RM 294 - Optimization

Project 2 - Index Fund

Team	
Name	EID
Kehl Zeyu Li	zl9735
Vi Tran	vtt332
Dinesh Bandaru	db46729
Raghav Vaidya	rv25256

Contents

1.	Introduction and problem scope
2.	Requisite Data
3.	Stock Selection
4.	Stock weights in portfolio
5.	Performance evaluation
6.	Mixed-Integer Attempt
7.	Conclusion

Introduction and Problem Scope

An index fund is like a bundle that includes a variety of stocks or bonds. It aims to mimic the performance of a specific market index, such as the NASDAQ 100. By investing in an index fund, you essentially own a small piece of numerous companies, providing a diversified and simplified approach to investing. Few of the components of NASDAQ 100 are given below.

Symbol	Name	Sector
AAPL	Apple	Information Technology
ABBV	AbbVie	Health Care
ABT	Abbott	Health Care
ACN	Accenture	Information Technology
ADBE	Adobe	Information Technology
AIG	American International Group	Financials
AMD	AMD	Information Technology
AMGN	Amgen	Health Care
AMT	American Tower	Real Estate
AMZN	Amazon	Consumer Discretionary

In the world of managing money in stocks, there are two main strategies: "active" and "passive." The common passive strategy is called "indexing," where the idea is to create a portfolio that follows the ups and downs of the overall market or a specific market index. This portfolio is known as an index fund.

If you wanted to make an index fund that copies a whole bunch of stocks in a market index, you might think of just buying all of them with the same proportions they have in that index. However, that's not practical and can be costly due to various reasons like having too many small positions and the expenses of keeping everything in balance. So, instead of buying all the stocks (let's call them 'n'), people often choose a smaller number of stocks (let's call them 'm'), which is much more manageable and still does a good job of representing the larger market.

To accomplish this, we have used a combination of Linear and Integer Programming modeling to come up with a fund that closely follows the NASDAQ 100 index. We will discuss the approach in detail in the following sections.

Requisite Data

The dataset employed for model development encompasses the daily price of the NASDAQ-100 index and its constituent stocks throughout the calendar year of 2019. Subsequently, for performance evaluation purposes, the dataset corresponding to the year 2020, featuring identical stock constituents, will be utilized. Given below is a snapshot of the dataset, where the first column is the day (when the market is open), followed by the price of NASDAQ-100 and after that each column represents the prices of the individual stocks in the NASDAQ-100 index.

Day	NDX	ATVI	ADBE	AMD	...	WDC	XEL	XLNX
1/2/2019	6360.87	46.35038	224.57	18.83		36.31247	46.04186	84.60027
1/3/2019	6147.13	44.70451	215.7	17.05		33.83533	45.86045	81.41497
1/4/2019	6422.67	46.48836	226.19	19		35.26847	46.30922	85.18472
1/7/2019	6488.25	47.79914	229.26	20.57		36.52128	46.10871	87.43489
1/8/2019	6551.85	49.2479	232.68	20.75		36.17011	46.64341	85.76918
1/9/2019	6600.69	50.09547	235.43	20.19		37.76459	46.27103	86.53872
1/10/2019	6620.94	50.60795	237.77	19.74		37.78358	47.04445	88.02908
1/11/2019	6601.4	45.86746	237.55	20.27		37.99237	47.06353	88.56484
1/14/2019	6541.04	46.60663	234.56	20.23		36.12265	46.7198	88.21417
1/15/2019	6669.64	46.16313	242.36	20.38		35.762	47.63643	88.84732
⋮						⋮		
12/27/2019	8770.98	58.80326	330.79	46.18	...	60.84355	61.91041	97.20953
12/30/2019	8709.73	58.49529	328.34	45.52		61.08838	62.07715	96.47865

Stock Selection

The very first step in developing our own Index fund is to select m stocks from n stocks (m < n). Suppose m = 5 and n = 100. We would need each of the 100 stocks to have one representative from the 5 selected stocks. One way we can estimate the similarity of stocks is the correlation between the daily returns. To calculate the daily returns we use the formula below on consecutive rows for a given stock.

$$Daily\ Return = \frac{\Delta\ (price\ between\ two\ consecutive\ market\ days)}{(Price\ in\ the\ previous\ day)}$$

Implementation in python : `df.pct_change() * 100`

Day	NDX	ATVI	ADBE	AMD	...	WDC	XEL	XLNX
1/2/2019	NaN	NaN	NaN	NaN		NaN	NaN	NaN
1/3/2019	-3.36%	-3.55%	-3.95%	-9.45%		-6.82%	-0.39%	-3.77%
1/4/2019	4.48%	3.99%	4.86%	11.44%		4.24%	0.98%	4.63%
1/7/2019	1.02%	2.82%	1.36%	8.26%		3.55%	-0.43%	2.64%
1/8/2019	0.98%	3.03%	1.49%	0.88%		-0.96%	1.16%	-1.91%
1/9/2019	0.75%	1.72%	1.18%	-2.70%		4.41%	-0.80%	0.90%
1/10/2019	0.31%	1.02%	0.99%	-2.23%		0.05%	1.67%	1.72%
1/11/2019	-0.30%	-9.37%	-0.09%	2.68%		0.55%	0.04%	0.61%
1/14/2019	-0.91%	1.61%	-1.26%	-0.20%		-4.92%	-0.73%	-0.40%
1/15/2019	1.97%	-0.95%	3.33%	0.74%		-1.00%	1.96%	0.72%
		...					...	
12/27/2019	-0.08%	0.51%	-0.12%	-0.97%	...	-1.07%	0.53%	-0.26%
12/30/2019	-0.70%	-0.52%	-0.74%	-1.43%		0.40%	0.27%	-0.75%

The above data frame shows the returns of each stock, daily, for the year 2019. Now, based on this data we have to calculate the similarity using correlation between the stocks. In this process we will drop the NDX column as that is what we want to match based on the stocks we select. The similarity matrix is given below.

	ATVI	ADBE	AMD	ALXN		WDC	XEL	XLNX
ATVI	1.0000	0.3999	0.3653	0.2216		0.3030	0.0434	0.2496
ADBE	0.3999	1.0000	0.4528	0.3689		0.3615	0.2074	0.2894
AMD	0.3653	0.4528	1.0000	0.3018	...	0.4388	0.1728	0.4780
ALXN	0.2231	0.3689	0.3018	1.0000		0.2899	0.0479	0.2003
		...					...	
WDC	0.3030	0.3615	0.4388	0.2899		1.0000	-0.0761	0.5169
XEL	0.0433	0.2074	0.0172	0.0479	...	-0.0761	1.0000	-0.1175
XLNX	0.2496	0.2894	0.4780	0.2003		0.5169	-0.1175	1.0000

Implementation in python : `df.corr()`

Now that our data is processed, we need to formulate constraints such that:

1. We select exactly m stocks to be held in fund: Each stock would be represented by a binary decision variables y_j which indicated which stocks j from the index are present in the fund.

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

Implementation in python : `model.addConstr(gp.quicksum(y[i] for i in range(n)) == m)`

2. Make sure that each stock i has exactly one representative stock j in the index: We could do this by having a binary decision variable $x_{i,j}$ indicating which stock j in the index is the best representative of stock i.

$$\sum_{j=1}^{n} x_{ij} = 1 \; for \; i = 1, 2, 3, \ldots, n$$

Implementation in python : `model.addConstrs(gp.quicksum((x[i,j] for j in range(n))) == 1 for i in range(n))`

3. Guarantees that stock i is best represented by stock j only if j is in the fund: We have defined the relationship between the stock i and j with x_{ij} and the presence of stock j in fund by y_j. Hence we would have:

$$x_{ij} \leq y_j \; for \; i, j = 1, 2, 3, \ldots, n$$

Implementation in python : `model.addConstrs((x[i,j]<=y[j]) for j in range(n) for i in range(n))`

4. Finally, The objective of the model maximizes the similarity between the n stocks and their representatives in the fund. Let us call The individual elements of the similarity matrix ρ_{ij}.

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

Implementation in python : `model.setObjective(gp.quicksum(x[i,j] * correlation_matrix.iloc[i,j] for i in range(n) for j in range(n)) , sense = gp.GRB.MAXIMIZE)`

On running the optimization for $m = 5$, we get the recommended stocks as the following:

LBTYK	MXIM	MSFT	VRTX	XEL

Stock weights in portfolio

Now that we have the individual stocks we need to assign weights to each of them so as to minimize the difference between the returns over time period *t* on the actual index and our fund.

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

$$where,\ q_t\ is\ the\ returns\ of\ the\ index\ at\ time\ t$$

$$w_i\ is\ the\ weight\ of\ stock\ i$$

$$r_{it}\ is\ the\ return\ of\ stock\ i\ at\ time\ period\ t$$

Implementation in python :

The above equation is non-linear, so we must reformulate this optimization problem as a linear program. Suppose we have an absolute value $|x - 1|$, *we can* represent the absolute element by two linear inequalities, we can use the following formulation:

$$y \geq x - 1 \,\&\, y \geq 1 - x$$

Now we can minimize y

These two inequalities capture the conditions imposed by the absolute value. We would have to have this pair for every time period *t*

The constraints in python for our weight calculation would be:

```
model.addConstrs(Y[t] >= c*(q[t]-gp.quicksum(w[i]*r.iloc[t,i] for i in
range(m))) for t in range(T) for c in coef) where coef = [1,-1]
```

Hence, the objective in python for our weight calculation would be:

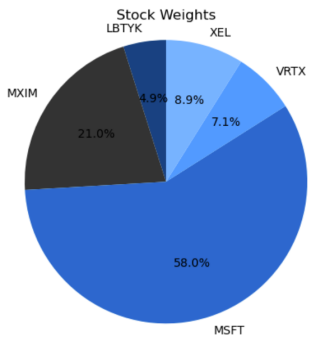
```
model.setObjective(gp.quicksum(Y[i] for i in range(T)),sense= gp.GRB.MINIMIZE)
```

We need one constraint to show the weights should sum up to a 100%

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

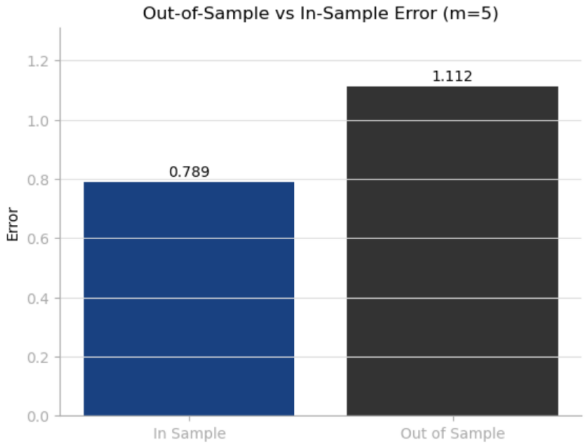
```
model.addConstr(gp.quicksum(w[i] for i in range(m)) == 1)
```

Upon executing the above equations, we get the weights of the stocks to be the following:

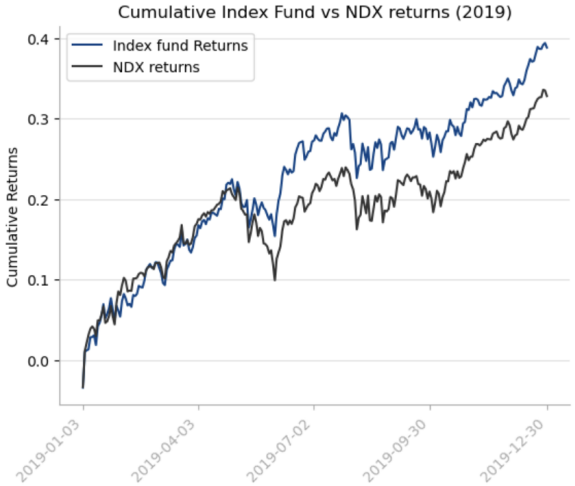


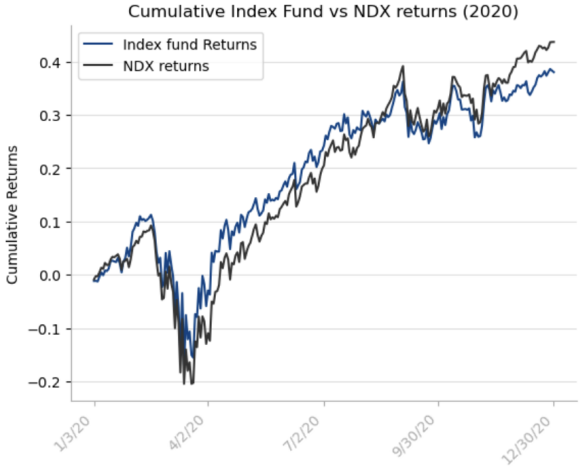
Performance evaluation

In this section we will take the weights calculated from the previous section and assess the performance on the data from the year 2020.

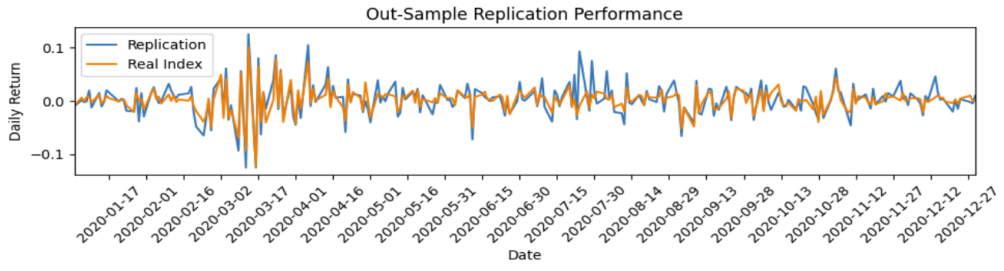


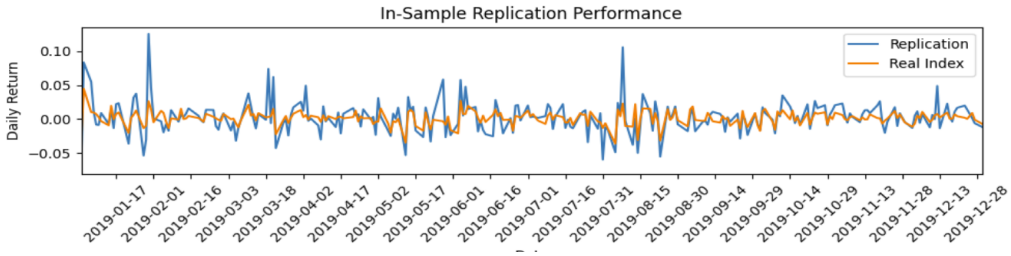
Below are two graphs showing how closely our index fund tracks the actual index over the year.(m=5)



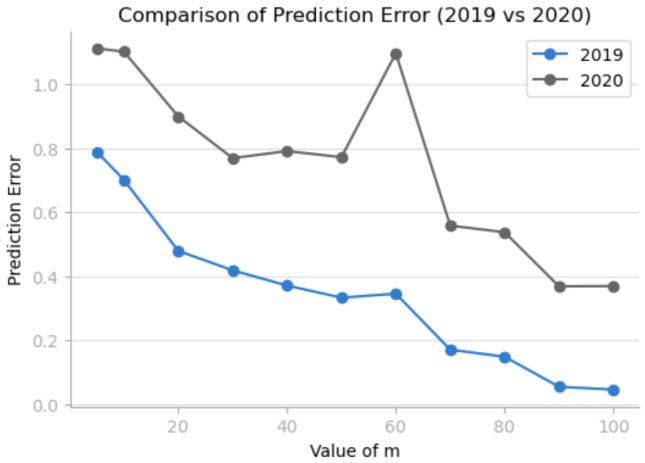


The two graphs show how well the NASDAQ index is replicated by our index fund.





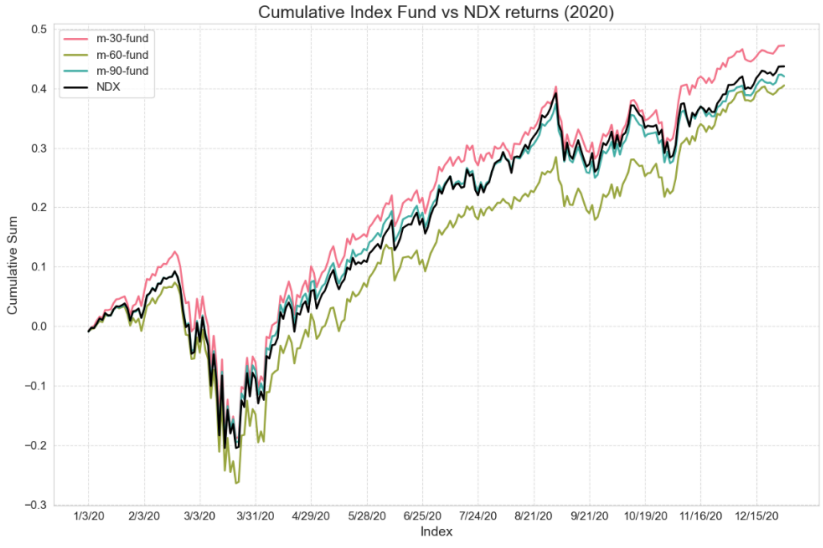
Now let's iterate through multiple values of *m* and assess the in-sample and out of sample performance. Let us take the values of *m* to be [5, 10, 20, 30, 40, 50, 60, 70, 80, 90].



Generally speaking, as the values of m increased, the performance of portfolios appears to improve as well. This was true in both the case for the 2019 dataset and the 2020 dataset. The performance using the 2019 data is significantly better than the 2020 across all values of m. The prediction error is the lowest, nearly reaching 0 at the highest values of 90 and 100. We believe this may be a case of slight overfitting as the 2019 data is the training one. It wouldn't be odd for it to perform better there, especially as there's no guarantee that trends in 2019 would hold for the following year.

There is, however, a large outlier in the case of m = 60. What we found was that the prediction error specifically when using that value spiked by a strong amount. This was especially true for the 2020 set, jumping as high as 1.09 and nearly reaching the peak amount of 1.11 set by an m value of 5 (what we discovered for the previous question). Several factors could have contributed to this. One, for example, is that our constraints may make it difficult to optimize at that specific value. Another could be that the data itself exhibits strange behavior at that amount of stocks.

As for where there may be diminishing returns of including more stocks in the portfolio, we found this to occur in two places. The first would be when m is equal to 30 since the error rate here and up towards 50 doesn't change much. The second occurrence is when m is equal to 90. Once again, from there to the maximum count for m, there is hardly any improvement. The following plot visualizes our significant findings:



Here, one can see the actual returns in comparison to what would happen if we used m-values of 30, 60, and 90. At an m of 90, we observe the highest accuracy. Meanwhile, 30 is slightly worse in comparison and 60 is almost always off the mark. Given what we have found, the best numerical option for stocks to include would be around 30 or 90.

Mixed-Integer Attempt

While creating a portfolio from a selection of stocks would be cheaper, they may not accurately reflect the index. As such, we can take another approach to consider every stock in the index by formulating the problem as a mixed-integer one. The new problem appears as follows:

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

$$where,\ q_t\ is\ the\ returns\ of\ the\ index\ at\ time\ t$$

$$w_i\ is\ the\ weight\ of\ stock\ i$$

$$r_{it}\ is\ the\ return\ of\ stock\ i\ at\ time\ period\ t$$

$$BigM\ constraint:\ 0\ \leq w_i \leq y_i M$$

Implementation in python :

The big difference here is the use of *n* instead of *m* as our problem now considers all weights and stocks rather than a selection of them. We defined a new model and while it shared many of the variables as the linear problem, there is a new binary variable defined as:

```
newY = new.addMVar(n, vtype='B')
```

To solve this problem, we created 2 new distinct constraints:

1. This made sure that the sum of y's was equal to m:

```
num_con = new.addConstr(sum(newY)==m)
```

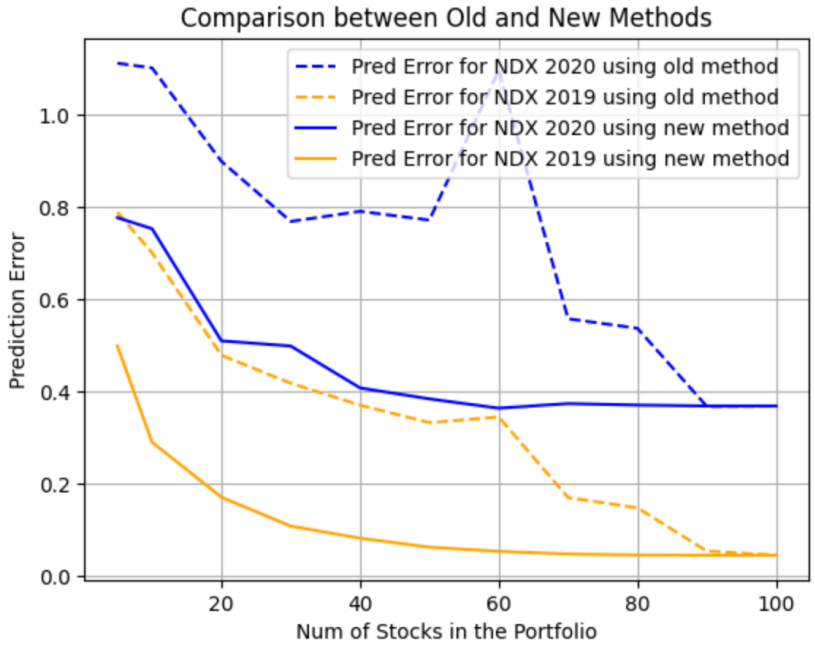
2. This made sure that $w_i = 0\ and\ y_i = 0$ while utilizing big M:

```
M_con = new.addConstrs(newW[i]<=newY[i]*M for i in range(n))
```

In confining M to be the smallest value we can use, we decided to try it setting it to 1 and m as 5 initially to do a test for both in and out-of-sample errors before running it for every 10th value of m to discover the following:

m	forward	back
5.0	0.77736	0.49926
10.0	0.75337	0.29014
20.0	0.51001	0.17081
30.0	0.49879	0.10798
40.0	0.40786	0.08183
50.0	0.38405	0.06233
60.0	0.36388	0.05326
70.0	0.3739	0.04758
80.0	0.37063	0.04523
90.0	0.36868	0.04491
100.0	0.36867	0.04491

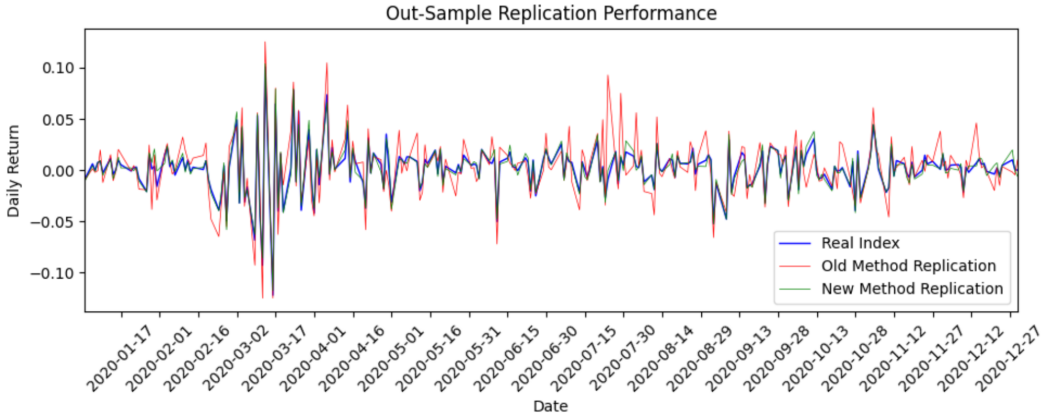
The forward and back columns here refer to out-of-sample and in-sample errors respectively. In the context of our data, forward is the errors for the 2020 dataset and back is the errors for the 2019 dataset. This is better seen in a plot that compares these results to that of the linear problem method:



For both datasets, the new method utilizing big M performed better across every value of m. They taper out to very similar values at 100 values, but at every point, the new method is a significant improvement providing less error. Although it may be a more expensive approach, it is worth the cost.

Conclusion

Having tested two separate methods for selecting weights, and testing multiple values of stocks to include in the portfolio, we have decided that it's best to use the MIP method that considers the weights of all the stocks.



Across the board, it performed far more accurately than the old method of selecting stocks to include and focusing on those weights. When using this method, 60 is the amount of stocks where we found there to be the lowest amount of error. Meanwhile, 40 is the amount of stocks where we begin to see error only slightly diminish. Should we wish to closely replicate the NASDAQ index, we recommend 60 stocks. If we wish to sacrifice accuracy over resources, 40 will work great as well.

Should we lack the time and money entirely to reliably use this, the selection method can also be considered. If using that instead, we recommend selecting either 30 or 90 stocks to include in the portfolio. 90 would provide the best accuracy but it is also more costly than its alternative. We would, however, advise caution against this method, as going forward with it for other projects, spikes in error that we observed here could appear again.

These are the finalized results of our findings. We hope that our recommendations will be beneficial and will hold for the years to come.