# Optimization
# Project 2 – Integer Programming

**Group 5:** April Hyeonjin Kim, Aishwarya Rajeev, Muskaan Singhania, Yvonne Wang

## Introduction

To gain insights to improve the efficiency of our strategies in equity money management, we wanted to create an index fund with m stocks from the NASDAQ-100 index (NDX) that track and perform as similar as NDX as possible, in a way that it minimizes the difference in return between the fund we created and the NDX.

We implemented two different methods to achieve this goal. The first method was to first select m stocks that are best representatives, that have the highest correlation scores to the stocks in 2019, assign weights, and then evaluate the performance of our fund using the index in 2020. The second method, however, was to first assign weights to stocks without selecting the stocks yet, find the combination that would minimize the difference in return between our fund and the NDX, and then evaluate the performance using the index in 2020.

Finally, we investigated the results of each method to compare our fund's performance using the index in both 2019 and 2020 and to make recommendations on which method to use, the optimal number of stocks to select, and the list of suggested stocks and weights when creating a fund that tracks the NDX the best.

## Methods

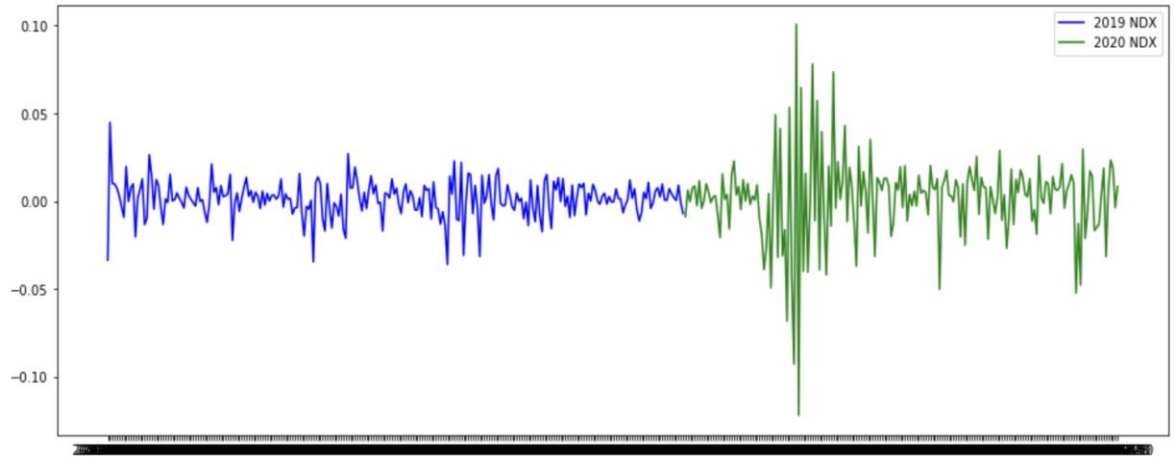
### The First Method:

As mentioned in the introduction, with fewer equities than the index, we tried to establish an index fund that closely mirrors the performance of the NASDAQ-100. This could be done by creating an integer program to choose precisely m stocks for the fund out of the available n stocks. In order to minimize the absolute discrepancies between the returns on the portfolio and

the returns on the index, we also utilized a linear algorithm to determine the best weights for these stocks.

First, we determined the daily stock returns for each stock in our dataframes for 2019 and 2020 stocks. As can be seen in the figure below, in 2020 the NDX returns seemed to be more volatile than in 2019. Returns were more risky and more rewarding for most parts of the year.

```
#Calcuating returns
stocks2019=stocks2019.set_index('X').pct_change()
stocks2020=stocks2020.set_index('X').pct_change()
```




Next, we created a similarity matrix that calculates the correlation between the returns of every pair of stocks, $\rho_{ij}$. This matrix allows us to determine which stocks are most comparable and best serve as an index representation.

**Decision variables:**

$y_j$ - 1/0 if stock *j* is present in the fund or not (*n* variables)

$x_{ij}$ - 1/0 if stock *j* in index is the best representative of stock *i* ($n^2$ variables)

**Objective function:**

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

**Constraints:**

- Only m stocks in the index must be held in the fund (1 constraint)
- Each stock i must have only one representative stock j in the index (n constraints)
- Stock i is best represented by stock j only if stock j is in the fund (n2 constraints)

Next step was to calculate the portfolio weights. For this, we needed to formulate a linear program to identify the optimal weights.

**Objective fn:**

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

Where:

- □it : the return of stock i (where stock i is the chosen stock ) at time period t
- □t : the return of the index at time t
- □i : weight of stock i in the portfolio

Now that all of the functions for choosing stocks and determining the best weights have been defined, we used m = 5 to calculate performance and then acquired the five top stocks for our portfolio, along with their weights.

```
df2 = pd.DataFrame(columns = ['stock', 'weight'])
df2['stock'] = list
df2['weight'] = weight
df2
```

| | stock | weight |
|---|---|---|
| 0 | LBTYK | 0.04319 |
| 1 | MXIM | 0.18638 |
| 2 | MSFT | 0.51670 |
| 3 | PAYX | 0.18985 |
| 4 | VRTX | 0.06388 |

We found that the 5 best stocks to include in our index fund are Microsoft (MSFT), Maxim (MXIM), Paychex (PAYX), Vertex Pharmaceuticals (VRTX) and Liberty Global PLC Class C(LBTYK). Based on 2019 data, we also calculated the weights for each stock - MSFT at 51.67%, MXIM at 18.64%, PAYX at 18.99%, VRTX at 6.39%, and LBTYK at 4.32%.

To see how well these 5 stocks track the index fund in 2020, we evaluated the performance by figuring out the difference between the index and portfolio returns. By providing the performance evaluation function with these 5 stocks and their respective weights, we obtained a difference of 0.96.

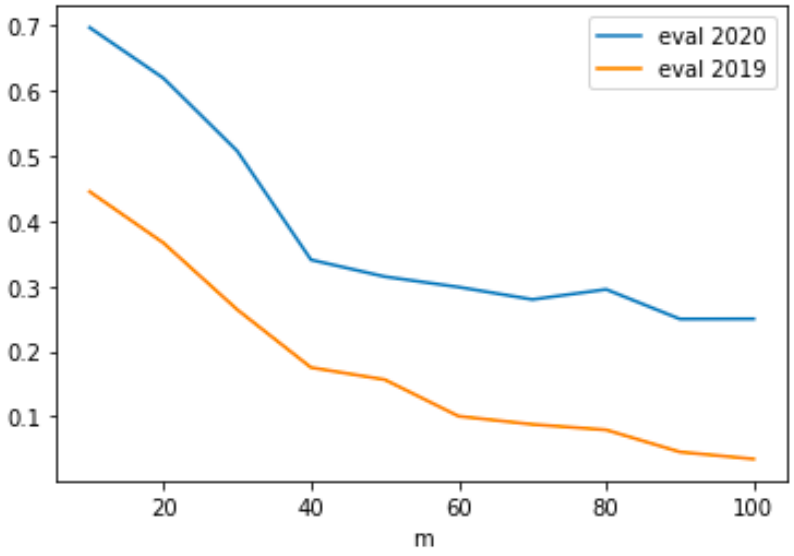
```
#performance evaluation
check_df = stocks2020[['NDX'] + list].iloc[1:,:]
T = len(check_df)
eval = 0
for time in range(T):
    qt = check_df.iloc[time]['NDX']
    return_val = check_df.iloc[time][list].to_list()
    eval += abs(qt - np.dot(weight, return_val))
eval
```

```
0.958124895061994
```

We wanted to reduce this value as much as we can as it is pretty high. A decline in this inaccuracy would indicate that our portfolio is considerably getting better at tracking the market. The small number of stocks in the portfolio could be the main cause of this high error, as the performance of the portfolio is anticipated to more closely resemble the index as additional stocks are added. To anticipate a reduction in this mistake, we tracked the performance of the portfolio for a range of values of m starting from 10 to 100. We also used the in-sample data (the NDX in 2019). The below plot shows the trend of the error vs. the number of stocks in the portfolio for 2019 and 2020.

```
df = pd.DataFrame(columns = ['m', 'eval 2020', 'eval 2019'])
df['m'] = m_list_valid
df['eval 2020'] = eval_2020
df['eval 2019'] = eval_2019
#plotting the results
print(df.plot.line(x='m', y=['eval 2020', 'eval 2019']))
df
```

| | m | eval 2020 | eval 2019 |
|---|---|---|---|
| **0** | 10 | 0.69697 | 0.44524 |
| **1** | 20 | 0.61934 | 0.36652 |
| **2** | 30 | 0.50753 | 0.26420 |
| **3** | 40 | 0.34056 | 0.17531 |
| **4** | 50 | 0.31478 | 0.15672 |
| **5** | 60 | 0.29874 | 0.10059 |
| **6** | 70 | 0.27972 | 0.08810 |
| **7** | 80 | 0.29519 | 0.07961 |
| **8** | 90 | 0.24973 | 0.04574 |
| **9** | 100 | 0.24994 | 0.03490 |




Using the in-sample data, we observed a steady decline in the error as we increase m, which aligned with our prediction that as the number of stocks increases, the portfolio more closely resemble the index. The error was the highest at 0.45 when m = 10 and the lowest at 0.03 when m = 100 (for m ranging from 10 to 100). The trend for the 2020 data, which is out-of-sample data, is a little more intriguing. The error initially declined similarly to the in-sample data, but there was a sharp increase in error at 80 stocks. This was most likely caused by the weights being trained on data from 2019 but tested on data from 2020. This could imply that the weights in our model, which were tightly adjusted to the 2019 data, may have overfitted, making it difficult to use the data from prior years. The error for the in-sample data was the highest at 0.70 when m = 10 and the lowest at 0.25 when m = 90 (for m ranging from 10 to 100).

### The Second Method:

Another method to approach was to completely disregard the stock selection IP and reframe the weight selection problem as a MIP with an integer-based constraint on the number of non-zero weights. This could be accomplished by taking the weight selection issue and changing m to n to optimize overall weights.

**Objective:**

We developed a linear program to determine the ideal weights, which were selected ideally by taking into account greatest similarity with index stocks, to minimize the difference in returns between the index and portfolio. The objective function is expressed as follows:

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

Where:

- □it : the return of stock i (where stock i is the chosen stock ) at time period t
- □t : the return of the index at time t
- □i : weight of stock i in the portfolio

Since the objective function is nonlinear, we can translate the problem into

$$\min_{w} \sum_{t}^{n} z_t$$

**Decision Variables:**

- wi: the weight of the stock i that selected by the portfolio

**Constraints:**

- Sum of weights for wi for n stocks in the fund adds to 1
- New constraints added by transferring the non-linear program to the linear program

$$z_t \geq q_t - \sum_{i=1}^{n} w_i r_{it} \longrightarrow \sum_{i=1}^{n} w_i r_{it} + z_t \geq q_t$$

$$z_t \geq \sum_{i=1}^{n} w_i r_{it} - q_t \longrightarrow -\sum_{i=1}^{n} w_i r_{it} + z_t \geq -q_t$$

Define □□ as whether stock □ is in fund. And have big M as 1

## Results：

We first used m = 5 to calculate performance and then acquired the five top stocks for our portfolio, along with their weights.

```
df2 = pd.DataFrame(columns = ['stock', 'weight'])
df2['stock'] = list
df2['weight'] = weight
df2
```

| | stock | weight |
|---|---|---|
| 0 | GOOGL | 0.11610 |
| 1 | AMZN | 0.17096 |
| 2 | ADI | 0.11567 |
| 3 | AAPL | 0.16077 |
| 4 | MSFT | 0.27765 |

We found that the 5 best stocks to include in our index fund are Google (GOOGL), Amazon (AMZN), Analogue Devices (ADI), Apple (AAPL), and Microsoft (MSFT). We also calculated the weights for each stock - MSFT at 27.77%, AMZN at 17.10%, AAPL at 16.08%, GOOGL at 11.61%, and ADI at 11.57%.

We evaluated the performance by figuring out the difference between the index and portfolio returns. By providing the performance evaluation function with these 5 stocks and their respective weights, we obtained a difference of 0.62.

```
check_df = stocks2020[['NDX'] + list].iloc[1:,:]
T = len(check_df)
eval = 0
for time in range(T):
    qt = check_df.iloc[time]['NDX']
    return_val = check_df.iloc[time][list].to_list()
    eval += abs(qt - np.dot(weight, return_val))
eval
```

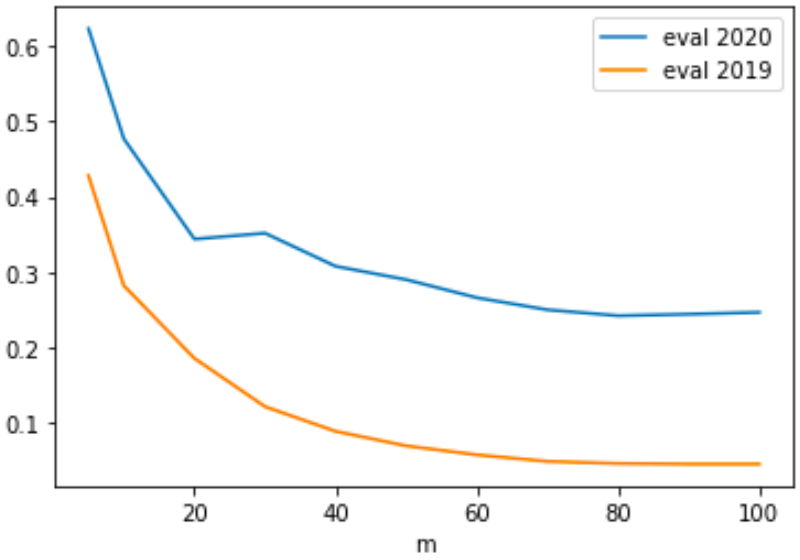
```
0.6232735438946407
```

Similar to what we did for the first method, we tracked the performance of the portfolio for a range of values of m starting from 10 to 100. We also used the in-sample data (the NDX in

2019). The below plot shows the trend of the error vs. the number of stocks in the portfolio for 2019 and 2020.

```
df3 = pd.DataFrame(columns = ['m', 'eval 2020','stocks','weights'])
df3['m'] = m_list_valid
df3['eval 2020'] = eval_2020_new
df3['eval 2019'] = eval_2019_new
df3['stocks'] = stock_list
df3['weights'] = weight_list
#plotting the results
print(df3.plot.line(x='m', y=['eval 2020', 'eval 2019']))
df3
```

| | m | eval 2020 | eval 2019 |
|---|---|---|---|
| 0 | 5 | 0.62327 | 0.42787 |
| 1 | 10 | 0.47695 | 0.28183 |
| 2 | 20 | 0.34367 | 0.18526 |
| 3 | 30 | 0.35150 | 0.12106 |
| 4 | 40 | 0.30753 | 0.08828 |
| 5 | 50 | 0.28977 | 0.06896 |
| 6 | 60 | 0.26575 | 0.05706 |
| 7 | 70 | 0.24969 | 0.04832 |
| 8 | 80 | 0.24184 | 0.04554 |
| 9 | 90 | 0.24386 | 0.04477 |
| 10 | 100 | 0.24635 | 0.04477 |




Using the in-sample data, we observed a steady decline in the error as we increase m, which aligned with our prediction that as the number of stocks increases, the portfolio more closely resemble the index. The error was the highest at 0.28 when m = 10 and the lowest at 0.04 when m = 90 (for m ranging from 10 to 100). The error did not decrease when m was increased from 90 to 100. Looking at the trend for the 2020 data, which is out-of-sample data, the error initially declined similarly to the in-sample data, but there was an increase in error at 30 stocks and then resumed to decrease steadily. The error for the in-sample data was the highest at 0.48 when m = 10 and the lowest at 0.24 when m = 80. Note that the error indeed increased when m was increased from 90 to 100.

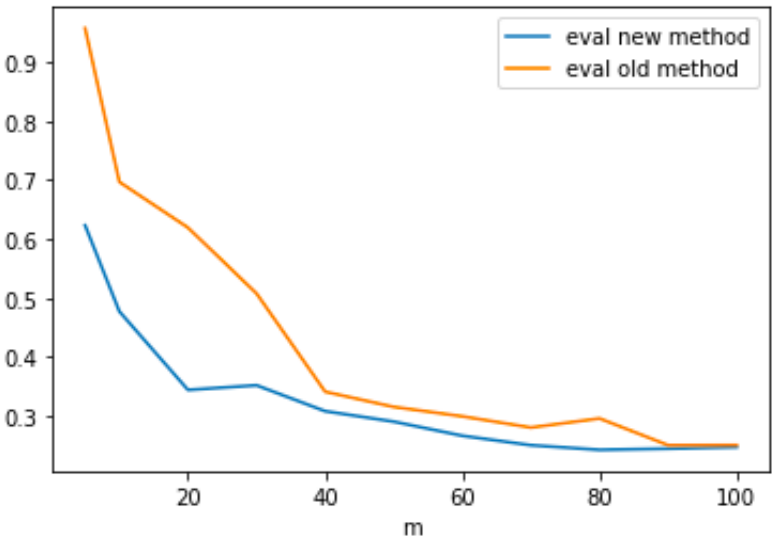
The list of selected stocks and weights allocated to each stock for the selected ms are shown in the chart below.

| | m | eval 2020 | stocks | weights |
|---|---|---|---|---|
| 0 | 5 | 0.62327 | GOOGL,AMZN,ADI,AAPL,MSFT | 0.11610273477399062,0.1709554488036626,0.11567... |
| 1 | 10 | 0.47695 | ADBE,ALXN,GOOG,AMZN,AAPL,FB,INTC,MSFT,TXN,WBA | 0.08231606750030371,0.02688551808054051,0.1023... |
| 2 | 20 | 0.34367 | ADBE,GOOGL,AMZN,AMGN,AAPL,AMAT,BIIB,BKNG,CSCO,... | 0.044894490033269105,0.08972134820884689,0.105... |
| 3 | 30 | 0.35150 | AMD,GOOGL,AMZN,AMGN,ADI,AAPL,ADSK,ADP,BIDU,BII... | 0.011493424201520497,0.08134038333073726,0.098... |
| 4 | 40 | 0.30753 | ATVI,ADBE,AMD,ALXN,GOOGL,AMZN,AMGN,AAPL,ADSK,A... | 0.007249434933375022,0.023368515302869243,0.00... |
| 5 | 50 | 0.28977 | ATVI,ADBE,AMD,ALXN,GOOGL,GOOG,AMZN,AMGN,ADI,AA... | 0.004067180033176219,0.025403973449725815,0.00... |
| 6 | 60 | 0.26575 | ATVI,ADBE,AMD,GOOGL,AMZN,AMGN,AAPL,ASML,ADP,BI... | 0.0037720108846738478,0.01603204286105352,0.00... |
| 7 | 70 | 0.24969 | ATVI,ADBE,AMD,ALXN,GOOGL,GOOG,AMZN,AMGN,AAPL,A... | 0.005062499748702437,0.020055266252574366,0.00... |
| 8 | 80 | 0.24184 | ATVI,ADBE,AMD,ALXN,GOOGL,GOOG,AMZN,AMGN,AAPL,A... | 0.0041211749263102 77,0.01853325252277896,0.003... |
| 9 | 90 | 0.24386 | ATVI,ADBE,AMD,ALXN,GOOGL,GOOG,AMZN,AMGN,AAPL,A... | 0.004116858850271998,0.016918516969962546,0.00... |
| 10 | 100 | 0.24635 | ATVI,ADBE,AMD,ALXN,GOOGL,GOOG,AMZN,AMGN,AAPL,A... | 0.004071696999234904,0.016987984368372837,0.00... |

Finally, we wanted to compare the errors of the different selected ms of the first and the second method to figure out which method performed better by plotting the figure below. We referred to the first method as ‘Old Method’ and the second method as ‘New Method.’

```
#Comparing the old and the new method
df5 = pd.read_csv("Weights_after_run.csv")
df4=df5.merge(df, on='m', how='inner')
df4.rename(columns={"eval 2020_x": "eval new method", "eval 2020_y": "eval old method"}, inplace=True)
df4.plot.line(x='m', y=['eval new method','eval old method'])
df4[['m','eval new method','eval old method']]
```

| | m | eval new method | eval old method |
|---|---|---|---|
| 0 | 5 | 0.62327 | 0.95812 |
| 1 | 10 | 0.47695 | 0.69697 |
| 2 | 20 | 0.34367 | 0.61934 |
| 3 | 30 | 0.35150 | 0.50753 |
| 4 | 40 | 0.30753 | 0.34056 |
| 5 | 50 | 0.28977 | 0.31478 |
| 6 | 60 | 0.26575 | 0.29874 |
| 7 | 70 | 0.24969 | 0.27972 |
| 8 | 80 | 0.24184 | 0.29519 |
| 9 | 90 | 0.24386 | 0.24973 |
| 10 | 100 | 0.24635 | 0.24994 |




The error rates were much higher for the first method for all selected ms with many fluctuations, while the second method showed relatively consistent performance with lower error rates.

## Recommendation

When we want to create a fund that performs similar to the NDX the best, we recommend using the second method ('New Method') for a better performance. By comparing the two methods, we figured out that the second method had much lower error rates with a more consistent performance, while the first method had much higher error rates with many fluctuations.

According to the results of our investigation, we recommend selecting 70 stocks for the best performance. The performance measured for the second method barely increased by increasing the number of selected stocks after 70 stocks, and the performance even decreased by selecting 80 stocks for the first method. In overall, we suggest selecting **70 stocks** from the index using **the second method** to create a fund that performs the best.

The list of 70 stocks that should be selected and each stock's weight follows:

ATVI,ADBE,AMD,ALXN,GOOGL,GOOG,AMZN,AMGN,AAPL,AMAT,ASML,ADSK,ADP,BIDU,BIIB,BMRN,BKNG,AVGO,CDNS,CHTR,CSCO,CTXS,CMCSA,COST,CSX,DOCU,DLTR,EBAY,FB,FAST,GILD,ILMN,INTC,INTU,ISRG,JD,KLAC,KHC,LBTYK,LULU,MAR,MXIM,MELI,MCHP,MU,MSFT,MDLZ,NTES,NFLX,NVDA,ORLY,PCAR,PAYX,PYPL,PEP,QCOM,REGN,ROST,SIRI,SBUX,SNPS,TMUS,TTWO,TSLA,TXN,ULTA,VRSN,VRSK,VRTX,WBA,XLNX

0.004332380803533093,0.0197024464984867 3,0.0036365548463160187,0.0030865441492019495,0.05195335055249279,0.031473182360112636,0.09652223646606897,0.014387493141013372,0.10305857983009997,0.004501509717679636,0.003397945799160843,0.00532523358431775,0.019998106457755085,0.004554071525990547,0.006318130287951131,0.00610286329123208,0.014004136817949698,0.01327084417400749,0.0058089927618710435,0.007586922224267183,0.026196715003188306,0.005460963590969584,0.024233435495410065,0.01184796076610116,0.010353553976720824,0.0013621676925259754,0.006680061204977771,0.006156799221375189,0.04950077986756185,0.00179704962777631,0.012340016124790597,0.006095781267756706,0.031621363347086394,0.008868012036412999,0.011627493554453928,0.004581005807167302,0.006386512320988713 5,0.006349094461726947,0.002679175139771039 7036,0.00178112158105517,0.006862197315498467,0.012108948355967342,0.001147327571051685 2,0.005525771806094352,0.00626018780121719,0.1001435292461882,0.01726173444766813,0.0026332357970785835,0.017703839085509165,0.0098578822204777

4,0.006496308444378226,0.001567711084307409,0.009397188802283778,0.020939216124533422,0.011519577444587556,0.011279319958274294,0.006535952940094334,0.005407629004391002,0.0049703645440954585,0.007722542348733207,6.184305317891982e-16,0.010270275025927204,0.002975242963536429,0.005917223120668975,0.012225514501982306,0.003524256228113314,0.010268688744571416,0.0059796661531409554,0.00787078195750895,0.009897065317858542,0.004896391516339011