr(deviations[t] >= index_returns_array[t] - portfolio_return_t)
        model.addConstr(deviations[t] >= portfolio_return_t - index_returns_array[t])

    # Sum of weights should be 1
    model.addConstr(quicksum(weights[i] for i in range(m)) == 1)

    # Optimize the model
    model.Params.OutputFlag = 0
    model.optimize()

    # Extract optimized weights
    optimized_weights = [weights[i].x for i in range(m)]
    return optimized_weights
```

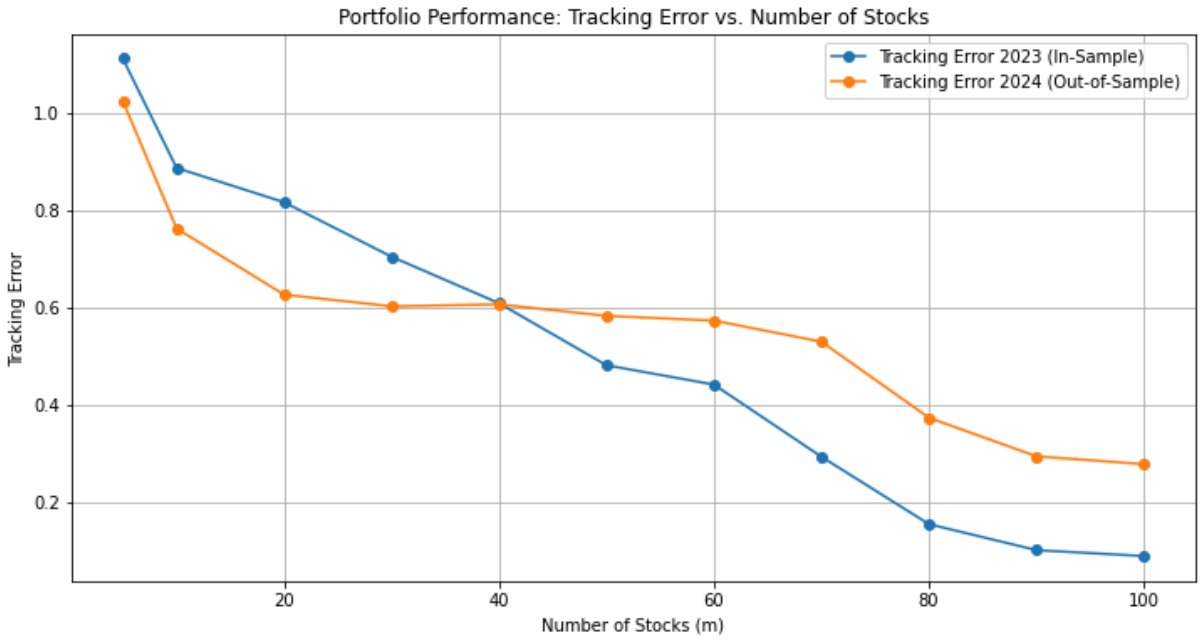
In-Sample and Out-of-Sample Tracking Errors

- **In-Sample Tracking Error (2023)**:

Tracking error decreased as m increased, showing that larger portfolios better align with the index. For instance, at m=10, the error was high, but it dropped significantly by m=80.

- **Out-of-Sample Tracking Error (2024)**:

Out-of-sample tracking errors also decreased as mmm increased but were higher than in-sample, reflecting market unpredictability. Errors minimized around m=80 to m=100, indicating robust performance for portfolios of that size.



Observation

Observations show diminishing returns in tracking error reduction around m = 70 , making m = 70 to m = 90 an optimal range for balancing accuracy and portfolio size. Consistent error

reduction across 2023 and 2024 data confirms the stability of the linear programming approach.

Conclusion

In 2023, the portfolios demonstrated strong in-sample performance, particularly with larger mmm values, due to optimization using 2023 data. However, in 2024, out-of-sample performance varied, with smaller mmm portfolios sometimes achieving lower tracking errors than in-sample, indicating that these simpler portfolios coincidentally aligned with 2024 trends. For mmm values exceeding 40, tracking errors for both in-sample and out-of-sample data converged, suggesting that larger, diversified portfolios provided stable performance across different years.

3.2 Mixed Integer Programming (MIP) with Big-M Constraint for Sparse Portfolio Selection

The final method, Mixed Integer Programming (MIP), was implemented to enforce sparsity by constraining the number of non-zero weights. The Big-M constraint ensures that only a specified number of stocks (m) contribute to the overall return, thus achieving a more manageable portfolio size while still closely tracking the index. This approach is particularly useful for creating a practical and cost-effective portfolio with fewer active positions, reducing rebalancing costs and simplifying portfolio management.

Objective: Minimize the sum of absolute deviations between the portfolio return and the index return, across all time periods t:

$\min \sum_{t=1}^{T} \left| q_t - \sum_{j=1}^{n} w_j r_{jt} \right|$, where q_t is the return of the index at time t, r_jt is the return of stock j at time t, w_j is the weight of stock j in the portfolio.

Constraints:

- **Weight Sum Constraint**: Ensures full investment by requiring that the sum of portfolio weights equals 1.

$$\sum_{j=1}^{n} w_j = 1$$

- **Big-M Constraint**: Controls which stocks can have non zero weights. The binary variable y_j determines stock inclusion, with y_j =1 if stock j is selected and y_j=0 otherwise. The Big M constant, set at 1, imposes an upper bound on w_j based on y_j:

$$w_j \leq M \cdot y_j \quad \forall j$$

This constraint ensures that w_j can only be positive if y_j = 1, effectively creating a sparse portfolio with exactly m non-zero weights.

- **Non-negativity Constraint**: Ensures that weights cannot be negative, preventing short positions in the portfolio

$w_j \geq 0$ for all j.

- **Cardinality Constraint**: Limits the portfolio to exactly m selected stocks with non zero weights.

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

This constraint directly enforces the desired sparsity level or the exact number of stocks in the portfolio.

Time Limit Setting: To manage computational complexity, a time limit of 3600 seconds (1 hour) was set for each MIP optimization run. This ensured that the solution process was feasible for larger values of m and allowed us to obtain results within a reasonable timeframe.

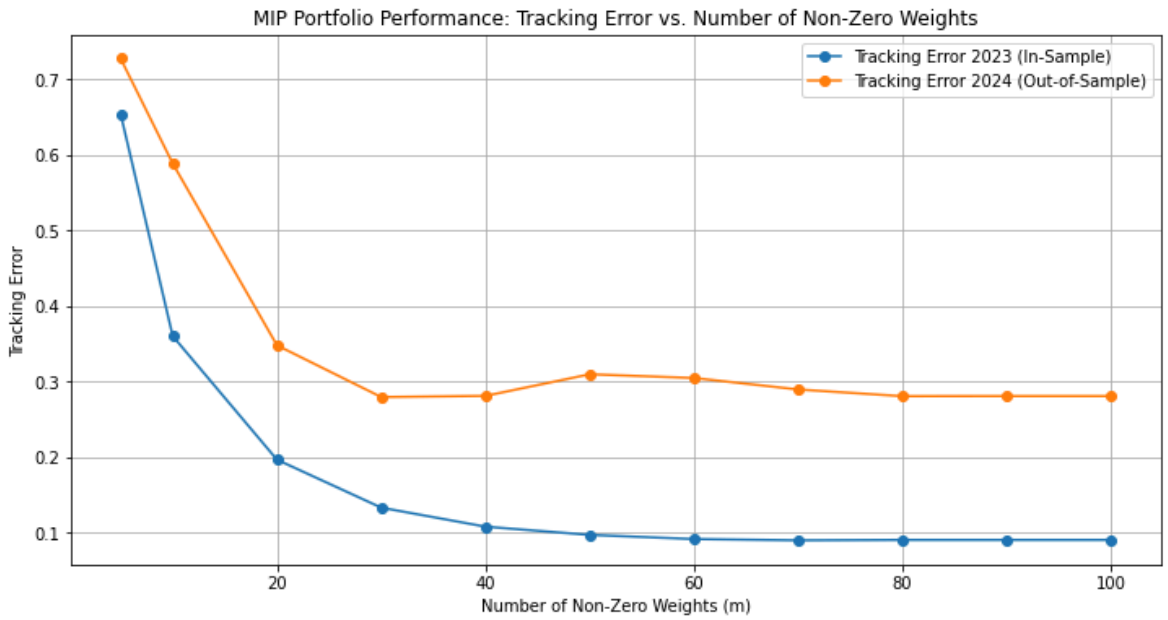
```python
def mip_weight_selection(m, stock_returns, index_returns, time_limit):
    T, n = stock_returns.shape

    # Convert data to NumPy arrays for compatibility with Gurobi
    stock_returns_array = stock_returns.to_numpy()
    index_returns_array = index_returns.to_numpy()

    # Create Gurobi model
    model = Model()
    model.Params.TimeLimit = time_limit  # Set time limit for Gurobi
    model.Params.OutputFlag = 0  # Suppress solver output

    # Variables
    w = model.addVars(n, lb=0, ub=BIG_M, name="w")  # Weight variables
    y = model.addVars(n, vtype=GRB.BINARY, name="y")  # Binary variables
    deviations = model.addVars(T, lb=0, name="dev")  # Deviation variables

    # Objective: Minimize sum of absolute deviations
    model.setObjective(quicksum(deviations[t] for t in range(T)), GRB.MINIMIZE)

    # Constraints to model absolute deviations
    for t in range(T):
        # Calculate the portfolio return for time t using NumPy arrays
        portfolio_return_t = quicksum(w[i] * stock_returns_array[t, i] for i in range(n))

        # Add absolute deviation constraints
        model.addConstr(deviations[t] >= index_returns_array[t] - portfolio_return_t)
        model.addConstr(deviations[t] >= portfolio_return_t - index_returns_array[t])

    # Sum of weights must equal 1
    model.addConstr(sum(w[i] for i in range(n)) == 1)

    # Big-M constraint: If y_i = 0, then w_i must be 0
    for i in range(n):
        model.addConstr(w[i] <= y[i] * BIG_M)

    # Sum of binary variables must equal m (number of non-zero weights)
    model.addConstr(quicksum(y[i] for i in range(n)) == m)

    # Optimize the model
    model.optimize()

    #----
    portfolio_weights = {stock_returns.columns[i]: w[i].x for i in range(n) if w[i].x > 0.001}
    selected_stocks = [stock_returns.columns[i] for i in range(n) if y[i].x > 0.5]
    return portfolio_weights, selected_stocks, model.ObjVal
```

Results:

The following table summarizes the tracking error results and computation time for various values of m in the MIP model with the Big-M constraint:

	m	tracking_error_2023	tracking_error_2024	time_taken
0	5	0.654442	0.727812	1506.620000
1	10	0.359938	0.588651	3600.605532
2	20	0.196127	0.347699	3600.401436
3	30	0.132593	0.279135	3600.388991
4	40	0.107376	0.280702	3600.422608
5	50	0.096482	0.309370	3600.534892
6	60	0.090990	0.304401	3600.805344
7	70	0.089284	0.289187	256.745197
8	80	0.089816	0.280314	2.315901
9	90	0.089828	0.280441	1.689465
10	100	0.089828	0.280441	0.292216

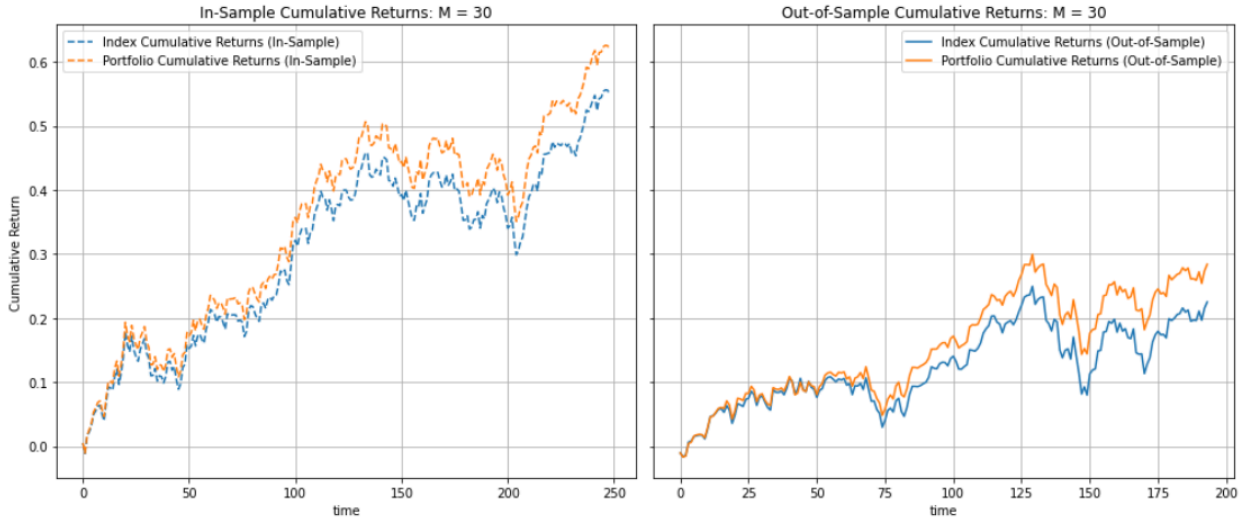
- **Tracking Error Trends**: Tracking error decreases as m increases, reaching a minimum around m = 80 to m = 100. This suggests that adding more stocks improves tracking accuracy, but the benefits plateau beyond m = 30.
    - **In-Sample (2023) Tracking Error**: This measures how closely the 2023-constructed portfolio aligns with the index, with tracking error decreasing as m increases (e.g., from 0.8861 at m=10 to 0.0881 at m=100).
    - **Out-of-Sample (2024) Tracking Error**: Evaluates the 2023 portfolio's performance with 2024 data, showing decreasing tracking errors as m grows, stabilizing around m=80, beyond which further additions offer minimal benefits.
- **Computation Time Efficiency**: For m = 30 and above, the computation time reduces significantly, indicating that the model becomes easier to solve as more stocks are included, likely due to fewer restrictive constraints in the solution space.



Optimal Portfolio Size and Diminishing Returns

Both in sample and out of sample results confirm that the benefits of adding more stocks diminish beyond m=80. This supports the recommendation from the MIP analysis to use an m range of 30-40 for a sparse, efficient portfolio, or up to 80 for enhanced accuracy with minimal additional complexity.

```
Running MIP for m = 30...
Set parameter TimeLimit to value 3600
m = 30: Tracking Error 2023 = 0.1325927596321318, Tracking Error 2024 = 0.2791351168659314, Time Taken = 3600.39 seconds
```



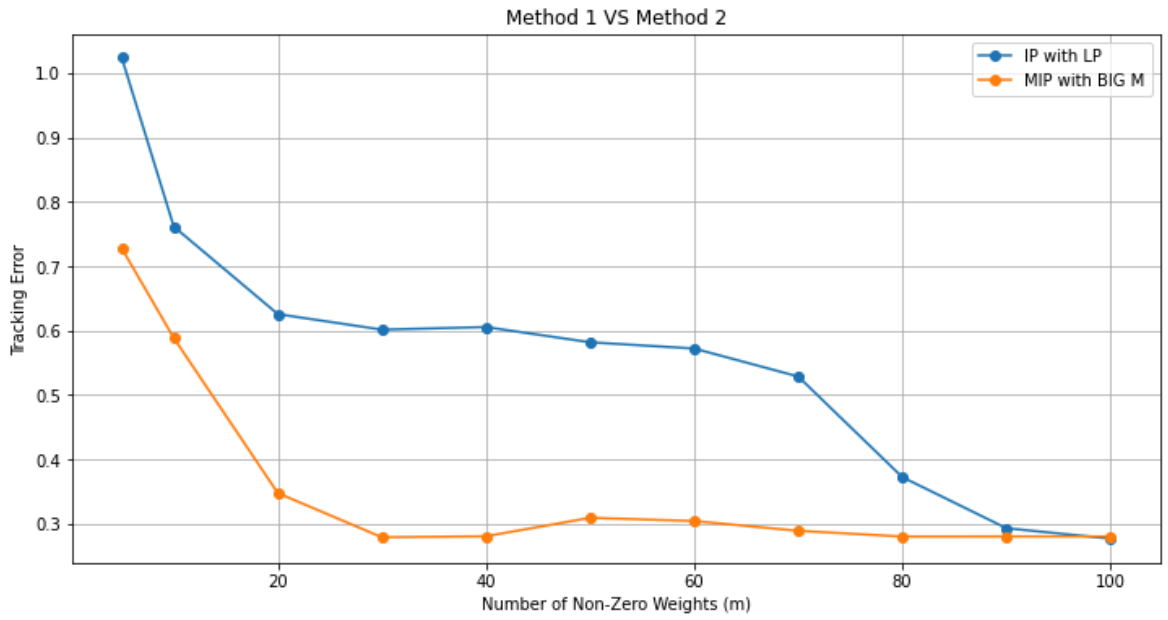
4. Comparison of Integer Programming and Mixed Integer Programming Approaches

1. **Integer Programming (IP)**:

This method selects m stocks from the NASDAQ-100 using a correlation matrix (ρ) to maximize similarity and ensure diversification. While effective at minimizing tracking error, the IP approach does not enforce weight sparsity, leading to some redundancy.

2. **Mixed Integer Programming (MIP) with Big-M Constraint**:

The MIP approach with the Big-M constraint uses binary variables to enforce sparsity by limiting non-zero weights. It achieved similar tracking errors to the IP method but with higher computational complexity, especially for small m. For m around 70-80, computation time dropped, and tracking error stabilized.



Conclusion: The IP approach is simpler for lower m values, but MIP provides better sparsity control, making it ideal for larger portfolios with non-zero weight constraints.

Practical Implications for Portfolio Management

The findings underscore the practical need to balance tracking accuracy with portfolio management costs. Here's how the results translate into actionable steps:

1. **Optimizing Portfolio Size (m)**:

A 30-40 stock portfolio offers high tracking accuracy and manageable trading costs, minimizing tracking error and aligning well with the index.

2. **Choice of Optimization Method**:

IP is ideal for small-to-medium portfolios focused on high tracking accuracy without sparsity constraints. MIP with Big-M is better for maintaining sparsity, especially for larger m where computational load decreases.

3. **Tracking Error Management**:

Setting m within the 20-30 range ensures stable, low tracking error and aligns the portfolio closely with index performance.

5. Conclusion and Recommendations

Summary of Key Findings

1. **Optimal Portfolio Size**: An optimal portfolio size of 20-30 stocks provides a near-optimal balance between tracking error and manageability.
2. **Tracking Accuracy**: As m increases, tracking error decreases, but diminishing returns are observed beyond m = 30.
3. **Optimization Method**: Both IP and MIP with Big-M are effective, but MIP offers added flexibility in controlling weight sparsity, especially valuable for larger portfolios.for example to get similar tracking accuracy we need 80-90 stocks in linear programming while with MIP we can better accuracy with just 20 stocks

Suggested Rebalancing Frequency and Method

1. **Frequency**: Quarterly rebalancing is advised to ensure the portfolio maintains alignment

with the index while minimizing transaction costs.

2. **Method**: Use the MIP with Big-M constraint for rebalancing, particularly if maintaining sparsity in weights is essential and it allows efficient adjustments without overcomplicating the portfolio structure.

Considerations for Future Portfolio Optimization

1. **Incorporate Additional Constraints**: In future analyses, consider adding sector or industry constraints to maintain diversification across economic sectors and further reduce concentration risk.
2. **Dynamic Portfolio Size (m)**: Conduct sensitivity analysis on m in response to market volatility or changing economic conditions. This dynamic adjustment can improve robustness in fluctuating markets.
3. **Alternative Similarity Metrics**: While the correlation matrix provided good results, testing alternative similarity metrics may enhance stock selection accuracy.