GROUP 3

Equity Money Management

SELECTION OF STOCKS AND WEIGHTS TO TRACK NASDAQ-100

VAISHNAVI GANESH [VG23798]
SANJHANA RANGARAJ [SR55737]
NISHANT KUSHWAHA [NSK779]
SANYAM JAIN [SJ33448]

Table of Contents

Introduction

From a practical standpoint, addressing the intricate challenge of crafting an optimal investment portfolio that closely mirrors a benchmark index carries profound implications for the financial industry. Within this dynamic landscape, financial businesses are continually pressed to deliver results that meet investor expectations and compete effectively in the marketplace. When formulated, this optimal portfolio contributes to cost-efficiency, reducing operational expenses and enhancing overall profitability. In essence, this challenge significantly shapes investment performance and underscores the enduring viability and growth prospects of financial enterprises.

The primary objective is to create a portfolio that effectively mirrors the performance of a benchmark index, such as the NASDAQ-100, while considering practical constraints and resource limitations. To address this optimization task, we employ two methods – 2 step approach and mixed-integer programming (MIP) approach, leveraging advanced mathematical techniques to determine the ideal number of component stocks to include in the portfolio. The critical decision of 'm,' or the number of stocks, carries significant weight, as it directly impacts the portfolio's ability to track the index's returns accurately.

To thoroughly evaluate this index approximation technique, the project analyzes the impact of the number of stocks 'm' on replication accuracy. Specifically, the optimization modeling exercise is repeated across varying values of 'm' - from smaller focused sets of stocks to larger diversified sets. This tests the concept of diminishing returns, helping identify at what point incrementally adding more stocks fails to meaningfully improve tracking performance compared to the NASDAQ-100 target. Robustness is incorporated by basing the optimization analysis on recent stock data from 2019, grounding the selection process in current real-world market dynamics.

Finally, the optimized portfolios for each 'm' value are evaluated on out-of-sample stock return data from 2020, providing insights into true realized performance. This assesses how effectively the optimized portfolios constructed by the model can replicate NASDAQ-100 returns under different market conditions from the optimization period. The results quantify the trade-offs between portfolio complexity in terms of number of stocks versus accuracy in matching the performance of the broader index. This aids in determining the ideal balance when constructing an index-approximating portfolio for practical implementation.

Approach Followed

Similarity Matrix

The similarity matrix is derived from stock correlations, where we assess the relationships between various stocks. We compute the percentage change for all stocks over the years, and subsequently, we generate a correlation matrix based on this percentage change.

	ATVI	ADBE	AMD	ALXN	ALGN
ATVI	1.000000	0.399939	0.365376	0.223162	0.216280
ADBE	0.399939	1.000000	0.452848	0.368928	0.363370
AMD	0.365376	0.452848	1.000000	0.301831	0.344252
ALXN	0.223162	0.368928	0.301831	1.000000	0.332433
ALGN	0.216280	0.363370	0.344252	0.332433	1.000000

It's important to highlight that alternative metrics can be employed for constructing the similarity matrix, rather than relying solely on correlation based on percentage change. With the correlation matrix at our disposal, we proceed to construct our portfolio using two distinct methodologies.

1. Two-step approach
2. MIP approach

2-Step Approach

In this approach, we use two Gurobi models that are independent of each other to first find the stocks that we want to include in our portfolio and then use another model to assign the weights for those selected stocks.

Stock Selection

In this approach we first select the stocks (represented by m) that go into the portfolio, as described below:

1. Consider m stocks, we should therefore find the m best stocks that closely tracks the index
2. Consider y_j as the binary decision variable that tells us whether the j^{th} stock is present in the portfolio or not
3. Consider x_{ij} to be the binary decision variables that tells us if stock j is the best representation of stock i (i.e.) the correlation between stock j and stock i is the highest positive value
4. Add constraints such that:
    a. There should be exactly m stocks in the portfolio

   b. Each stock should have only one best representative (i.e.) out of all the possible combinations of correlations, there should be only 100 entries with the value of 1 representing the best correlation with a particular stock.
   c. The j^{th} stock in the index that best represents stock i, is considered to be the best only if it is present in the portfolio
5. The objective of the problem is to increase the similarity between the 100 stocks in the index and the 5 stocks in the portfolio

Using the correlation matrix and the decision variables to add a stock or not (y) and whether a particular stock j is the best representation of the stock I (x_{ij}), we run a gurobi model to optimize for the best 5 stocks that represent the index by maximizing the correlation of the returns of the portfolio and the index. The objective function and the constraints are mathematically represented as:

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

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

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

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

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

These are translated into the code as follows:

```
def stockSelection(m, stock_names, correlation_dict): # Function to select the stocks in the
2_step prcess
    # Some minute sanity checks
    assert m <= len(stocks2019_names)
    assert len(stocks2019_names) <= len(correlation_dict)

    stock_selection_model = gp.Model()
    stock_selection_model.ModelSense = gp.GRB.MAXIMIZE

    x = stock_selection_model.addVars(stock_names,stock_names,vtype='B') # similarity variable
    y = stock_selection_model.addVars(stock_names,vtype='B') # limiting variable

    # Constraint to select just m stocks
    sum_y = 0
    for i in stock_names:
        sum_y += y[i]
    stock_selection_model.addConstr(sum_y == m)

    # Constraint to get similar stocks
    for i in stock_names:
        sum_x = 0
        for j in stock_names:
```

```
            sum_x += x[i, j]
        stock_selection_model.addConstr( sum_x == 1)

    # Constraint to deselect dissimilar stock / to make sure that the best representation is
in the portfolio
    for i in stock_names:
        for j in stock_names:
            stock_selection_model.addConstr(x[i, j] <= y[j])

    # Setting the objective function to maximize the correlation
    sum_obj = 0
    for i in stock_names:
        for j in stock_names:
            sum_obj += correlation_dict[i][j] * x[i, j]
    stock_selection_model.setObjective(sum_obj)

    # Setting model parameters
    stock_selection_model.Params.OutputFlag = 0
    stock_selection_model.optimize()

    # Converting selected stocks to a list
    selectedStocks = []
    for i in stock_names:
        for j in stock_names:
            if x[i,j].X == 1 and j not in selectedStocks:
                selectedStocks.append(j)

    return stock_selection_model, selectedStocks
```

After running the above code, we get the 5 best stocks to be included in the portfolio as follows:

```
The maximized correlation value from the objective function is: 54.84
The selected stocks are: ['MSFT', 'VRTX', 'MXIM', 'LBTYK', 'XEL']
```

Portfolio Weights

Now that we have the stocks selected, the next goal is to select the continuous variables of the precise weight or percentage of the portfolio to allocate to each chosen stock.

1. Consider the return of a given stock at a given time and multiply it with the weight of that stock.
2. We will have to consider the return of the index at that particular time as well and minimize the difference between these two values, but this will lead to a non-linear approach and therefore to make the solution linear, we consider separate decision variables for a time t and minimize the sum of those decision variables at that time, and add constraints to make sure that the decision variable is greater than or equal to the difference of the return and the sum-product of the weights and stock returns.
3. To make sure that we solve for the absolute value of the difference, we also add a constraint that the decision variable is greater than or equal to the negative value of the difference as well

4. This will make sure that we minimize over y_1, and also make sure that it is larger than or equal to the difference and the negative of the difference and since we are minimizing, the value of y_1 will automatically become equal to the absolute value of the difference

The mathematical representation of the objective function and the constraints are as follows:

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

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

$$w_i \geq 0.$$

We add decision variables to determine the weights given to each stock in the portfolio with the constraints that the weights assigned to all the stocks should add to 1 and we add dummy decision variables to add constraints such that these variables are greater than the absolute value of the difference between the index return and the portfolio return and to make sure that absolute value is captured, we add 2 constraints that the dummy decision variable should be greater than the difference as well as the negative value of the difference.

The code in which this above are implemented are as follows:

```
def portfolioWeights(m,selectedStocks, stock_names, stocks_indices_old, stock_indices_new, \
                     stock_returns_old, stock_returns_new, q, q_new, timeLimit = 300,
save=False): # Function to get the model weight (2-step)
    # Minute sanity checks
    assert len(q.columns) == 1
    assert len(q_new.columns) == 1

    # Converting actual NDX returns provided to a list
    q = list(q[list(q.columns)[0]].values)
    q_new = list(q_new[list(q_new.columns)[0]].values)

    # Intializing the model and setting the model parameters
    weight_model = gp.Model()
    weight_model.ModelSense = gp.GRB.MINIMIZE
    weight_model.Params.OutputFlag = 0
    weight_model.Params.TIME_LIMIT = timeLimit

    # Decision Variables
    weight_SelectedStocks = weight_model.addVars(selectedStocks, ub=[1]*len(selectedStocks)) #
variable for weight of selected stock
    u = weight_model.addVars(stocks_indices_old) # dummy variable to solve non-linearities

    # Constraint for sum of weight = 1
    sum_weighgts = 0
    for i in selectedStocks:
        sum_weighgts += weight_SelectedStocks[i]
    weight_model.addConstr(sum_weighgts == 1)
```

```
    # Constraints to minimize non linearity like min -> |X-1|
    for j in stocks_indices_old:
        sum_weight_return = 0
        for i in selectedStocks:
            sum_weight_return += weight_SelectedStocks[i] * stock_returns_old[i][j]
        weight_model.addConstr(u[j] >= (q[j-1] - sum_weight_return))
        weight_model.addConstr(u[j] >= -(q[j-1] - sum_weight_return))

    # Setting the objective to minimize the absolute errors
    sum_obj = 0
    for j in stocks_indices_old:
        sum_obj += u[j]
    weight_model.setObjective(sum_obj)

    weight_model.optimize()

    # Calculating returns from model for the old data
    returns_old = []
    for j in stocks_indices_old:
        sum_return = 0
        for i in selectedStocks:
            sum_return += weight_SelectedStocks[i].X * stock_returns_old[i][j]
        returns_old.append(sum_return)

    # Calculating returns from model for the new data
    returns_new = []
 for j in stock_indices_new:
        sum_return = 0
        for i in selectedStocks:
            sum_return += weight_SelectedStocks[i].X * stock_returns_new[i][j]
        returns_new.append(sum_return)

    # Saving the calculated returns as per user's demand
    if save:
        returns_old_save = []
        for j in stocks_indices_old:
            sum_return = 0
            for i in selectedStocks:
                sum_return += weight_SelectedStocks[i].X * stock_returns_old[i][j]
            returns_old_save.append([j, q[j-1], sum_return])
        returns_old_save = pd.DataFrame(returns_old_save, columns=['Index', 'Index Return',
'Portfolio Return'])
        returns_old_save.to_csv(f'Returns_old_m-{m}_2step.csv', index=False)

        returns_new_save = []
        for j in stock_indices_new:
            sum_return = 0
            for i in selectedStocks:
                sum_return += weight_SelectedStocks[i].X * stock_returns_new[i][j]
            returns_new_save.append([j, q_new[j-1], sum_return])
        returns_new_save = pd.DataFrame(returns_new_save, columns=['Index', 'Index Return',
'Portfolio Return'])
        returns_new_save.to_csv(f'Returns_new_m-{m}_2step.csv', index=False)

    # Saving the weights in a dataframe
    selectedStocks_weights = pd.DataFrame(index=stock_names)
    selectedStocks_weights[f'm_{m}'] = np.nan
    for i in selectedStocks:
        selectedStocks_weights[f'm_{m}'].loc[i] = weight_SelectedStocks[i].X
    selectedStocks_weights = selectedStocks_weights[selectedStocks_weights[f'm_{m}'].notna()]
```

```
    # Evaluating the model on the old and the new data
    model_error = weight_model.objVal
    new_return_error = sum(abs(np.array(q_new) - np.array(returns_new)))
    old_return_error_rmse = np.sqrt(np.mean((np.array(q) - np.array(returns_old))**2))
    new_return_error_rmse = np.sqrt(np.mean((np.array(q_new) - np.array(returns_new))**2))

    return weight_model, selectedStocks_weights, model_error, old_return_error_rmse,
new_return_error, new_return_error_rmse
```

The result we get using the two-step approach when selecting 5 stocks as mentioned is shown below. Microsoft (MSFT) is allocated the highest weight of over 50% of the total portfolio value. The other 4 positions were sized between 5-15%, resulting in a portfolio with performance tightly correlated to MSFT while still maintaining a degree of diversification.

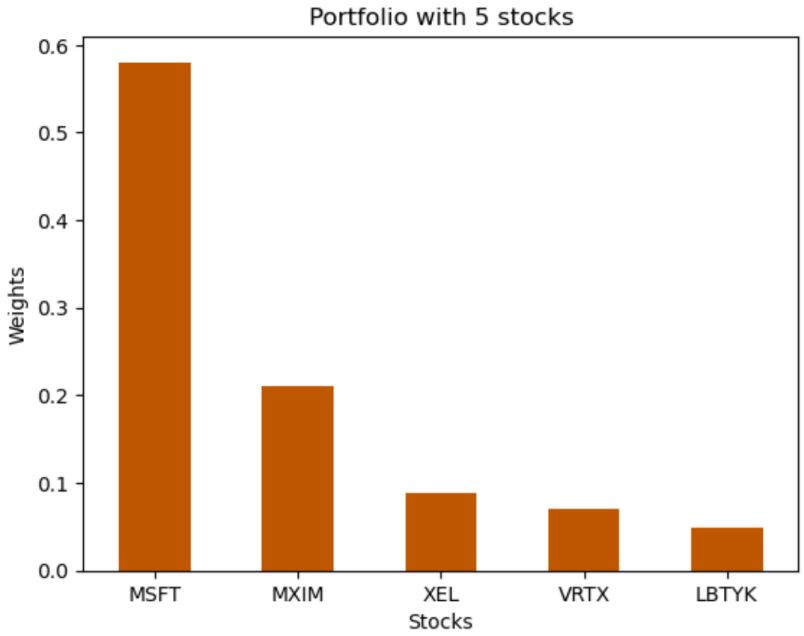

Fig 1 Weightage of different stocks when we choose m=5

Now that we have the selected stocks and the weights, we calculate the portfolio returns and the index returns to see how the portfolio return has tracked the index and to do that, we calculate 2 different types of errors:

1. Return error – The sum of absolute differences between the portfolio return and the index return
2. RMSE – The error between the portfolio return and the index return taken on a daily basis, squared, taking a mean of that and taking the square root of that value

We calculate these 2 errors for m = 5 and all the other portfolio sizes and we get the following results:

	m	#StocksSelected	Selection_Obj_Val	2019_return_error	2019_return_RMSE	2020_return_error	2020_return_RMSE
0	5	5	54.839907	0.789178	0.004400	1.112437	0.006248
1	10	10	59.331930	0.701218	0.003790	1.102404	0.006221
2	15	15	63.268591	0.582276	0.003218	1.264689	0.007145
3	20	20	66.648075	0.478836	0.002653	0.899598	0.005088
4	25	25	69.764755	0.442966	0.002472	0.805065	0.004772
5	30	30	72.696797	0.418015	0.002399	0.769110	0.004666
6	35	35	75.505895	0.382997	0.002185	0.766774	0.004419
7	40	40	78.259157	0.370517	0.002157	0.791047	0.004471
8	45	45	80.892084	0.343944	0.002102	0.760495	0.004267
9	50	50	83.316268	0.332540	0.001968	0.772100	0.004311
10	55	55	85.636309	0.312249	0.001859	0.796320	0.004528
11	60	60	87.877505	0.344890	0.002046	1.097304	0.005957
12	65	65	90.023438	0.177765	0.001060	0.558750	0.002880
13	70	70	92.062402	0.169824	0.001012	0.557854	0.002955
14	75	75	94.007810	0.158949	0.000977	0.561073	0.003022
15	80	80	95.728853	0.147683	0.000884	0.537323	0.002923
16	85	85	97.212076	0.110861	0.000711	0.451875	0.002455
17	90	90	98.517121	0.053779	0.000350	0.367790	0.002051
18	95	95	99.530702	0.048666	0.000322	0.368971	0.002077
19	100	100	100.000000	0.044911	0.000299	0.368671	0.002082

Tab 1 (2-step approach) Errors between index and portfolio returns of different portfolios

MIP Approach

Instead of using a two-step process, we can formulate the entire portfolio optimization problem as a single mixed integer programming (MIP) model. This integrated approach includes both the stock selection and weight determination decisions in one optimization model. To control the number of stocks selected, we introduce binary variables 'y_i' indicating whether stock i is chosen, along with big M constraints that force the weight w_i to zero when y_i is zero. By combining the separate stock selection and weighting models into a single MIP with integer constraints, we can directly optimize the portfolio weights while restricting the number of non-zero holdings to a fixed number m. Though more difficult to solve, this consolidated MIP approach provides an alternative methodology for the portfolio optimization problem.

Here we repeat the same steps that we have done for the two step approach but we formulate it as one problem by adding the Big M constraint. It is added as follows:

```
    # Setting the bigM constraints
    for i in stock_names:
        MIP_model.addConstr(weights[i] <= y[i]*bigM)
```

The MIP problem is implemented using the below code chunk:

```
def portfolioWeights_MIP(m, stock_names, stocks_indices_old, stock_indices_new, \
                         stock_returns_old, stock_returns_new, q, q_new, timeLimit = 300,
save=False): # Function to get the model weight (MIP)
    # Minute sanity checks
    assert len(q.columns) == 1
    assert len(q_new.columns) == 1

    # Converting actual NDX returns provided to a list
    q = list(q[list(q.columns)[0]].values)
    q_new = list(q_new[list(q_new.columns)[0]].values)

    # Intializing the model and setting the model parameters
    MIP_model = gp.Model()
    MIP_model.ModelSense = gp.GRB.MINIMIZE
    MIP_model.Params.OutputFlag = 0
    MIP_model.Params.TIME_LIMIT = timeLimit
    bigM = 1

    # Decision Variables
    weights = MIP_model.addVars(stock_names) # variable for weight of selected stock
    y = MIP_model.addVars(stock_names,vtype='B') # variable to limit the weights or push them
to zero - BigM constraint
    u = MIP_model.addVars(stocks_indices_old) # dummy variable to solve non-linearities

    # Constraint for sum of weight = 1
    sum_weights = 0
    for i in stock_names:
        sum_weights += weights[i]
    MIP_model.addConstr(sum_weights == 1)

    # Constraints to minimize non linearity like min -> |X-1|
    for j in stocks_indices_old:
        sum_return = 0
        for i in stock_names:
            sum_return += weights[i]*stock_returns_old[i][j]
        MIP_model.addConstr(u[j] >=  (q[j-1] - sum_return))
        MIP_model.addConstr(u[j] >=  -(q[j-1] - sum_return))

    # Setting the bigM constraints
    for i in stock_names:
        MIP_model.addConstr(weights[i] <= y[i]*bigM)

    # Constraint to select the number of constraints specified by the user
    sum_y = 0
    for i in stock_names:
        sum_y += y[i]
    MIP_model.addConstr(sum_y == m)

    # Setting the objective to minimize the absolute errors
    sum_obj = 0
    for j in stocks_indices_old:
```

```
        sum_obj += u[j]
    MIP_model.setObjective(sum_obj)

    MIP_model.optimize()

    # Calculating returns from model for the old data
    returns_old = []
    for j in stocks_indices_old:
        sum_return = 0
        for i in stock_names:
            sum_return += weights[i].X * stock_returns_old[i][j]
        returns_old.append(sum_return)

    # Calculating returns from model for the new data
    returns_new = []
    for j in stock_indices_new:
        sum_return = 0
        for i in stock_names:
            sum_return += weights[i].X * stock_returns_new[i][j]
        returns_new.append(sum_return)

    # Saving the calculated returns as per user's demand
    if save:
        returns_old_save = []
        for j in stocks_indices_old:
            sum_return = 0
            for i in selectedStocks:
                sum_return += weights[i].X * stock_returns_old[i][j]
            returns_old_save.append([j, q[j-1], sum_return])
        returns_old_save = pd.DataFrame(returns_old_save, columns=['Index', 'Index Return',
'Portfolio Return'])
        returns_old_save.to_csv(f'Returns_old_m-{m}_T-{timeLimit}.csv', index=False)

        returns_new_save = []
        for j in stock_indices_new:
            sum_return = 0
            for i in selectedStocks:
                sum_return += weights[i].X * stock_returns_new[i][j]
            returns_new_save.append([j, q_new[j-1], sum_return])
        returns_new_save = pd.DataFrame(returns_new_save, columns=['Index', 'Index Return',
'Portfolio Return'])
        returns_new_save.to_csv(f'Returns_new_m-{m}_T-{timeLimit}.csv', index=False)

    # Saving the weights in a dataframe
    stockWeights = pd.DataFrame(index=stock_names)
    stockWeights[f'm_{m}'] = np.nan
    for i in stock_names:
        stockWeights[f'm_{m}'].loc[i] = weights[i].X
    stockWeights = stockWeights[stockWeights[f'm_{m}'].notna()]

    # Evaluating the model on the old and the new data
    model_error = MIP_model.objVal
    new_return_error = sum(abs(np.array(q_new) - np.array(returns_new)))
    old_return_error_rmse = np.sqrt(np.mean((np.array(q) - np.array(returns_old))**2))
    new_return_error_rmse = np.sqrt(np.mean((np.array(q_new) - np.array(returns_new))**2))

    return MIP_model, stockWeights, model_error, old_return_error_rmse, new_return_error,
new_return_error_rmse
```

To evaluate solution quality tradeoffs against runtime, the MIP is solved with two different time limits - a short limit of 20 seconds and a longer limit of 1 hour and the model is run repeatedly with varying values of m (number of stocks) from 5 to 100.

The MIP with 20 secs approach is implemented as follows:

```
# Running the MIP approach with a time limit of 20 secs for a wide array of selected stocks
and saving the results
tries = []
i = 5
while i <= len(stocks2019_names):
    tries.append(i)
    i += 5

# tries = [90, 95]
data = []
for try_ in (pbar:= tqdm(tries)):
    pbar.set_description(f"Calculating for m = {try_}")
    row = []
    row.append(try_)

    weight_model, weight, model_error, old_return_rmse, new_return_error, new_return_rmse =
portfolioWeights_MIP(m=try_, \ stock_names=stocks2019_names,  \
                                          stocks_indices_old=stocks2019_index, \
                                           stock_indices_new=stocks2020_index, \
                                    stock_returns_old=stocks2019_returns_dict, \
                                    stock_returns_new=stocks2020_returns_dict, \
                                                                  q=q2019, \
                                                              q_new=q2020, \
                                                               timeLimit=20,
                                                                 save=False)

    weight = weight.loc[weight[f"m_{try_}"] > 0]
    row.append(len(weight))
    row.append(model_error)
    row.append(old_return_rmse)
    row.append(new_return_error)
    row.append(new_return_rmse)

    data.append(row)

cols = ['m', '#StocksSelected', '2019_return_error', '2019_return_RMSE', '2020_return_error',
'2020_return_RMSE']
data = pd.DataFrame(data, columns=cols)
data.to_csv("MIPApproach_T-20.csv", index=False)
data
```

We calculate the errors and the RMSE of the MPI approach and we get the following errors after running it for 20 seconds per model for each portfolio size:

	m	#StocksSelected	2019_return_error	2019_return_RMSE	2020_return_error	2020_return_RMSE
0	5	5	0.499259	0.002859	0.777362	0.004181
1	10	10	0.304668	0.001663	0.640518	0.003533
2	15	15	0.234798	0.001322	0.698479	0.003699
3	20	21	0.209536	0.001216	0.608830	0.003294
4	25	25	0.235961	0.001332	0.655472	0.003458
5	30	29	0.182629	0.000994	0.485423	0.002605
6	35	35	0.172666	0.000975	0.476920	0.002570
7	40	39	0.135122	0.000782	0.411286	0.002287
8	45	45	0.103895	0.000612	0.378738	0.002080
9	50	50	0.072081	0.000445	0.379610	0.002071
10	55	54	0.072243	0.000442	0.404361	0.002263
11	60	59	0.062469	0.000398	0.387233	0.002117
12	65	65	0.052271	0.000316	0.367752	0.002064
13	70	70	0.048280	0.000309	0.363705	0.002022
14	75	75	0.046548	0.000300	0.369853	0.002086
15	80	80	0.045227	0.000296	0.370629	0.002095
16	85	85	0.044966	0.000301	0.361019	0.002044
17	90	88	0.044911	0.000299	0.368671	0.002082
18	95	88	0.044911	0.000299	0.368671	0.002082
19	100	88	0.044911	0.000299	0.368671	0.002082

Tab 2 (MIP Approach – 20 sec) Errors between index and portfolio returns of different portfolios

The MIP with 1 hour approach is implemented as follows:

```
# Running the MIP approach with a time limit of 20 secs for a wide array of selected stocks
and saving the results
tries = []
i = 5
while i <= len(stocks2019_names):
    tries.append(i)
    i += 5

# tries = [90, 95]
data = []
```

```
for try_ in (pbar:= tqdm(tries)):
    pbar.set_description(f"Calculating for m = {try_}")
    row = []
    row.append(try_)

    weight_model, weight, model_error, old_return_rmse, new_return_error, new_return_rmse =
portfolioWeights_MIP(m=try_, \ stock_names=stocks2019_names,  \
                                          stocks_indices_old=stocks2019_index, \
                                           stock_indices_new=stocks2020_index, \
                                    stock_returns_old=stocks2019_returns_dict, \
                                    stock_returns_new=stocks2020_returns_dict, \
                                                                  q=q2019, \
                                                               q_new=q2020, \
                                                             timeLimit=3600,
                                                                 save=False)

    weight = weight.loc[weight[f"m_{try_}"] > 0]
    row.append(len(weight))
    row.append(model_error)
    row.append(old_return_rmse)
    row.append(new_return_error)
    row.append(new_return_rmse)

    data.append(row)

cols = ['m', '#StocksSelected', '2019_return_error', '2019_return_RMSE', '2020_return_error',
'2020_return_RMSE']
data = pd.DataFrame(data, columns=cols)
data.to_csv("MIPApproach_T-20.csv", index=False)
data
```

We calculate the errors and the RMSE of the MPI approach and we get the following errors after running it for an hour per model for each portfolio size:

	m	#StocksSelected	2019_return_error	2019_return_RMSE	2020_return_error	2020_return_RMSE
0	5	5	0.499259	0.002859	0.777362	0.004181
1	10	10	0.290137	0.001564	0.753372	0.004347
2	15	15	0.207520	0.001198	0.636754	0.003357
3	20	20	0.157501	0.000914	0.559464	0.002906
4	25	25	0.133442	0.000776	0.478015	0.002620
5	30	30	0.105653	0.000616	0.491623	0.002564
6	35	35	0.092986	0.000565	0.476436	0.002609
7	40	40	0.078577	0.000455	0.408250	0.002210
8	45	45	0.067697	0.000436	0.406685	0.002198
9	50	50	0.060979	0.000379	0.391677	0.002157
10	55	55	0.056206	0.000389	0.376091	0.002084
11	60	60	0.052442	0.000349	0.373526	0.002082
12	65	65	0.049173	0.000315	0.369197	0.002059
13	70	70	0.047488	0.000312	0.360502	0.002015
14	75	75	0.046042	0.000304	0.362995	0.002046
15	80	80	0.045227	0.000296	0.370629	0.002095
16	85	85	0.044966	0.000301	0.361019	0.002044
17	90	88	0.044911	0.000299	0.368671	0.002082
18	95	88	0.044911	0.000299	0.368671	0.002082
19	100	88	0.044911	0.000299	0.368671	0.002082

Tab 3 (MIP Approach – 1 hour) Errors between index and portfolio returns of different portfolios

We ran the model for both 20 seconds and an hour as run time to see if there were any differences in the errors as estimated by a better optimal answer after running it for an hour as opposed to running it for 20 seconds and we found that the differences were not that huge. This means that irrespective of the runtime, gurobi was able to find only slightly better optimal solutions in an hour compared to 20 seconds.

Below is the plot that we have obtained for error comparison of MIP with 20s and 3600s

Fig 2 (MIP Approach) Errors between models with different runtime

MIP Error comparison inferences and recommendations:

The error comparison between the 20 second and 1 hour (3600 seconds) time limits indicates that for lower values of m (fewer stocks in the portfolio), the 20 second solutions exhibit more variability and higher errors compared to the 1 hour solutions. However, as m increases, the errors for the 20 second case stabilize and converge towards the 1 hour errors.
This suggests that for concentrated portfolios with small m, limiting the MIP to only 20 seconds prevents it from finding high quality solutions. More runtime is needed to optimize the weights properly when there are fewer stocks. But for very diversified portfolios with large m, even 20 seconds provides sufficient time to find near optimal solutions.

From a business perspective, the implications are:

- For **focused portfolios with limited holdings**, it is worth spending more time to optimize, as the **1 hour solutions** lead to significantly better risk-adjusted returns.
- However, for **highly diversified portfolios**, a **short 20 second** optimization may be adequate, as longer runtimes provide diminishing improvements.

This indicates that businesses should calibrate the time spent on optimization based on the intended portfolio concentration. Concentrated portfolios warrant more compute time to yield better solutions, while diversified portfolios can rely on quick approximations without significant degradation in quality. Properly matching optimization time to portfolio breadth allows efficiently balancing solution quality with compute resources.

Results

Both methods were employed to analyze the stock price data. We utilized the provided stock data to compute the returns for the optimized equity portfolio in the years 2019 and 2020. Subsequently, a comparison was made between the portfolio returns and the broader market index returns for these two years, aiming to assess the variance or difference between the portfolio and the index.

Two-Step approach:
This quantitative analysis revealed error values of ***0.789178*** *for 2019 and* ***1.1124*** *for 2020* while selecting 5 stocks for the portfolio.

MIP approach:
This quantitative analysis where the model run for 1 hour, revealed error values of ***0.044910*** *for 2019 and* ***0.368670*** *for 2020* while selecting 5 stocks for the portfolio.

To figure out how the portfolio is following the index, we have plotted the return of both the portfolio and the index on a daily basis, and the aim is to indicate the extent to which the returns of the optimized portfolio deviated from the market benchmark for each day, based on the stock data. Here we look at the days starting from Day 0 to Day 250 and the returns on the y-axis for 2019 and 2020

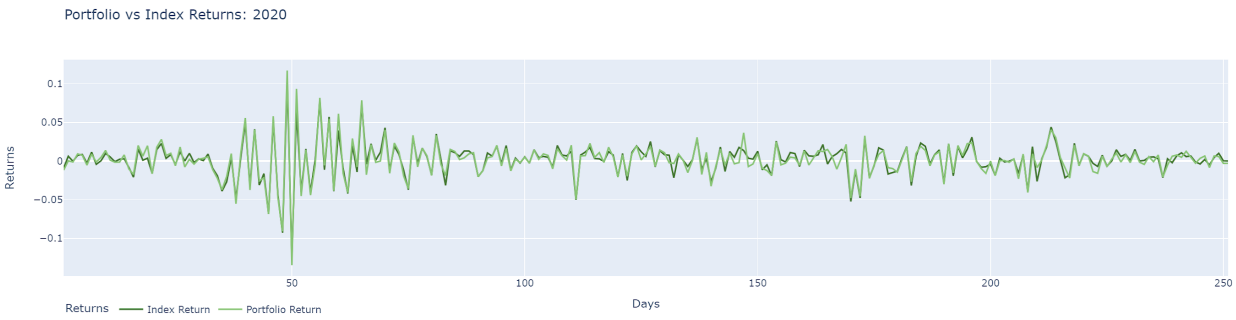

Fig 3 (2-step Approach) Errors between index and portfolio returns across different days in 2020

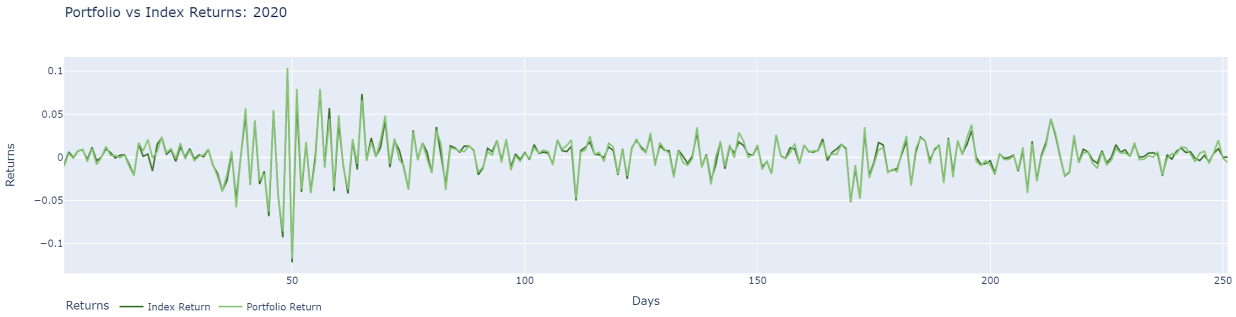

Fig 4 (MIP Approach) Errors between index and portfolio returns across different days in 2020

Given the two approaches, looking at daily returns and how approximately the portfolios follow them did not help identify the best approach, so we focused on the errors between the returns and see if we could evaluate the performance of these approaches in terms of the portfolio sizes as well.

Additional analysis was conducted to investigate how alterations in portfolio size influenced the error value. We computed the optimized portfolio return for various quantities of stocks (denoted as 'm'), including 5, 20, 40, 60, 80, and 100 stocks, for the years 2019 and 2020. The findings revealed a consistent trend: as the number of holdings increased, the error value steadily diminished, signifying that the portfolio's returns were progressively converging with those of the broader market.

A noteworthy deviation arises with the two-step approach, characterized by an abrupt surge in error values during the computation of weight assignments for assembling a portfolio comprising 60 stocks. To gain a comprehensive understanding of these observations, it is essential to delve deeper into the 2020 error assessments for both methodologies and examine how they compare side by side.

Fig 5 (2-step Approach) Errors between the years for different portfolio sizes

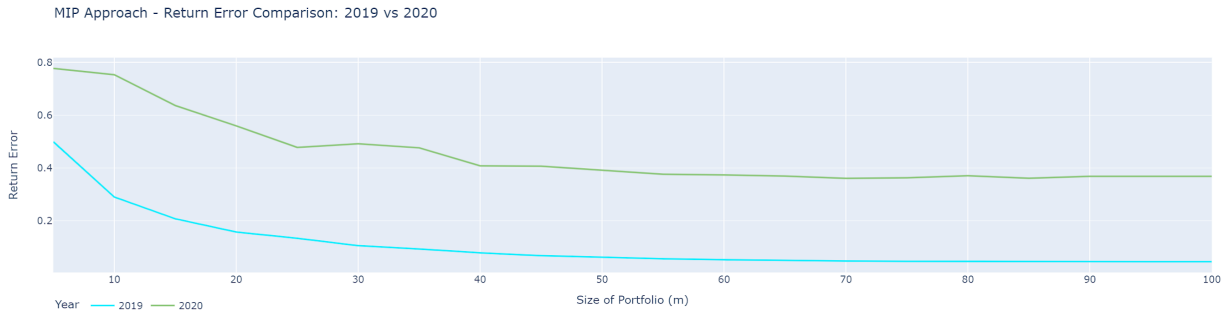

Fig 6 (MIP Approach) Errors between the years for different portfolio sizes

There was a decline in error values reached a plateau at a portfolio size of approximately 88 stocks, beyond which there was minimal further enhancement. This analysis conclusively demonstrated that a substantial portion of the diversification benefits was achieved with a portfolio consisting of around 88 stocks, with marginal gains from additional holdings becoming increasingly limited.

The 12 stocks that have not been added to the portfolio even with the possibility of adding in all 100 are: ALGN, ADI, ANSS, CDW, CERN, CPRT, DOCU, LBTYA, SGEN, SWKS, SPLK and WDC

These results provide valuable insights into the optimal size of the portfolio and its relationship to error values, guiding our approach to portfolio construction.

Comparison of Two-Step Approach and MIP Approach:

Fig 7 Error of 2-step and MIP approaches for different portfolio sizes

Considering the 5-stock portfolio, the stocks selected by the two-step approach and the MIP approach are different fundamentally due to the process being followed. The two-step approach has 2 different models where the stocks are selected first and then the second model is forced to assign weights to those 5 selected stocks whereas the MIP approach considers only one model where the weights are assigned directly which also solves the selection problem. Though the process of selection and assigning stocks is the same in both cases, the way they are done is different which causes gurobi to select different stocks and the weights will also differ based on this factor. The different stocks being selected in the m=5 portfolio for both the methods are as follows:

	m_5
MSFT	0.580352
MXIM	0.210388
XEL	0.089208
VRTX	0.071190
LBTYK	0.048862

Two-Step Approach

	m_5
MSFT	0.309409
AMZN	0.205360
AAPL	0.185634
CTXS	0.165613
TXN	0.133984

MIP Approach

In general, when evaluating the 2020 data, it becomes evident that the MIP approach exhibits lower error rates in comparison to the two-step approach. Furthermore, the MIP method demonstrates a consistent and gradual decrease in error values, while the two-step process displays a notable degree of variability in its error measurements.

Now that we have decided on going with the MIP approach, we need to decide the number of stocks to include in the portfolio and for that we use the error in returns and we bring up the MIP error chart to help us with this decision:

Fig 8 (MIP Approach) Errors between the years for different portfolio sizes

Looking at 2020 data we see that the errors are flatter towards the end and it is evident that we can go for anything near 100 but that might not be a good idea because the whole idea of the portfolio is to have a lesser number of stocks and assign weights to them. So the goal would be to have a sweet spot between the error and the number of stocks to be included in the portfolio. When we look at it that way, we want to look at the least value where the error curve starts to flatten and in this case, the size of the portfolio is 40 beyond which the curve is flat.

To further bolster the fact that m=40 follows the index better, we compare the daily returns of the portfolio with 5 stocks and 40 stocks with the index returns as follows:

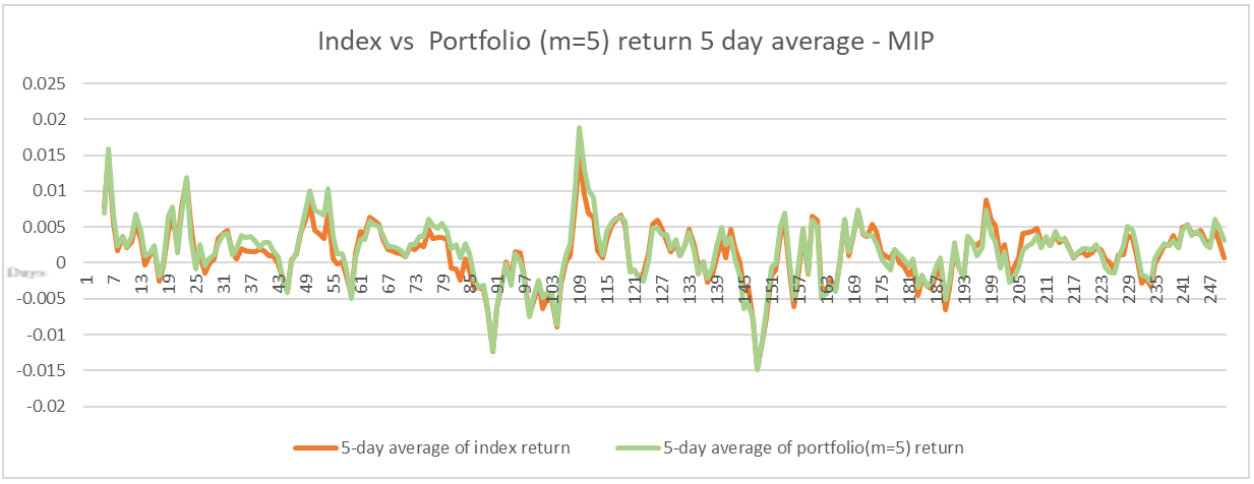

Fig 9 Avg 5 Day return of Index vs Portfolio when m=5 for MIP Approach

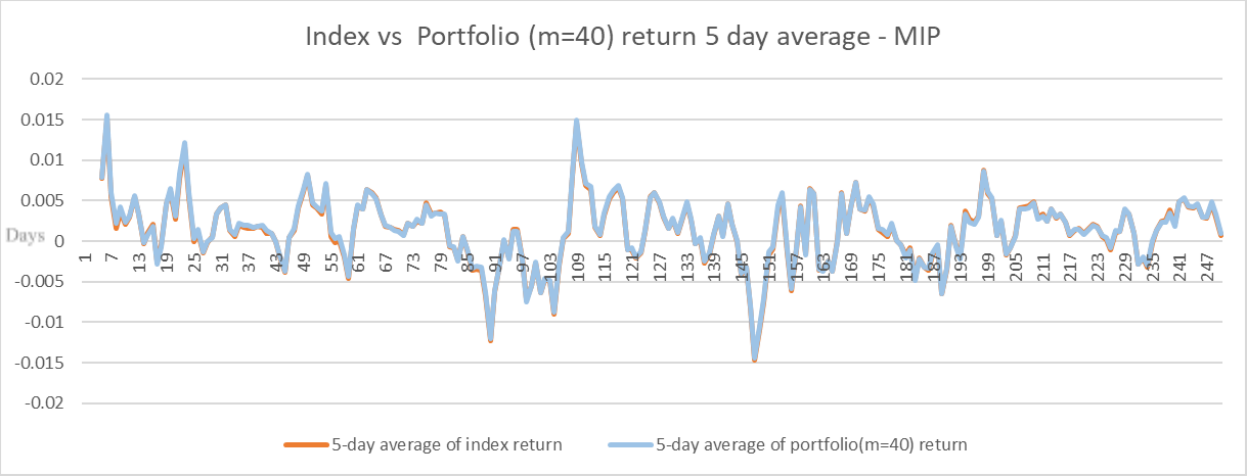

Fig 10 Avg 5 Day return of Index vs Portfolio when m=40 for MIP Approach

As we can see from the graphs, the m=40 portfolio follows the index more closely compared to a portfolio size of 5 and gives us more incentive to use this portfolio.

Recommendations

1. **Size of the Portfolio:**

   Our suggestion is to set 'm' at approximately 40, corresponding to the noticeable 'elbow' point on the chart.

   This quantity is sufficiently minimal yet effective in replicating the desired index. If the expenses associated with incorporating additional stocks are manageable, there's room to expand 'm'. The precise determination of 'm' should factor in the financial implications of enlarging the portfolio (such as rebalancing charges and market reactions to transactions) against the potential inaccuracies in emulating the NASDAQ-100.

   It's crucial to acknowledge, however, the substantial computational demand and time commitment Method 2 necessitates. We should resort to Method 1 only under constraints of resources. Our current optimization algorithm has operated for roughly 10 hours, but with access to enhanced computational capabilities and extended operation periods, we might realize further optimization in performance.

2. **Assigning Weights:**

   We advocate for the adoption of Method 2 in structuring our Index fund to reflect the NASDAQ-100 accurately.

   Upon analysis, Method 2 outshines Method 1 in performance, exhibiting reduced error rates for nearly all 'm' values. Its efficacy maximizes when 'm' equals 50, beyond which the improvements stagnate. In contrast, Method 1 shows significant inconsistency, though its performance progressively enhances with an increasing 'm'.

3. **Run-Time:**

   Considering m=40, it is a trade-off between a highly diversified portfolio and a smaller focused portfolio, therefore a time period of 20-40 minutes would be ideal.

   We saw that the focused portfolios require more time to get an better result and the diversified portfolios did not require much time to get a better result which did not change with more run-time either. Since we want go with a 40-stock portfolio, going for a time-period somewhere in the middle of the range of 0-60 minutes makes sense to get better results for a portfolio that lies in the middle of the range in terms of the portfolio size as well.