The University of Texas at Austin
McCombs School of Business

RM 294: OPTIMIZATION
PROJECT 2 REPORT

GROUP 25 MEMBERS

1. SANTHOSH RAMKUMAR (SR55965)
2. BHUVANA CHANDRIKA KOTHAPALLI (BK24542)
3. GAYTRI RIYA VASAL (GRV377)
4. JAHNAVI ANGATI (JA54632)

INDEX

PROBLEM OVERVIEW

In the modern financial landscape, investors, both institutional and individual, face a deluge of information as they seek to optimize returns while minimizing risks and initial capital investment. Equity money management strategies can be broadly categorized as either 'active' or 'passive.' Passive investing, exemplified by indexing, revolves around replicating market indices or specific sectors. However, the critical challenge is to construct tracking portfolios that closely mirror these benchmarks while adeptly navigating frequent and significant transaction costs resulting from price fluctuations and the need for periodic portfolio rebalancing. The core issue is to create a tracking portfolio composed of (n) assets, effectively capturing most target portfolio or sector movements, all while employing as few stock (m) as possible, striking a balance between precision and cost-efficiency.

PROJECT GOAL AND APPROACH

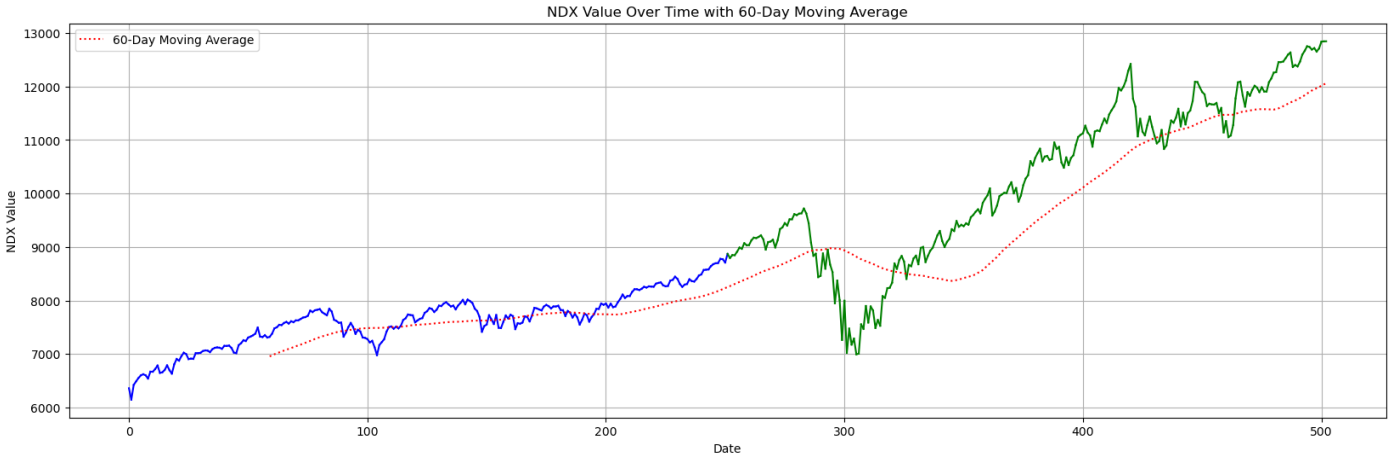
The primary goal of this project is to design and implement an Index fund that accurately tracks the NASDAQ-100 index while using a substantially smaller number of stocks (m) than the index's total components (n). This will involve the following steps:

- **Formulating an Integer Program:** Create an integer program that selects precisely m out of n stocks for the portfolio. The program will take a 'similarity matrix' (ρ) as input, where ρij represents the similarity between stock i and j, using metrics like correlation or other similarity measures.
- **Solving a Linear Program:** Determine the quantity of each selected stock to purchase for the portfolio using a linear program.
- **Performance Evaluation:** Assess the performance of the newly created index fund against the NASDAQ-100 index using out-of-sample data. This evaluation will be conducted for various values of m, allowing for a comprehensive understanding of the fund's performance under different levels of diversification.

In essence, the project aims to strike a balance between precision, cost-effectiveness, and diversification in passive equity management by optimizing the construction of an index fund that efficiently tracks the NASDAQ-100 index while minimizing the number of constituent stocks.

WHY SHOULD WE CONSIDER INVESTING?

To answer this, we have plotted a graph to visualize how the NDX index values changed daily over the specified time period, with a focus on capturing trends and patterns using the moving average line. This moving average provides a smoothed representation of the NDX index's trend over time. The **blue line** represents 2019 data and the **green line** represents 2020 data.



From the above graph, it is evident that there is a growing trend of the index from 2019 to 2020 which suggests a compelling opportunity for investors. This upward trajectory implies the potential for capital appreciation and positive returns on investments in the NASDAQ-100 index. Additionally, it indicates a favorable market environment and investor confidence in the listed companies and hence there's value in investing!

Note: Code is provided in appendix (1).

1.DATA OVERVIEW

The dataset consists of two CSV files containing daily prices for the NASDAQ-100 index and its component stocks in 2019 and 2020. These files are structured with date as the first column, followed by the index price (NDX), and columns 3 to 102 containing the prices of individual component stocks. For the analysis, we will utilize the 2019 data for portfolio construction and then assess the performance of the constructed portfolios using the 2020 data. It's important to note that this project is designed to be adaptable to different datasets, allowing for the evaluation of the methodology's effectiveness with various indexes and components. This section sets the foundation for the subsequent analysis of stock selection, weight construction, and portfolio performance.

2. MAPPING STOCK SYNERGY

We begin by calculating the correlation matrix of returns for a simple reason - to understand the relationships between different stocks as if we're connecting the dots in a complex puzzle. This critical step allows us, as analysts, to make informed decisions in building a portfolio that mimics the behavior of the NASDAQ-100 index.

Code

```
#Loading the Data
stocks_2019 = pd.read_csv(r'stocks2019.csv')
stocks_2020 = pd.read_csv(r'stocks2020.csv')
```

```
returns_2019 = stocks_2019.copy()  # Create a copy of the original DataFrame
returns_2019.drop(columns=['X'], inplace=True)  # Drop the 'Date' column

# Calculate daily returns
returns_2019 = returns_2019.pct_change().fillna(0)
returns_2019.head(10)
```

```
#Now, lets calculate the correlation matrix --> This will give us the correlation of the 'RETURNS'
#from each stock with repect to each other

correl_2019 = returns_2019.drop(columns=['NDX']).corr()
correl_2019.head()
```

Output

Below is the correlation matrix representing the relationships among the initial set of stocks.

	atvi	ADBE	AMD	ALXN	ALGN	GOOGL	GOOG	AMZN	AMGN	ADI	ANSS	AAPL
atvi	1.000000	0.399950	0.365389	0.223166	0.216289	0.433106	0.426786	0.467082	0.203969	0.329366	0.384474	0.430635
ADBE	0.399950	1.000000	0.452876	0.368931	0.363384	0.552137	0.540417	0.598244	0.292003	0.473834	0.734349	0.506160
AMD	0.365389	0.452876	1.000000	0.301835	0.344268	0.418879	0.417272	0.549310	0.151486	0.503752	0.491757	0.495625
ALXN	0.223166	0.368931	0.301835	1.000000	0.332437	0.315997	0.307702	0.363174	0.342026	0.317044	0.407130	0.313158
ALGN	0.216289	0.363384	0.344268	0.332437	1.000000	0.248759	0.250329	0.399289	0.264613	0.328292	0.371002	0.357672
GOOGL	0.433106	0.552137	0.418879	0.315997	0.248759	1.000000	0.997363	0.619086	0.258290	0.404610	0.482417	0.557860
GOOG	0.426786	0.540417	0.417272	0.307702	0.250329	0.997363	1.000000	0.609681	0.254718	0.400878	0.476795	0.558475
AMZN	0.467082	0.598244	0.549310	0.363174	0.399289	0.619086	0.609681	1.000000	0.240476	0.476477	0.576238	0.593011
AMGN	0.203969	0.292003	0.151486	0.342026	0.264613	0.258290	0.254718	0.240476	1.000000	0.266780	0.271639	0.279211
ADI	0.329366	0.473834	0.503752	0.317044	0.328292	0.404610	0.400878	0.476477	0.266780	1.000000	0.539976	0.547007
ANSS	0.384474	0.734349	0.491757	0.407130	0.371002	0.482417	0.476795	0.576238	0.271639	0.539976	1.000000	0.552767
AAPL	0.430635	0.506160	0.495625	0.313158	0.357672	0.557860	0.558475	0.593011	0.279211	0.547007	0.552767	1.000000

3. INITIAL PORTFOLIO CONSTRUCTION

Approach

In the stock selection and weight construction phase, our primary objective is to design an index fund that effectively tracks the NASDAQ-100 index with a reduced number of component stocks. We employed a two-step optimization process, combining Integer Programming (IP) for stock selection and Linear Programming (LP) for weight allocation. In the IP step, we sought to identify the most suitable subset of stocks from the NASDAQ-100 index that would form our portfolio. The criteria for stock selection were based on the 'similarity matrix,' denoted as ρ, which represents the relationships between individual stocks. By maximizing the similarity between chosen stocks and their representatives, we aimed to create a well-diversified portfolio. In the LP phase, we aimed to determine the optimal weights for the selected stocks, considering their historical returns in relation to the NASDAQ-100 index. We formulated this problem as a linear program, making use of a transformation technique to address the non-linearity introduced by the absolute value function. This approach allowed us to calculate the portfolio weights that would closely mirror the index's returns. The outcome of this phase provided us with insights into how well the constructed portfolio tracked the index in both in-sample (2019) and out-of-sample (2020) scenarios.

Stock Selection

In our stock selection process, our foremost objective is to maximize the similarity between the chosen m stocks and the entire index, comprising n stocks. This is expressed by the following equation:

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

Here, pi,j represents the correlation between the stocks in the created fund and the index, and xi,j is a binary constraint indicating whether a stock from the index is included in the fund.

To ensure the fulfillment of this objective, we have incorporated three fundamental constraints:

Constraint 1: This constraint implies that no more than q (m in the original context) elements from set I (the index) can be mapped to set F (the fund). This ensures that only a limited number of stock from the index can be included in the fund.

Constraint 2: This constraint asserts that each element in set I maps to a single element in set F. It enforces a one-to-one relationship between the stocks in the index and those in the fund, promoting the idea that each index stock corresponds to one fund stock.

Constraint 3: This constraint suggests that if an element from set I is mapped to an element in set F (i.e., a stock from the index is included in the fund), then the selected element in set F must be present in the fund. This constraint ensures that if an index stock is chosen, it must be part of the fund.

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

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

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

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

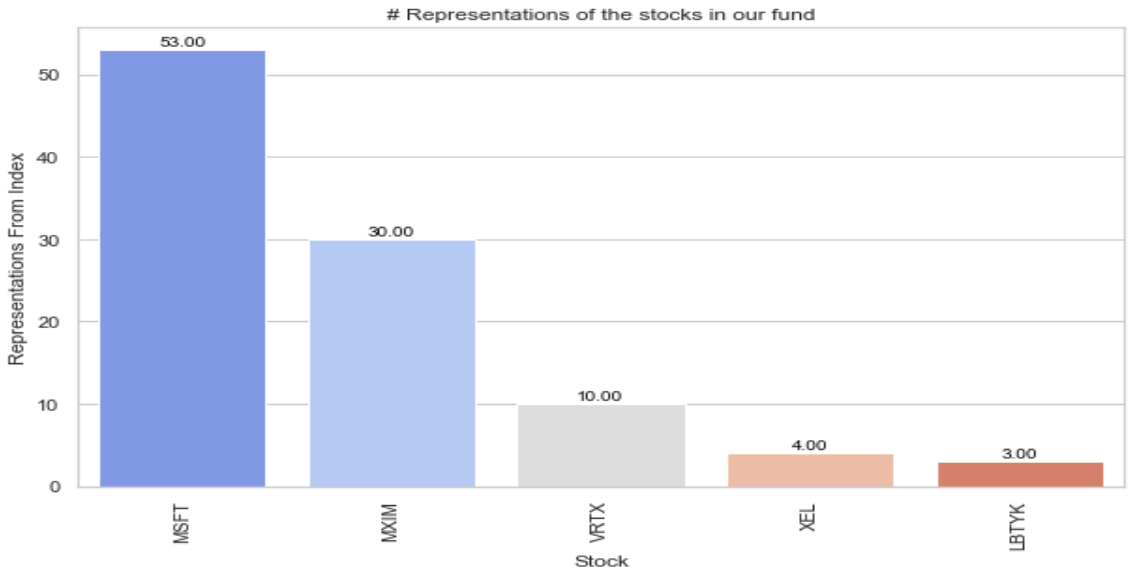
Now, let's construct an index fund with 5 stocks.

Stock Selection Code (Index fund with 5 stocks)

```
m = 5

fund_creator = gp.Model()
selected_stocks = fund_creator.addMVar(100, vtype = 'B') #Binary variable to flag the selected stocks
linking_stocks = fund_creator.addMVar((100, 100), vtype = 'B') #Binary variable to link stock in index to the fund

selection_const = fund_creator.addConstr(gp.quicksum(selected_stocks[i] for i in range(n)) <= m)
one_representative_const = fund_creator.addConstrs(gp.quicksum(linking_stocks[i,j] for j in range(n)) == 1 for i in range(n))
ensure_best_representative_const = fund_creator.addConstrs(linking_stocks[i,j] <= selected_stocks[j] for j in range(n) for i in range(n))

# Set the objective function using the calculated values
fund_creator.setObjective(gp.quicksum(correl_2019.iloc[i, j] * linking_stocks[i,j] for i in range(n) for j in range(n)), sense = gp.GRB.MAXIMIZE)
fund_creator.Params.OutputFlag = 0
fund_creator.optimize()

# Convert the array to a pandas Series for easier manipulation
selection_series = pd.Series(selected_stocks.x)

# Get the selected columns from returns_2019
selected_columns = correl_2019.columns[selection_series == 1]
selected_columns_list = selected_columns.tolist()
# Print the names of the selected columns
selected_columns_list
```

Optimization Output and analysis

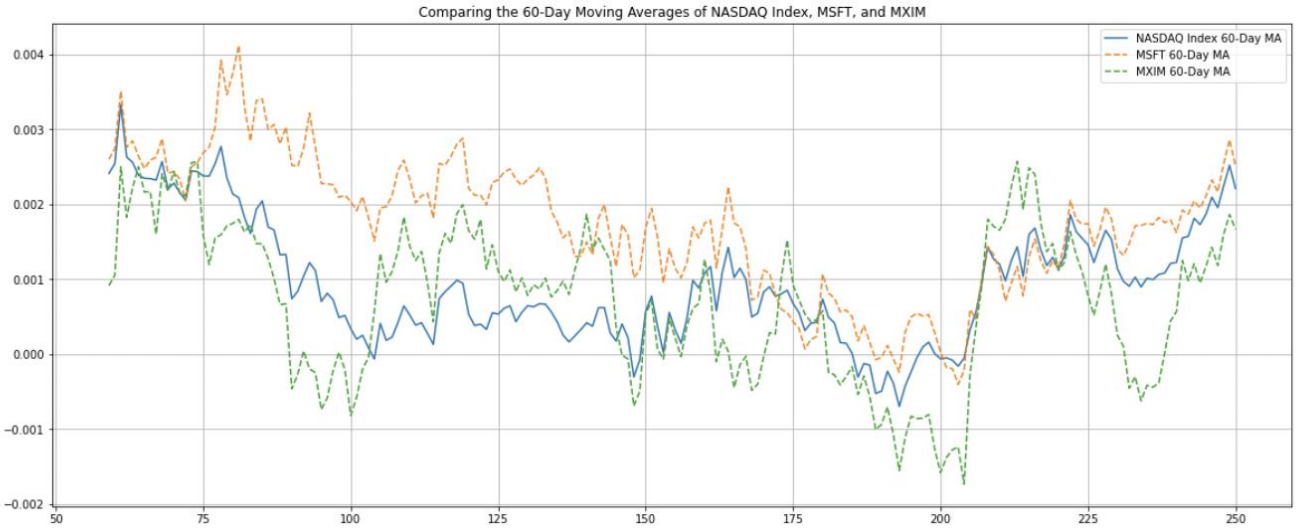
LBTYK, MXIM, MSFT, VRTX, and XEL are the five best stocks to mimic the index. As the stocks have been identified based on their correlation with 100 stocks in the NASDAQ index, let's see how representative these fund stocks are of the actual index.



From the chart above, we see that **MSFT (Microsoft)** is the most representative stock out of the top 5 selected. Basically, **53** stocks in the index are mapped to MSFT based on optimization to maximize correlation.

In order to perform a more comprehensive analysis, one ought to examine how this stock's return trends with index returns.

For this, we created a plot that compares the 60-day moving averages of the NASDAQ Index, MSFT stock, and MXIM stock for the year 2019.



- MSFT demonstrates a reasonable correlation with the index, following its general movements. However, it is notable that the magnitude of its returns differs noticeably, as evident from the chart.
- As for MXIM, its correlation with the NASDAQ index is intermittent. Similar to MSFT, the scale of its returns significantly deviates from that of the index.
- While the three other stocks in the fund may not be as strong representatives, they do exhibit a degree of correlation with the index.
- These observations instill confidence in our stock selection approach, which relies on correlation as a key criterion. Now, let's move on to applying appropriate weights to these stocks in the funds such that it closely follows the index.

Portfolio Weights Calculation

After we've determined the stocks that will make up our fund portfolio, the next step involves calculating the ideal allocation for each stock. This allocation aims to closely match the weighted average returns of our portfolio with the returns of the index.

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

To ensure the integrity of this allocation, two critical constraints must be met. Firstly, the sum of all weight allocations should equal 1, guaranteeing that the entire portfolio allocation is effectively utilized. Secondly, each individual weight should be within the range of 0 to 1, signifying that the allocation for each stock remains non-negative and does not exceed 100%.

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

$$w_i \geq 0.$$

Approach

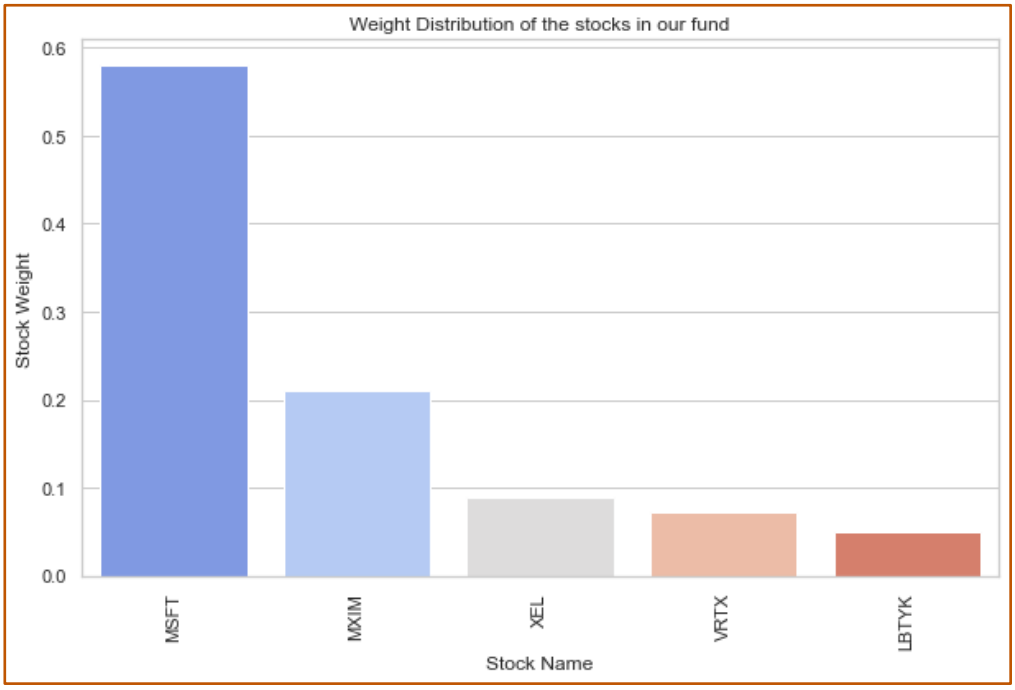
The goal here is to determine the optimal allocation of each chosen stock within the portfolio. This optimization process aims to minimize the disparity between the NASDAQ index and the portfolio's returns, which are weighted based on the selected stocks. In essence, these weights indicate the proportion of each stock's inclusion in the fund, ensuring that the portfolio closely mirrors the performance of the NASDAQ index.

```
portfolio = gp.Model()
portfolio_weights = portfolio.addMVar(5)
index_diff = portfolio.addMVar(days_2019)
portfolio_weights_const = portfolio.addConstr(gp.quicksum(portfolio_weights[i] for i in range(m)) == 1)
mod_const_1 = portfolio.addConstrs(
    (gp.quicksum(portfolio_weights[i] * selected_stocks.iloc[j, i] for i in range(m)) - index_2019[j]) <= index_diff[j]
    for j in range(days_2019)
)
mod_const_2 = portfolio.addConstrs(
    (-1 * gp.quicksum(portfolio_weights[i] * selected_stocks.iloc[j, i] for i in range(m)) + index_2019[j]) <= index_diff[j]
    for j in range(days_2019)
)
portfolio.setObjective(gp.quicksum(index_diff[i] for i in range(days_2019)), sense = gp.GRB.MINIMIZE)
portfolio.Params.OutputFlag = 0
portfolio.optimize()
```

```
print('\n The difference between 2019 NASDAQ Index return and our fund return = ', round(portfolio.objval, 2), '\n \n ', 'The weights assigned to stocks in our funds are -')

portfolio_weights_df = pd.DataFrame()
portfolio_weights_df['stock'] = selected_columns_list
portfolio_weights_df['weights'] = portfolio_weights.x.tolist()

portfolio_weights_df = portfolio_weights_df.sort_values('weights', ascending = False)
portfolio_weights_df
```

Output and Analysis

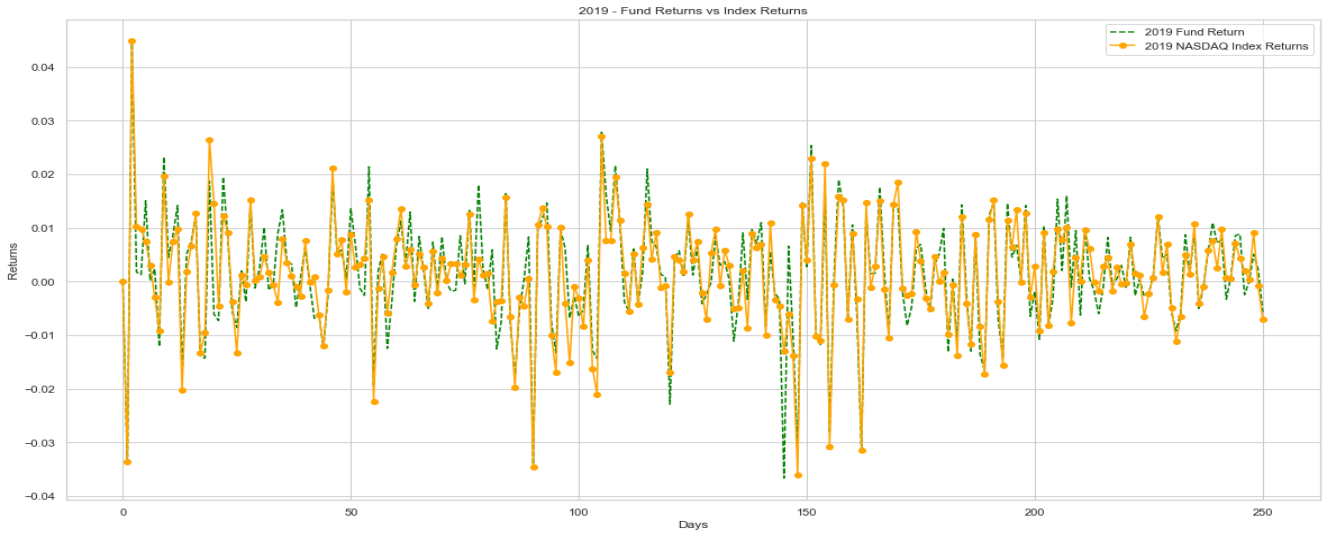
	stock	weights
2	MSFT	0.580352
1	MXIM	0.210388
4	XEL	0.089208
3	VRTX	0.071190
0	LBTYK	0.048862



From the above plot, it is evident that the weights allocated to the stocks in our portfolio correspond to their effectiveness in representing the index. MSFT and MXIM, being the most representative stocks, have received the highest weightings.

Let's compare the performance of the fund to the NASDAQ index, both in the training (2019) and testing (2020) periods. This helps in understanding how well the portfolio construction method performed in mirroring the index returns in different market conditions.

Plotting fund return v.s. NASDAQ index return for 2019



Note: Check Appendix for Code (2)

Plotting fund return v.s. NASDAQ index return for 2020

Note: Check Appendix for Code (3)

The difference between the 2019 NASDAQ Index and the fund we have created (Training error) = **0.79.**

The difference between the 2020 NASDAQ Index and the fund we have created (Testing error) = **1.11.**

These errors suggest that the portfolio construction method was effective in replicating the index's returns during the training period. However, in the testing period (2020), there was a slightly higher deviation, suggesting that the fund's ability to replicate the index's returns may be influenced by changing market conditions. It is important to further analyze the factors contributing to this variance and consider potential adjustments to improve fund performance in different market environments.

3. PORTFOLIO ANALYSIS FOR DIFFERENT ‘M’ VALUES

Approach

We have observed that constructing a fund with only 5 stocks yields satisfactory results, but now, we seek to explore the potential for further improvement. To do so, we will revisit the stock selection process by varying the number of stocks included in the fund, denoted as 'm.' We offer users the flexibility to customize the range of 'm' values to be evaluated, with intervals of 10 between each value. This analysis covers a broad spectrum of m, ranging from 10 to 100. By systematically adjusting 'm,' we aim to pinpoint the optimal balance between portfolio size and tracking performance. This allows us to determine whether there are diminishing returns in terms of tracking accuracy as more stocks are included in the portfolio. The results obtained for each 'm' value will provide valuable insights into the trade-offs between portfolio complexity and tracking effectiveness.

Code

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

```
selected_stock_dictionary = {}
weights_dictionary = {}
objective_selected_stock = []
for m in m_array :

    if n >= m :

        fund_creator = gp . Model ()
        selected_stocks = fund_creator . addMVar ( 100 , vtype = 'B' ) #Binary variable to flag the selected stocks
        linking_stocks = fund_creator . addMVar (( 100 , 100 ), vtype = 'B' ) #Binary variable to link stock in index to the fund

        selection_const = fund_creator . addConstr ( gp . quicksum ( selected_stocks [ i ] for i in range ( n )) <= m )
        one_representative_const = fund_creator . addConstrs ( gp . quicksum ( linking_stocks [ i , j ] for j in range ( n )) == 1 for i in range ( n ))
        ensure_best_representative_const = fund_creator . addConstrs ( linking_stocks [ i , j ] <= selected_stocks [ j ] for j in range ( n ) for i in range ( n ))

        # Set the objective function using the calculated values
        fund_creator . setObjective ( gp . quicksum ( correl_2019 . iloc [ i , j ] * linking_stocks [ i , j ] for i in range ( n ) for j in range ( n ) ), sense = gp . GRB . MAXIMIZE )
        fund_creator . Params . OutputFlag = 0
        fund_creator . optimize ()

        objective_selected_stock . append ( fund_creator.objval ) _ _

        # Convert the array to a pandas Series for easier manipulation
        selection_series = pd . Series ( selected_stocks . x )

        # Get the selected columns from returns_2019
        selected_columns = correl_2019 . columns [ selection_series == 1 ]
        selected_columns_list = selected_columns . tolist ()
        print ( ' \n M = ' , m , ' | Selected Stocks = ' , selected_columns_list , ' \n ' )
        selected_stock_dictionary [ m ] = selected_columns_list

        selected_stocks = returns_2019 [ selected_columns_list ]
        index_2019 = returns_2019 [ 'NDX' ]

        portfolio = gp . Model ()
        portfolio_weights = portfolio . addMVar ( m )
        index_diff = portfolio . addMVar ( days_2019 )
        portfolio_weights_const = portfolio . addConstr ( gp . quicksum ( portfolio_weights [ i ] for i in range ( m )) == 1 )
```

```
        mod_const_1 = portfolio . addConstrs (
            ( gp . quicksum ( portfolio_weights [ i ] * selected_stocks . iloc [ j , i ] for i in range ( m )) - index_2019 [ j ]) <= index_diff [ j ]
            for j in range ( days_2019 )
        )
        mod_const_2 = portfolio . addConstrs (
            ( - 1 * gp . quicksum ( portfolio_weights [ i ] * selected_stocks . iloc [ j , i ] for i in range ( m )) + index_2019 [ j ]) <= index_diff [ j ]
            for j in range ( days_2019 )
        )
        portfolio . setObjective ( gp . quicksum ( index_diff [ i ] for i in range ( days_2019 )), sense = gp . GRB . MINIMIZE )
        portfolio . Params . OutputFlag = 0
        portfolio . optimize ()
        print ( ' \n Portfolio Weights = ' , portfolio_weights . x , ' \n ' )
        weights_dictionary [ m ] = portfolio_weights . x

    else :
        print ( 'Error! Trying to select more stocks than present in the raw data!' )
```

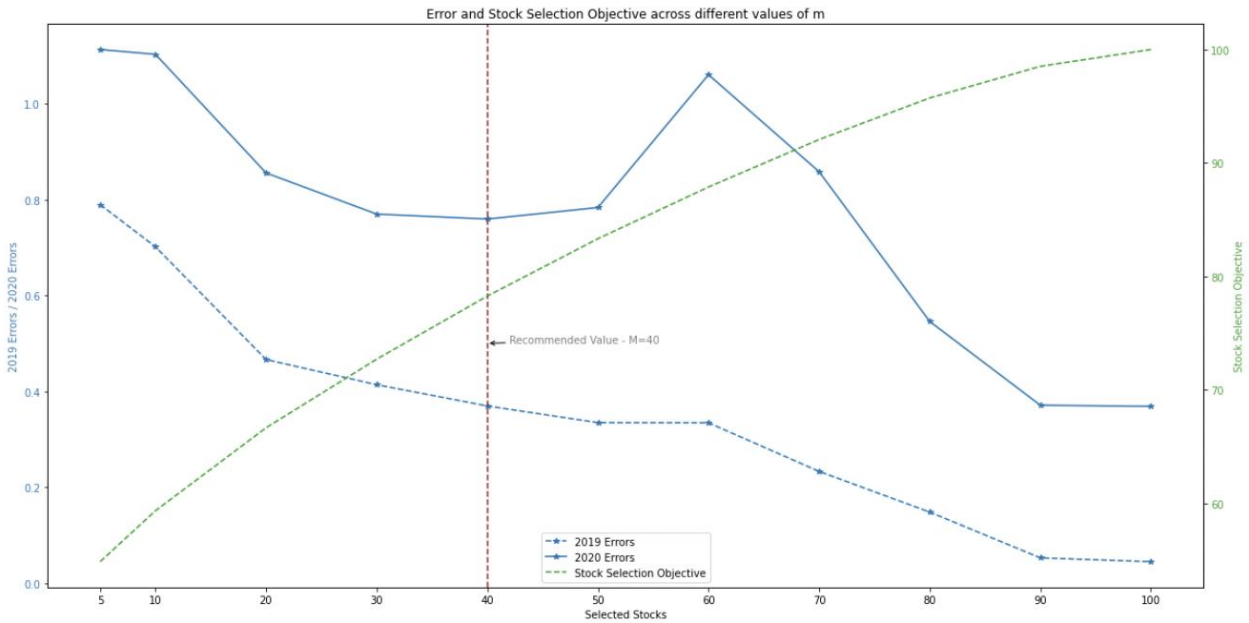
.

Output

Selected Stocks	Stock Selection Objective	2019 Errors (Basic Model)	2020 Errors (Basic Model)
5	54.841179	0.789178	1.112437
10	59.333081	0.701218	1.102404
20	66.648971	0.466233	0.855213
30	72.697515	0.413754	0.769412
40	78.25975	0.369479	0.759083
50	83.316676	0.334688	0.783056
60	87.87783	0.334359	1.05997
70	92.062665	0.233021	0.857701
80	95.729016	0.148219	0.54554
90	98.517181	0.05281	0.370989
100	100	0.044911	0.368671

Analysis

Let’s plot a chart to visualize the comparison of the errors in tracking the NASDAQ index for different values of 'm' and how well the stock selection aligns with the optimization objective. It can help in determining the optimal value of 'm' that balances tracking accuracy and computational efficiency.



- The red dotted line on the graph represents our recommended value, denoted as 'm=40'. This particular value strikes an optimal balance, effectively addressing two crucial aspects of stock selection: maximizing the representation of the NASDAQ index and minimizing tracking errors for both 2019 and 2020.
- 'm=40' serves as a strategic choice, as it achieves the highest level of representation while keeping the number of selected stocks to a minimum.
- This also minimizes the risk associated with over-exposure to a large number of stocks while ensuring that the selected stocks play a substantial role in the portfolio's overall performance.

4. COMPARISION OF APPROACHES

In contrast to our initial approach, the second method represents a more precise and refined strategy. The key distinction lies in the way we tackle stock selection and weight assignment. In the first approach, we first picked the stocks and subsequently determined their respective weights. However, the second approach involves a reformulation of the weight selection process as a Mixed-Integer Programming (MIP) problem. This unique approach introduces a constraint on the number of non-zero weights, requiring them to be integers. This change enhances the accuracy and reliability of the stock selection process. By comparing these two approaches, we aim to discern which method better aligns with our goal of crafting a portfolio that accurately mirrors the NASDAQ-100 index while managing practical constraints.

Objective function

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

Constraint 1:

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

Constraint 2: Wi = 0 if yi = 0

$$w_i \leq y_i$$

Constraint 3: Sum of weights = 1

$$\sum_{i,1}^{m} w_i \ , 1$$

Choosing big M: We can set big M to 1 as the total weight we assign is equal to 1. This would be the ***smallest value of big M*** that can effectively constrain the model.

Code

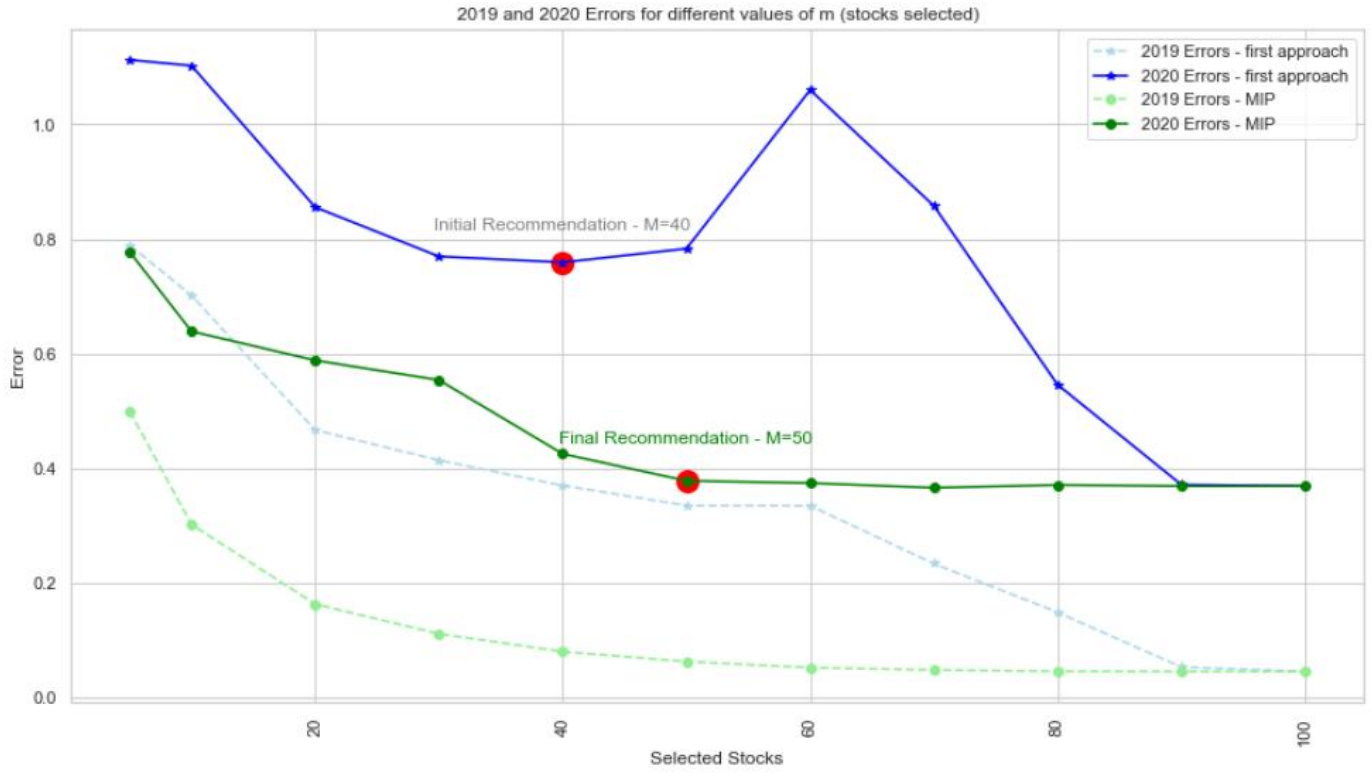
```
m_array = [ 5 , 10 , 20 , 30 , 40 , 50 , 60 , 70 , 80 , 90 , 100 ]
T = 60
M = 1 #The smallest value of Big M that can effectively constrain the model
stock_weight_dictionary_complex = {}
selected_stock_dictionary_complex = {}

for m in m_array :

    if n >= m :
        MIP_model = gp . Model ()
        select_stock = MIP_model . addMVar ( n , vtype = 'B' )
        stock_weight = MIP_model . addMVar ( n )
        index_difference = MIP_model . addMVar ( days_2019 )

        selection_const_complex = MIP_model . addConstr ( gp . quicksum ( select_stock [ i ] for i in range ( n )) == m )

        portfolio_weights_const_complex = MIP_model . addConstr ( gp . quicksum ( stock_weight [ i ] for i in range ( n )) == 1 )

        mod_const_1_complex = MIP_model . addConstrs (
            ( gp . quicksum ( stock_weight [ i ] * returns_2019 . iloc [ j , i + 1 ] for i in range ( n )) - index_2019 [ j ]) <= index_difference [ j ]
            for j in range ( days_2019 )
        )

        mod_const_2 = MIP_model . addConstrs (
            ( - 1 * gp . quicksum ( stock_weight [ i ] * returns_2019 . iloc [ j , i + 1 ] for i in range ( n )) + index_2019 [ j ]) <= index_difference [ j ]
            for j in range ( days_2019 )
        )

        big_m_const = MIP_model . addConstrs ( stock_weight [ i ] <= select_stock [ i ] * M for i in range ( n ))

        MIP_model . setObjective ( gp . quicksum ( index_difference [ i ] for i in range ( days_2019 )), sense = gp . GRB . MINIMIZE )
        MIP_model . setParam ( gp . GRB . Param . TimeLimit , T )
        MIP_model . Params . OutputFlag = 0
        MIP_model . optimize ()

        print ( ' \n ' , 'Number of Stocks selected = ' , m , ' | ' , 'Objective Value = ' , MIP_model . objval , ' \n ' )
        selected_stock_dictionary_complex [ m ] = selected_stock . x
        stock_weight_dictionary_complex [ m ] = stock_weight . x
```

Analysis

Comparing the results of two different stock selection approaches by tabulating the results.

Selected Stocks	Stock Selection Objective	2019 Errors (Basic Model)	2020 Errors (Basic Model)	2019 Errors (MIP Model)	2020 Errors (MIP Model)
5	54.841179	0.789178	1.112437	0.499259	0.777362
10	59.333081	0.701218	1.102404	0.301498	0.63855
20	66.648971	0.466233	0.855213	0.162433	0.587937
30	72.697515	0.413754	0.769412	0.110752	0.553649
40	78.25975	0.369479	0.759083	0.079619	0.424802
50	83.316676	0.334688	0.783056	0.062217	0.377808
60	87.87783	0.334359	1.05997	0.05172	0.373836
70	92.062665	0.233021	0.857701	0.047631	0.365585
80	95.729016	0.148219	0.54554	0.045227	0.3706
90	98.517181	0.05281	0.370989	0.044911	0.368682
100	100	0.044911	0.368671	0.044911	0.368671

Plotting the errors for different values of 'm' (the number of stocks selected) using two different stock selection approaches: the first approach (based on correlation) and the second approach (based on minimizing the difference between fund returns and the NASDAQ index returns)



- For both the years 2019 and 2020, the errors seem to decrease as the number of selected stocks increases. This is intuitive since selecting more stocks allows for a more accurate replication of the NASDAQ index.
- There's a noticeable difference in errors between the two approaches. The second approach (in green) appears to perform better (lower errors) than the first approach (in blue). This suggests that the second approach is more effective at replicating the NASDAQ index.
- This is expected to be the case as in the second method we are directly optimizing to minimize the difference between NASDAQ index returns and the stock selection is directly driven by this objective. Whereas, in the first approach, stock selection is based on a simplistic approach using correlation between stocks which doesn't factor in the index returns. Due to this difference, the second approach will always be better than the first.
- The initial recommendation was to select 40 stocks (m=40), but based on the results from the second approach, it seems that selecting 50 stocks (m=50) offers a significant reduction in error, especially for the year 2020. Hence, the final recommendation was updated to m=50.

By adjusting the approach to select 50 stocks instead of 40, the fund achieves a more accurate replication of the NASDAQ index, as evidenced by the reduced error. This improved tracking not

only boosts investor trust by offering a performance that closely mirrors the target index, but it also enhances portfolio flexibility, allowing for a more adaptive response to market movements. While there might be initial transaction costs associated with a broader stock selection, the fund stands to benefit in the long run.

The potential savings from reduced rebalancing costs, stemming from a better-aligned portfolio that minimizes divergence from the index, can offset these upfront expenses. In essence, the strategic shift to include 50 stocks optimizes both performance and cost, positioning the fund as a more efficient and trustworthy investment option.

5. RECOMMENDATIONS AND CONCLUSION

In the ever-evolving world of financial investments, our mutual fund strives to deliver consistent performance that aligns with market benchmarks. As an integral part of our ongoing efforts to refine our strategies, we undertook a comprehensive analysis to optimize our equity index fund. The primary objective was to ensure our fund closely replicates the NASDAQ index's performance. This report presents our findings and recommendations on the ideal number of component stocks and their weightage in our portfolio.

Recommendations:

Building on our findings, we put forth the following recommendations:

1. **Portfolio Composition:** We recommend a portfolio consisting of 50 component stocks. This composition not only aligns with our objective of closely tracking the NASDAQ index but also ensures manageability.
2. **Weight Distribution:** Our analysis provides an optimized weight distribution for each stock, with significant emphasis on stocks that exhibit high representativeness of the NASDAQ index.
3. **Striking a Balance Between Accuracy and Manageability:** While an 80 or 90-stock portfolio minimizes tracking error, it demands more extensive management due to a larger number of positions and potential transaction costs. In contrast, a 50-stock portfolio offers an optimal compromise between replication accuracy and ease of management. Hence, it’s important to strike a balance between them.

APPENDIX

1.

```
# Combine the 'NDX' columns into a single list
ndxs_combined = list(stocks_2019['NDX']) + list(stocks_2020['NDX'])

# Calculate the 60-day moving average
rolling_average = pd.Series(ndxs_combined).rolling(window=60).mean()

# Create a line chart with different colors for 2019 and 2020 NDX data
plt.figure(figsize=(20, 6))
colors = ['blue'] * len(stocks_2019) + ['green'] * len(stocks_2020)

for i in range(len(ndxs_combined) - 1):
    plt.plot([i, i + 1], [ndxs_combined[i], ndxs_combined[i + 1]], color=colors[i])

# Plot the moving average as a red dotted line with a label
plt.plot(rolling_average, label='60-Day Moving Average', color='red', linestyle='dotted')

plt.xlabel('Date')
plt.ylabel('NDX Value')
plt.title('NDX Value Over Time with 60-Day Moving Average')
plt.legend()
plt.grid(True)
plt.show()
```

2.

```
#Lets plot the fund return vs NASDAQ index return for 2019
fil_2019_returns  =  returns_2019 [ selected_columns_list ]
weight_x  =  portfolio_weights_df [ 'weights' ]
weighted_fil_2019_return  =  pd . DataFrame ()
for  i , col  in  enumerate ( fil_2019_returns . columns ):
    weighted_fil_2019_return [ col ]  =  fil_2019_returns [ col ] * weight_x [ i ]
weighted_fil_2019_return [ 'total_returns_fund' ]  =  weighted_fil_2019_return [ selected_columns_list ] . sum ( axis = 1 )

plt . figure ( figsize = ( 20 ,  10 ))

plt . plot ( weighted_fil_2019_return [ 'total_returns_fund' ],  label = '2019 Fund Return' ,   color = 'green' ,  linestyle = '--' )
plt . plot ( index_2019 ,  label = '2019 NASDAQ Index Returns' ,  marker = 'o' ,   color = 'orange' )
# Add a Legend
plt . legend ()

# Add labels and title
plt . xlabel ( 'Days' )
plt . ylabel ( 'Returns' )
plt . title ( '2019 - Fund Returns vs Index Returns' )
```

3.

```
returns_2020  =  stocks_2020 . copy ()   # Create a copy of the original DataFrame
returns_2020 . drop ( columns = [ 'Unnamed: 0' ],  inplace = True )   # Drop the 'Date' column if it exists
returns_2020  =  returns_2020 . pct_change ()
returns_2020  =  returns_2020 . fillna ( 0 )
index_2020  =  returns_2020 [ 'NDX' ]


#Now lets filter the 2020 return dataset for the selected stocks
fil_2020_returns  =  returns_2020 [ selected_columns_list ]
weight_x  =  portfolio_weights_df [ 'weights' ]
weighted_fil_2020_return  =  pd . DataFrame ()
for  i , col  in  enumerate ( fil_2020_returns . columns ):
    weighted_fil_2020_return [ col ]  =  fil_2020_returns [ col ] * weight_x [ i ]
weighted_fil_2020_return [ 'total_returns_fund' ]  =  weighted_fil_2020_return [ selected_columns_list ] . sum ( axis = 1 )
m_5_2020_error  =  sum ( abs ( returns_2020 [ 'NDX' ]  -  weighted_fil_2020_return [ 'total_returns_fund' ]))

print ( ' \n The difference between the 2019 NASDAQ Index and the fund we have created (Training error) = ' ,  round ( portfolio . objval , 2 ), ' \n ' )

print ( 'The difference between the 2020 NASDAQ Index and the fund we have created (Testing error) = '  ,  round ( m_5_2020_error , 2 ))


plt . figure ( figsize = ( 20 ,  10 ))

plt . plot ( weighted_fil_2020_return [ 'total_returns_fund' ],  label = '2020 Fund Return' ,   color = 'green' ,  linestyle = '--' )
plt . plot ( index_2020 ,  label = '2020 NASDAQ Index Returns' ,  marker = 'o' ,   color = 'orange' )
# Add a Legend
plt . legend ()

# Add labels and title
plt . xlabel ( 'Days' )
plt . ylabel ( 'Returns' )
plt . title ( '2020 - Fund Returns vs Index Returns' )
```