Project 2 – Integer Programming

**Group 10: Ping Zhang (pz3238), Daniel (Mu-An) Shen (ms85344),
Sam LaPlatney (sjl2953), Soumik Choudhuri (sc64856)**

Objectives

Our goal is to recommend an index fund that tracks the NASDAQ-100 as closely as possible while also keeping the number of component stocks in the fund to a minimum. To accomplish this, we will utilize two optimization methods. For a specified number of component stocks, the optimizers will choose a portfolio that minimizes the absolute difference between the overall index return and the overall portfolio return. The optimizers will choose both the stocks to be included in the funds and the respective weights of those stocks. We will then recommend a specific portfolio based on an analysis of 22 portfolios (11 from each method).

Methods

For each method, the problem will be mathematically formulated as an integer optimization problem. The following steps will be required in each of the methods:

1) Gather daily prices of the index and the component stocks of the NASDAQ-100 in 2019 and 2020, calculate the returns of the component stocks and the index in 2019 and 2020, as well as correlation matrix of stock returns;

2) Obtain optimal solutions using 2019 price data, and evaluate the solutions using 2020 price data;

3) Compare evaluation results to decide which method works better and make recommendations based on the results.

Next, we will introduce each optimization method in detail and our analysis of the results from each method. Please refer to Appendix for more details about coding.

Method 1

Model Building

Method 1 includes two stages: stock selection and portfolio weight calculation. This method aims to select stocks that are the best representatives of the other stocks, the composition of selected stocks that mimics index overall performance the closest, and the minimum number of the stocks to represent the index well.

- **Stage 1: Stock Selection**

To select the stocks which are the best representatives of the other stocks in the index, we seek to maximize the correlation between the selected stocks and each of the other stocks. Stage 1 can be formulated as the following:

Decision variables: binary variables xij, yj

- xij: For each stock in the index i =1,...,100, the binary decision variables xij indicate which stock j in the index j=1,...,100, is the best representative of stock i. Since we have 100 component stocks, we will have 100*100 xij variables.
- yj: The binary decision variables yj indicate which stocks j from the index are present in the fund. And we will have 100 yj variables.

Objective function: maximize sum product of xij and its corresponding correlation coefficient.

The objective of the model maximizes the similarity between the n stocks and their representatives in the fund.

Constraints:

1. select the exact number of stocks to be held in the fund
2. each stock i has exactly one representative stock j in the index
3. guarantees that stock i is best represented by stock j only if j is in the fund

- **Stage 2: Portfolio Weight Calculation**

In this stage, we approach allocating the optimal weights to the selected stocks by minimizing the difference between the weighted stock return and the index overall return. In order to transform this integer program with absolute value, we created y variables for convenience to solve this difference minimization. Our team formulate the linear program optimization as the following:

Decision variables: binary variable yi, continuous variable wi

- yi: The yi variables symbolize the difference between index overall return and weighted stock return. Since we have 250 daily returns for each stock and the index, we will have 250 yi variables.
- wi: The weights for each of the selected stocks. And we will have 100 wi variables.

Objective function: Minimize sum of yi

Minimize the sum of absolute value of differences between index overall return and sum product of weighted stock return.

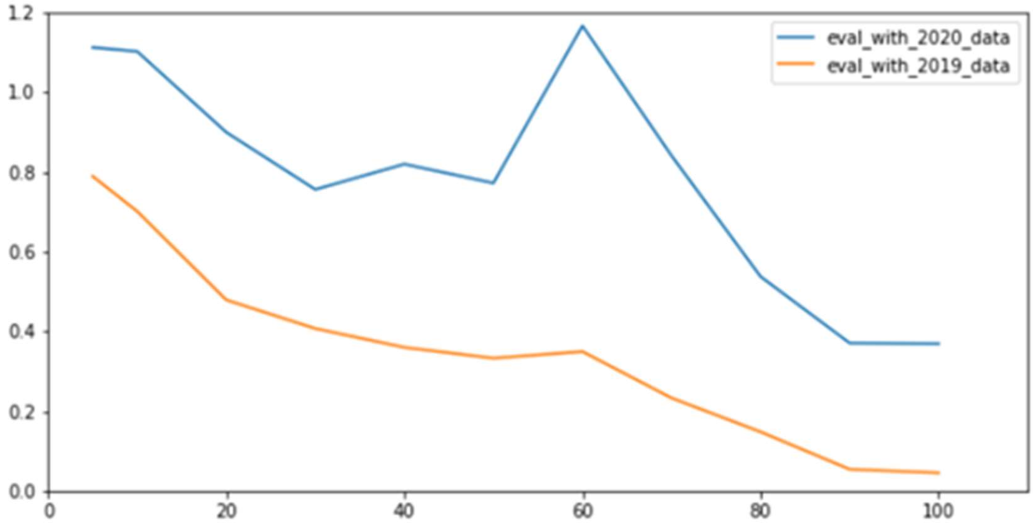
Constraints:

1. yi ≥ | index_return - weighted stock return |, which can be translated into:
   yi ≥ index_return - weighted stock return
   yi ≥ weighted stock return - index_return
2. Sum of wi = 1

In/Out-of- sample Evaluation

After establishing integer programming model as above, we set m (the number of stocks in the portfolio) to 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100, iterated through different numbers of stock selected, and evaluated in-sample and out-of-sample performance of method 1 using 2019 and 2020 data. **When we first set m to 5, we obtain in-sample error of 78.92%, and out-of-sample error of 111.24%.** As we increase the number of stocks included in the portfolio, overall the return of the portfolio becomes closer to the index overall return. **The in-sample error is between 4.49% and 78.92%. The out-of-sample error is between 36.87% and 111.24%.** However, we find that method 1 shows different patterns with 2019 and 2020 data. When evaluated using 2020 data, the return difference goes down with fluctuation before m reaches 60, and shows a significant increase when m is 60, then decrease rapidily to minimum as m increases to 90. When evaluated using 2019 data, the return difference goes down with littile fluctuation. **In general, method 1 performs better with 2019 data than with 2020 data.** We think this is because the optimal solutions are obtained using 2019 data. Within the same economic environment and financial period, the difference between the index and the portfolio dereases as the size of portfolio increases. While when it comes to 2020, affected by COVID-19 and changes in economics, the correlations among stocks may change too. Therefore, the portfolio selected based on 2019 correlations cannot track the movements of the NASDAQ-100 index in a good manner.

Although the selected stocks and weights work well with the 2019 data, it was not as stable when applied to the 2020 data. Significant fluctuations in test data accuracy and constantly increasing accuracy on training data made it difficult to pinpoint an optimal number of stocks to recommend.



In conclusion, although Method 1 could successfully select and allocate stocks and their weights, it did not provide us with any strong evidence for recommending a portfolio of a certain size.

Method 2

Because method 1 has a significant increase in out-of-sample performance at m=60, we are led to believe that method 1 is not robust. In optimization method 2, we completely ignore stage 1 of the first method (stock selection) and reformulate stage 2 (weight selection) to be an MIP that constrains the number of non-zero weights to be an integer. To be concise, method 2 is based on the weight selection model of method 1 but optimizes over all weights at the same time. Method 2 is formulated and evaluated as the following:

Model Building

Decision variables: binary variables xi and yi, continuous variable wi

- xi: The binary decision variable xi symbolizes the difference between index overall return and weighted stock return. Since we have 250 daily returns for each stock and the index, we will have 250 xi variables.
- yi: The binary decision variable yi indicates whether stock i from the index is included in the fund. Since we have 100 component stocks, we will have 100 yi variables.
- wi: The weights for each of the selected stocks. And we will have 100 wi variables.

Objective function: Minimize sum of xi

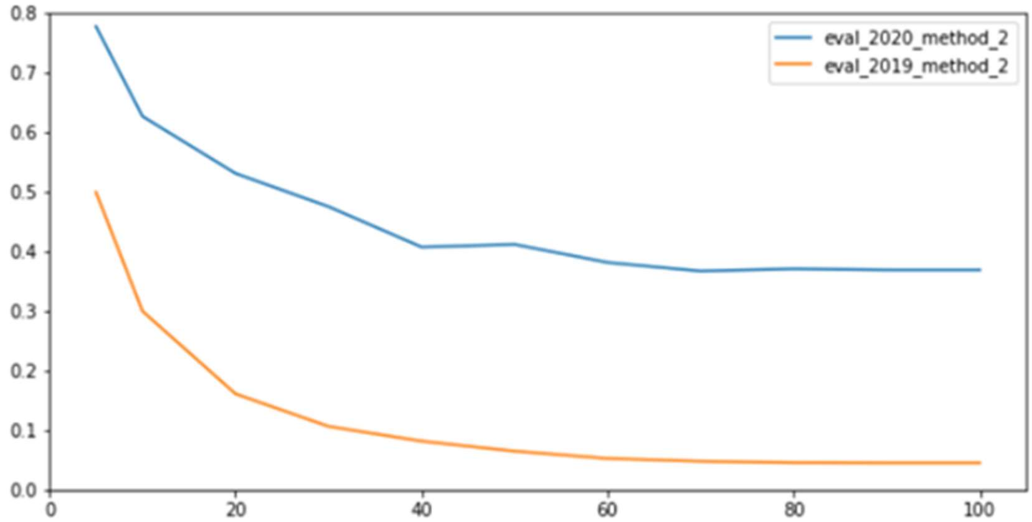
Minimize the sum of absolute value of differences between index overall return and sum product of weighted stock return.

Constraints:

1. xi ≥ | index_return - weighted stock return |, which can be translated into:
   xi ≥ index_return - weighted stock return
   xi ≥ weighted stock return - index_return
2. M * yi - wi ≥ 0
   - When yi is 0, wi shall be 0 too because there is no need to assign weight to a stock we do not choose. When yi is 1, wi can vary from 0 to 1 but no more than yi. Thus, we get a constraint as 0 < wi <yi.
   - Theoretically we can set M to 1. However, after tying different M (i.e. 1, 1.01, 1.05, 10, 20, 100), **we find the smallest value of M we could use is 1.002**, which is very close to 1. We believe this is due to rounding or boundary issues with Gurobi so that we cannot set M to 1. **Finally we decide to set M to 2.**
3. Sum of wi = 1
4. Sum of yi = m, m is the number of stocks we want to include in our portfolio.

In/Out-of- sample Evaluation

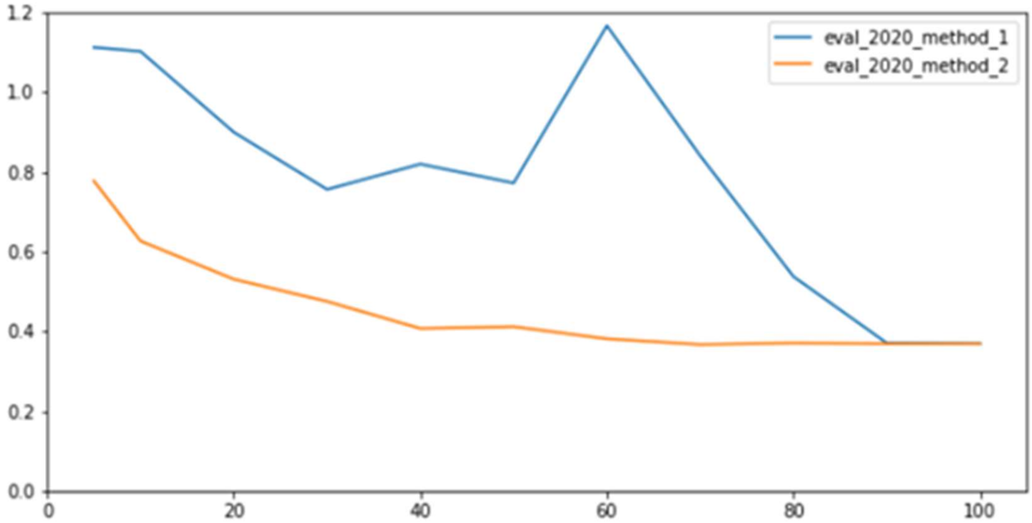
After establishing integer programming model as described above, we set m (number of stocks) to 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100, iterated through different numbers of stock selected, and evaluated in-sample and out-of-sample performance of method 2 using 2019 and 2020 data. As we increase the number of stocks included in the portfolio, it is obvious that the return of the portfolio will get closer to the index overall return. The in-sample error is between 4.49% and 49.93%. The out-of-sample error is between 36.87% and 77.74%. **In terms of both in-sample and out-of-sample performance, method 2 works better than method 1.** We think this is because method 2 focuses on how to select stocks among all component stocks to minimize the return difference without being restricted by correlations. Therefore, although economic environment changed in 2020, the portfolios are less affected and can mirror the movements of the NASDAQ-100 index in an accurate manner.



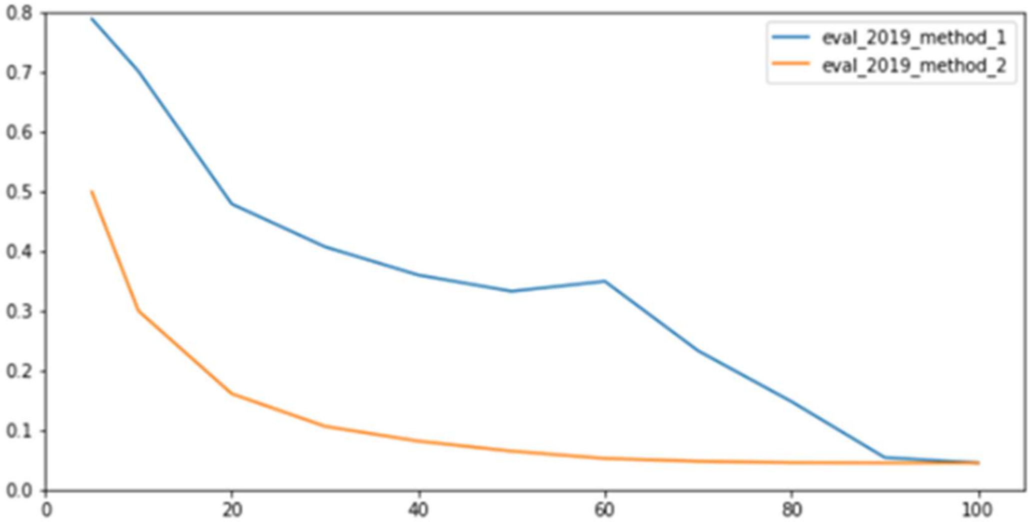
In the graph above, we can see that using method 2 the return difference between portfolio and index goes down much faster with portfolio size compared to method 1. Additionally, there is little to no fluctuation in the error of both the train and test sets using method 2. The slope of these error curves will also make it easier to identify an optimal number of stocks because the error levels off at around m=40 in both training and testing data.

Recommendations

To evaluate the performance of our two optimization methods, we compared the difference between our fund's return and the index's return on 2020 (out of sample) data. In the graph below we plotted this difference for each method, with varying sizes of the number of stocks in our fund. The graph illustrates when using method 2, the difference between fund return and index steadily decreases with fund size until a size of about 40 and never rises above a 100% difference. Whereas, the line for method 1 shows unpredictable fluctuation in the difference between fund return and index return. Based on these results we moved forward with method 2 as our preferred method for optimization.



In the graph below which compares the in sample accuracies for each method, we can see that using method 2 the return difference between portfolio and index goes down much faster with portfolio size compared to method 1. Additionally, the error curve is much smoother using method 2 which will make it easier to identify an optimal number of stocks because the error levels off at around m=40 in both training data (graph below) and testing data (graph above).



The goal in our recommendation is to minimize both portfolio size and the difference between index return and portfolio return. As shown in the graph above, the difference between index and fund returns levels off at a portfolio size of 40 stocks. Any decrease in difference after this point is insignificant when compared to the number of additional stocks required to achieve this decrease. **Thus, we will recommend a portfolio size of 40 stocks using optimization method 2.**

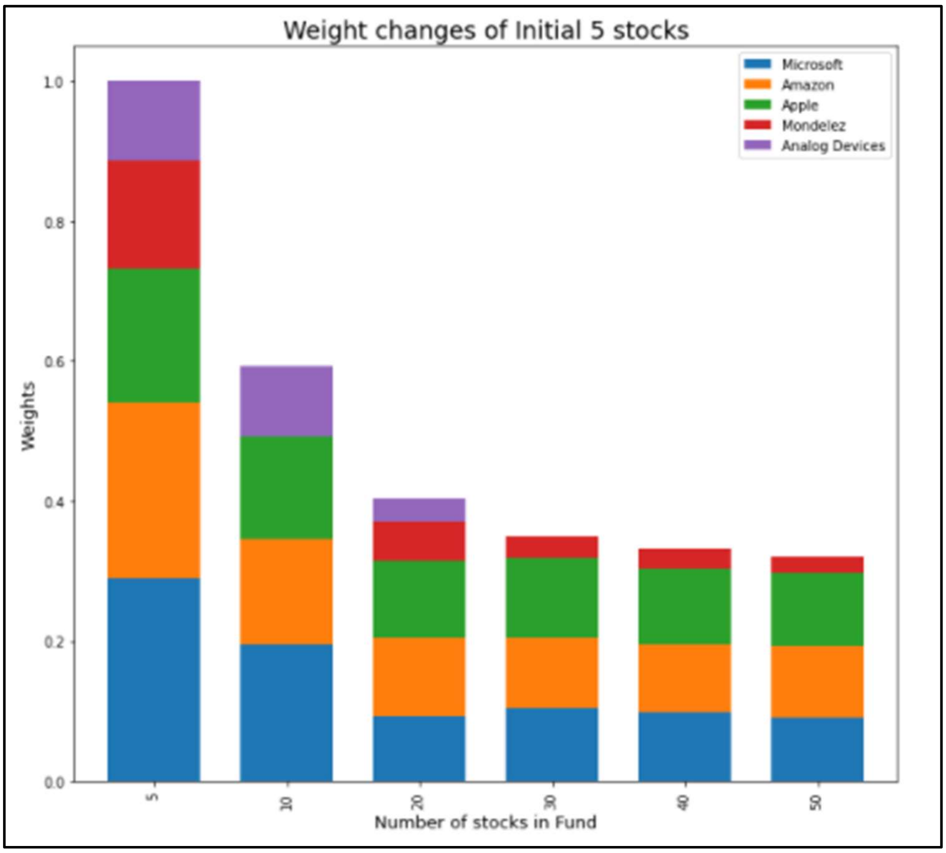
The following data frame displays the 40 stocks chosen and their corresponding weights in descending order.

	Chosen Stocks	Chosen Weights
0	AAPL	10.82%
1	MSFT	9.81%
2	AMZN	9.79%
3	GOOG	7.88%
4	FB	5.45%
5	INTC	3.30%
6	MDLZ	2.84%
7	CSCO	2.78%
8	ADBE	2.76%
9	TXN	2.73%
10	PAYX	2.65%
11	CTXS	2.12%
12	ADP	2.06%
13	CMCSA	2.02%
14	GILD	1.99%
15	NFLX	1.84%
16	AVGO	1.76%
17	AMGN	1.74%
18	BKNG	1.70%
19	PYPL	1.69%
20	PEP	1.68%
21	COST	1.67%
22	AMAT	1.61%
23	VRSN	1.58%
24	CSX	1.52%
25	TMUS	1.41%
26	NVDA	1.36%
27	ILMN	1.25%
28	BMRN	1.11%
29	MU	1.10%
30	EBAY	1.09%
31	CHTR	1.03%
32	WBA	0.84%
33	QCOM	0.82%
34	JD	0.74%
35	BIDU	0.74%
36	LULU	0.68%
37	BIIB	0.68%
38	ULTA	0.68%
39	TSLA	0.66%

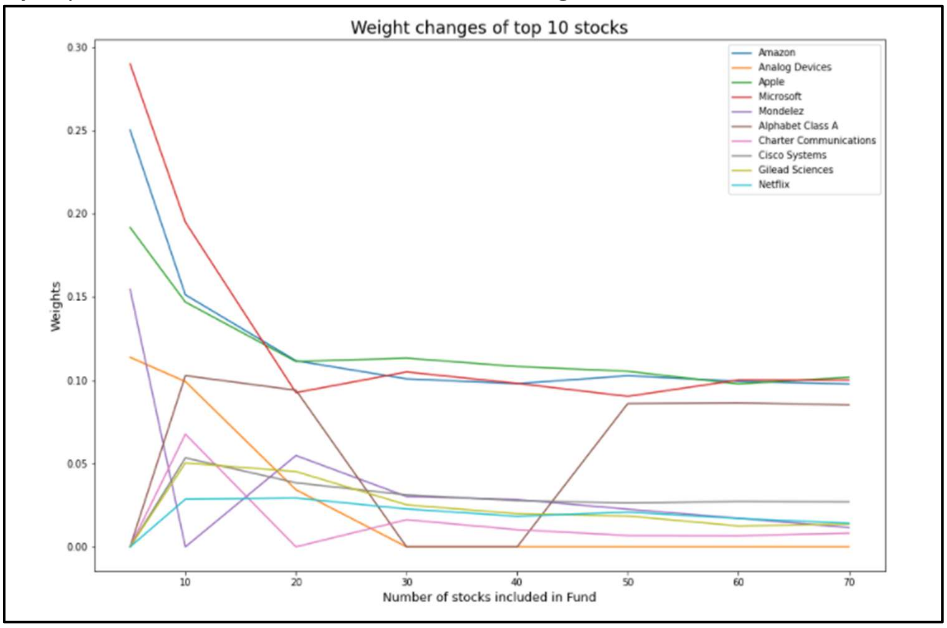
Further Observations

We want to include the following graphs to give you a better idea of our recommendation.

The first graph shows the “initial 5 stocks” (stocks that are selected when portfolio size is 5) and their respective weights under different portfolio sizes. Some interesting points here show that “Mondelez” drops out of the portfolio when the size increases from 5 to 10. It then reenters the portfolio at sizes 20-50. We can also see that “Analog Devices” is present in portfolios of size 5,10 and 20, but drops out of the portfolio at sizes 30 through 50. Additionally, we see that the relative weights of Amazon, Apple, Microsoft, and Mondelez remain stable in portfolios of sizes 30, 40, and 50. This provides insight as to why the difference between index returns and fund returns begins to level off in this same range.



The second graph below shows the same information, but includes the initial 10 stocks (stocks selected when portfolio size is 10). You can see that the weights begin to level out at 30 for all stocks except Alphabet. Alphabet takes a significant increase in weight when portfolio size jumps from 40 to 50. However, the difference between fund return and index return when size is 40 and 50 is virtually the same, meaning this jump had no effect on difference over this range.



Appendix

```python
import numpy as np
import pandas as pd
import gurobipy as gp
import matplotlib.pyplot as plt

time = 3600  # set time limit for gurobi (3600s = 1h)
```

A.1 Calculate the returns of the stocks in the 2019 file

```python
# compute daily returns using pandas pct_change()
daily_return_2019 = df_2019.pct_change()  # daily_return_2019 = (df_2019-df_2019.shift(1))/df_2019.shift(1)
# skip first row with NA
daily_return_2019 = daily_return_2019[1:]
daily_return_2019.head()
```

X	NDX	ATVI	ADBE	AMD	ALXN	ALGN	GOOGL	GOOG	AMZN	AMGN	...	TCOM	ULTA	VRSN	VRSK
2019-01-03	-0.033602	-0.035509	-0.039498	-0.094530	0.022030	-0.085791	-0.027696	-0.028484	-0.025242	-0.015216	...	-0.022834	-0.018591	-0.034989	-0.030557
2019-01-04	0.044824	0.039903	0.048632	0.114370	0.057779	0.010445	0.051294	0.053786	0.050064	0.034184	...	0.058976	0.047954	0.044744	0.044147
2019-01-07	0.010211	0.028196	0.013573	0.082632	0.018302	0.017192	-0.001994	-0.002167	0.034353	0.013457	...	0.022067	0.062620	0.016312	0.001000
2019-01-08	0.009802	0.030309	0.014918	0.008751	0.006207	0.015954	0.008783	0.007385	0.016612	0.012824	...	0.010281	0.018450	0.036460	0.008902
2019-01-09	0.007454	0.017210	0.011819	-0.026988	0.012430	0.038196	-0.003427	-0.001505	0.001714	-0.001196	...	0.023745	0.018804	-0.008157	0.003781

5 rows × 101 columns

A.2 Calculate the correlation matrix P for 100 stocks

```python
df = daily_return_2019.drop('NDX',axis=1)
p = df.corr()#round(decimals=4)
p.head()
```

	ATVI	ADBE	AMD	ALXN	ALGN	GOOGL	GOOG	AMZN	AMGN	ADI	...	TCOM	ULTA	VRSN	VRSK	VRTX
ATVI	1.000000	0.399939	0.365376	0.223162	0.216280	0.433097	0.426777	0.467076	0.203956	0.329355	...	0.322906	0.128241	0.464850	0.316549	0.25967
ADBE	0.399939	1.000000	0.452848	0.368928	0.363370	0.552125	0.540404	0.598237	0.291978	0.473815	...	0.360392	0.201151	0.711339	0.541243	0.402171
AMD	0.365376	0.452848	1.000000	0.301831	0.344252	0.418861	0.417254	0.549302	0.151452	0.503733	...	0.332776	0.210623	0.498342	0.330900	0.27298
ALXN	0.223162	0.368928	0.301831	1.000000	0.332433	0.315993	0.307698	0.363170	0.342022	0.317040	...	0.257143	0.408936	0.350581	0.191489	0.52242
ALGN	0.216280	0.363370	0.344252	0.332433	1.000000	0.248747	0.250316	0.399281	0.264599	0.328280	...	0.175957	0.128559	0.360886	0.251855	0.33497

5 rows × 100 columns

A.3 Calculate the returns of the stocks in the 2020 file

```
df_2020 = pd.read_csv('stocks2020.csv')
df_2020.set_index(['Unnamed: 0'],inplace=True)

# compute daily returns using pandas pct_change()
daily_return_2020 = df_2020.pct_change()  # daily_return_2020 = (df_20120-df_2020.shift(1))/df_2020.shift(1)
# skip first row with NA
daily_return_2020 = daily_return_2020[1:]
daily_return_2020.head()
```

Unnamed: 0	NDX	ATVI	ADBE	AMD	ALXN	ALGN	GOOGL	GOOG	AMZN	AMGN	...	TCOM	ULTA	VRSN	VR:
1/3/20	-0.008827	0.000341	-0.007834	-0.010183	-0.013260	-0.011421	-0.005231	-0.004907	-0.012139	-0.006789	...	-0.021369	-0.017207	0.021095	0.0097
1/6/20	0.006211	0.018238	0.005726	-0.004321	0.001598	0.019398	0.026654	0.024657	0.014886	0.007674	...	-0.013543	0.003118	0.009259	0.0022
1/7/20	-0.000234	0.010043	-0.000959	-0.002893	0.002533	-0.009864	-0.001932	-0.000624	0.002092	-0.009405	...	0.045951	0.008528	0.002318	0.0083
1/8/20	0.007452	-0.007623	0.013438	-0.008705	0.016191	0.010386	0.007118	0.007880	-0.007809	0.000756	...	-0.012323	0.019400	0.004626	0.0092
1/9/20	0.008669	-0.009018	0.007636	0.023834	0.019893	0.036853	0.010498	0.011044	0.004799	0.002980	...	0.006781	0.021318	0.023169	0.0096

5 rows × 101 columns

Method 1

Define global variables

```
N = len(p)  # number of stocks in the index
m = 5
m_all = [5,10,20,30,40,50,60,70,80,90,100]  # number of stocks we pick for portfolio
M = 2   #1.002 is the smallest value of big M we could use
```

B.1 Find the best m stocks to include in portfolio

```
obj = [] #Xij=N**2, Yij=N
for i in range(N):
    for j in range(N):
        obj.append(p.iloc[i,j])
obj = np.array(obj+[0]*N)

A = np.zeros((N+N**2+1,N**2+N)) #Xij=N, Xij<Yij=N, Yij=M
b = np.zeros(N+N**2+1)
direction = np.array([' ']*(N+N**2+1))

ind_vec = np.array(range(N))
row = 0
for j in range(N):
    A[row,j*N + ind_vec] = 1
    b[row] = 1
    direction[row] = '='
    row+=1

for i in range(N):
    for k in ind_vec*N +i:
        A[row, k] = 1
        A[row, -N+i] = -1
        b[row] = 0
        direction[row] = '<'
        row+=1

A[-1,-N:] = 1
direction[row] = '='
```

```
stock_idx_all = []

for m in m_all:
    b[row] = m

    portMod = gp.Model()
    portMod_x = portMod.addMVar(N**2+N,vtype=['B']*(N**2+N))
    portMod_con = portMod.addMConstrs(A, portMod_x, direction, b)
    portMod.setMObjective(None,obj,0,sense=gp.GRB.MAXIMIZE)
    portMod.Params.OutputFlag = 0 # tell gurobi to shut up!!
    portMod.optimize()

    stock_idx = []
    for i in range(N):
        if portMod_x.x[-N:][i] == 1:
            stock_idx.append(i)
    stock_idx_all.append(stock_idx)
    print('For m = '+str(m)+', stock indexes are:', stock_idx)
```

```
For m = 5, stock indexes are: [56, 59, 63, 94, 98]
For m = 10, stock indexes are: [0, 4, 40, 53, 56, 59, 63, 79, 94, 98]
For m = 20, stock indexes are: [0, 4, 5, 10, 15, 17, 30, 36, 40, 51, 53, 56, 59, 63, 64, 72, 76, 91, 94, 98]
For m = 30, stock indexes are: [0, 1, 5, 12, 15, 17, 19, 24, 28, 34, 35, 36, 39, 46, 51, 53, 55, 57, 59, 63, 64, 66, 72, 75, 76, 77, 86, 8
8, 91, 94]
For m = 40, stock indexes are: [0, 1, 4, 5, 8, 12, 15, 17, 19, 23, 24, 25, 29, 31, 34, 35, 36, 37, 46, 51, 53, 55, 57, 59, 60, 63, 64, 65,
66, 71, 72, 76, 77, 81, 86, 88, 91, 94, 95, 98]
For m = 50, stock indexes are: [0, 1, 3, 4, 5, 8, 12, 15, 17, 22, 23, 24, 25, 28, 29, 30, 31, 32, 34, 35, 36, 37, 39, 40, 44, 46, 51, 53,
55, 57, 59, 60, 63, 64, 66, 68, 71, 72, 75, 76, 77, 79, 80, 81, 84, 86, 88, 91, 94, 95]
For m = 60, stock indexes are: [1, 2, 3, 4, 5, 8, 12, 15, 16, 17, 18, 22, 23, 24, 25, 27, 28, 29, 30, 31, 32, 34, 35, 36, 37, 38, 40, 44,
45, 46, 47, 50, 53, 56, 57, 58, 59, 60, 64, 65, 66, 67, 68, 71, 72, 76, 77, 79, 80, 81, 84, 85, 86, 87, 88, 90, 91, 94, 95, 98]
For m = 70, stock indexes are: [0, 2, 3, 4, 5, 8, 12, 15, 16, 17, 18, 19, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36,
37, 38, 39, 40, 41, 42, 44, 45, 46, 47, 50, 51, 53, 56, 57, 58, 60, 61, 62, 63, 64, 65, 66, 67, 68, 71, 75, 76, 77, 79, 80, 81, 83, 84, 8
6, 88, 90, 91, 94, 95, 96, 98]
For m = 80, stock indexes are: [0, 2, 3, 4, 5, 8, 11, 12, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 50, 51, 53, 55, 57, 58, 60, 61, 62, 63, 64, 65, 66, 67, 68, 71, 72, 75, 76, 7
7, 78, 79, 80, 81, 83, 84, 86, 87, 88, 90, 91, 93, 94, 95, 96, 98, 99]
For m = 90, stock indexes are: [0, 1, 2, 3, 4, 6, 7, 8, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 22, 23, 24, 25, 26, 27, 28, 29, 30, 3
1, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 53, 55, 57, 58, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69,
71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 83, 84, 85, 86, 87, 88, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]
For m = 100, stock indexes are: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28,
29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 6
3, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97,
98, 99]
```

B.2 Get the weights of those stocks with m = 5, 10, 20, …, 90, 100, using the 2019 data

```
weightMod_x_all = []
objVal_all = []

for stock_idx in stock_idx_all:
    NDX_return_2019 = daily_return_2019['NDX']
    stocks_return_2019 = daily_return_2019.iloc[:,[i + 1 for i in stock_idx]]

    num1 = len(stocks_return_2019)  #250
    num2 = len(stock_idx) #5
    obj = np.array([1]*num1+[0]*num2) #yi=250, wi=5

    A = np.zeros((num1*2+1,num1+num2))
    b = []
    direction = np.array(['>']*(num1*2)+['='])

    row = 0
    col = 0
    for i in range(num1*2):
        if row % 2 == 0:
            A[row,col] = 1
            A[row,-len(stock_idx):] = stocks_return_2019.iloc[col].tolist()
            b.append(NDX_return_2019.iloc[col])
            row+=1
        else:
            A[row,col] = 1
            A[row,-len(stock_idx):] = [i*(-1) for i in stocks_return_2019.iloc[col].tolist()]
            b.append((NDX_return_2019.iloc[col])*(-1))
            row+=1
            col+=1
    A[row,-len(stock_idx):] = [1]*len(stock_idx)
    b.append(1)
```

```

    weightMod = gp.Model()
    weightMod_x = weightMod.addMVar(num1+num2)
    weightMod_con = weightMod.addMConstrs(A, weightMod_x, direction, b)
    weightMod.setMObjective(None,obj,0,sense=gp.GRB.MINIMIZE)
    weightMod.Params.OutputFlag = 0 # tell gurobi to shut up!!
    weightMod.optimize()

    weightMod_x_all.append(weightMod_x.x[-len(stock_idx):])
    objVal_all.append(weightMod.objVal)
    #print(weightMod_x.x[-len(stock_idx):])

print(objVal_all)
```

```
[0.7891782824631473, 0.7012177959266304, 0.47883578791133574, 0.40702064157423623, 0.3597098446202794, 0.3325400929154757, 0.3492171000088
9666, 0.23270620779515877, 0.14768252800096132, 0.05382736809219517, 0.04491081639360345]
```

B.3 Evaluate portfolio using 2020 data

```
eval_2020 = []

for i in range(len(m_all)):
    stock_idx = stock_idx_all[i]
    weightMod_x = weightMod_x_all[i]

    NDX_return_2020 = daily_return_2020['NDX'].tolist()
    stocks_return_2020 = daily_return_2020.iloc[:,[i + 1 for i in stock_idx]]

    A = stocks_return_2020.to_numpy()
    B = weightMod_x
    portfolio_return_2020 = pd.DataFrame(A*B).T.sum().tolist()

    sum = 0
    for j in range(len(portfolio_return_2020)):
        if NDX_return_2020[j] - portfolio_return_2020[j] > 0:
            sum += NDX_return_2020[j] - portfolio_return_2020[j]
        else:
            sum += portfolio_return_2020[j] - NDX_return_2020[j]
    eval_2020.append(sum)

print(eval_2020)  #difference in return between portfolio and the index
```

```
[1.1124373455076468, 1.1024044007151053, 0.8995975248049078, 0.7560018038680235, 0.8195922138576562, 0.7720996592107853, 1.166752195643991
5, 0.8406088378670545, 0.5373227700587152, 0.37050577624951, 0.36868191990333]
```

B.4 In-sample and out-of-sample performance

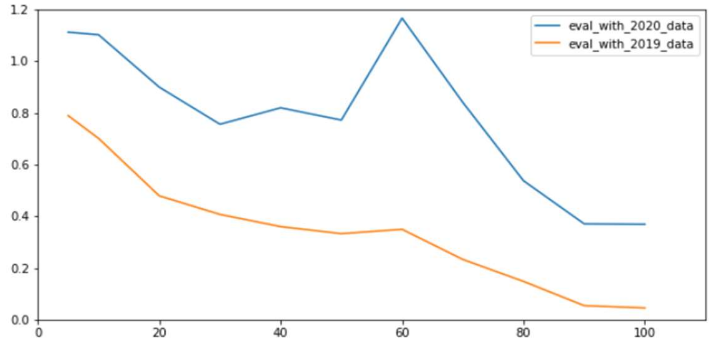
```
df1['evaluation_by_2019'] = eval_2019  # we can use Model.objVal directly
df1
```

	num_of_m	evaluation_by_2020	evaluation_by_2019
0	5	1.112437	0.789178
1	10	1.102404	0.701218
2	20	0.899598	0.478836
3	30	0.756002	0.407021
4	40	0.819592	0.359710
5	50	0.772100	0.332540
6	60	1.166752	0.349217
7	70	0.840609	0.232706
8	80	0.537323	0.147683
9	90	0.370506	0.053827
10	100	0.368682	0.044911

```
df1.set_index('num_of_m',inplace=True)
```

```
plt.figure(figsize=(10,5))
plt.plot(df1)
plt.legend(['eval_with_2020_data', 'eval_with_2019_data'])
plt.xlim([0, 110])
plt.ylim([0, 1.2])
```

```
(0.0, 1.2)
```



Method 2

C.1 Find the best m stocks to include in portfolio

```
num = len(stocks_return_2019) # 250 time slot for 2019
```

```
NDX_return_2019 = daily_return_2019['NDX']
stocks_return_2019 = daily_return_2019.drop('NDX',axis=1)

obj = np.array([1]*num+[0]*N*2) #xi=250, wi=100, yi=100

A = np.zeros((num*2+N+2,num+N*2))
b = []
direction = np.array(['>']*(num*2+N)+['=']*2)

row = 0
col = 0
# fill first (num*2) rows for A
for i in range(num*2):
    if row % 2 == 0:
        A[row,col] = 1
        A[row,num:(num+N)] = stocks_return_2019.iloc[col].tolist()
        b.append(NDX_return_2019.iloc[col])
        row+=1
    else:
        A[row,col] = 1
        A[row,num:(num+N)] = [i*(-1) for i in stocks_return_2019.iloc[col].tolist()]
        b.append((NDX_return_2019.iloc[col])*(-1))
        row+=1
        col+=1
```

```
# fill next N rows for A
for j in range(N):
    A[row,num+j] = -1
    A[row,num+N+j] = M
    b.append(0)
    row+=1
# add a constraint that sum of wi shall be 1
A[row,num:(num+N)] = [1]*N
b.append(1)

# add a constraint that sum of yi shall be m
A[row+1,(num+N):(num+N*2)] = [1]*N
b.append(0)

```

```python
# for loop for different m
eval_2019 = []
weights = []
for m in m_all:
    b[-1] = m

    weightMod = gp.Model()
    weightMod_x = weightMod.addMVar(len(obj),vtype=['C']*(num+N)+['B']*N)
    weightMod_con = weightMod.addMConstrs(A, weightMod_x, direction, b)
    weightMod.setMObjective(None,obj,0,sense=gp.GRB.MINIMIZE)

    #weightMod.Params.OutputFlag = 0 # tell gurobi to shut up!!
    weightMod.Params.TimeLimit = time # tell gurobi to shut up!!
    weightMod.optimize()

    eval_2019.append(weightMod.objVal)
    weights.append(weightMod_x.x[num:(num+N)])
    #print(weightMod_x.x[num:(num+N)])
    #print(weightMod_x.x[(num+N):(num+N+N)])

#print(eval_2019) #difference in return between portfolio and the index

# Initialize a data frame to record evaluation results
df2 = pd.DataFrame(columns = ['num_of_m', 'evaluation_by_2019'])
df2['num_of_m'] = m
df2['evaluation_by_2019'] = eval_2019
df2.to_csv(r'C:\Users\Elisha\1. Optimization\Project 2\eval_2019_new_method.csv',index = False)

weights = pd.DataFrame(weights)
weights.to_csv(r'C:\Users\Elisha\1. Optimization\Project 2\weights_new_method.csv',index = False)
```

```
Changed value of parameter TimeLimit to 3600.0
   Prev: inf  Min: 0.0  Max: inf  Default: inf
Gurobi Optimizer version 9.1.2 build v9.1.2rc0 (win64)
Thread count: 8 physical cores, 16 logical processors, using up to 16 threads
Optimize a model with 602 rows, 450 columns and 50708 nonzeros
Model fingerprint: 0xaf031104
Variable types: 350 continuous, 100 integer (100 binary)
Coefficient statistics:
  Matrix range     [2e-05, 2e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [3e-05, 5e+00]
Found heuristic solution: objective 1.9064653
Presolve time: 0.04s
Presolved: 602 rows, 450 columns, 50708 nonzeros
```

C.2 Evaluate portfolio using 2020 data

```python
eval_2020 = []

for i in range(len(m_all)):
    stock_idx = []
    weight_val = []
    count = 0
    for idx in range(N):
        if weights_new.to_numpy()[i][idx] > 0:
            stock_idx.append(count)
            weight_val.append(weights_new.to_numpy()[i][idx])
        count=count+1

    NDX_return_2020 = daily_return_2020['NDX'].tolist()
    stocks_return_2020 = daily_return_2020.iloc[:,[i + 1 for i in stock_idx]]

    A = stocks_return_2020.to_numpy()
    B = weight_val
    portfolio_return_2020 = pd.DataFrame(A*B).T.sum().tolist()

    sum = 0
    for j in range(len(portfolio_return_2020)):
        if NDX_return_2020[j] - portfolio_return_2020[j] > 0:
            sum += NDX_return_2020[j] - portfolio_return_2020[j]
        else:
            sum += portfolio_return_2020[j] - NDX_return_2020[j]
    eval_2020.append(sum)

print(eval_2020)  #difference in return between portfolio and the index
```

```
[0.7773624843660872, 0.6264951040219303, 0.5306697740724354, 0.47491114813551055, 0.40706120148804376, 0.41147957421833237, 0.381149004185
5099, 0.3668044658751281, 0.37060025329098006, 0.36868191551733176, 0.36868191551733176]
```

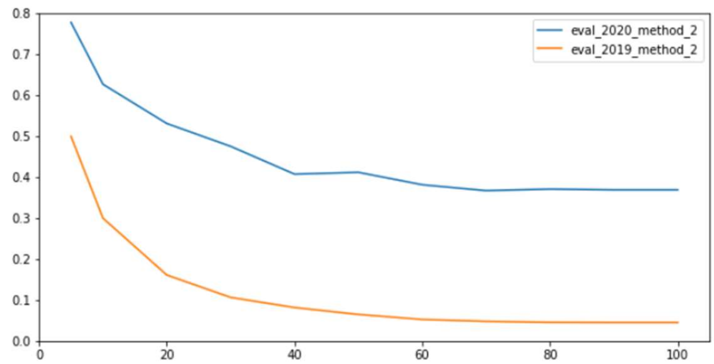
C.3 In-sample and out-of-sample performance

```
# pd.read_csv('eval_new_method.csv') for cross check
df3 = df1.copy()
df3['eval_2020_new_method'] = eval_2020
df3['eval_2019_new_method'] = pd.read_csv('eval_2019_new_method.csv')['evaluation_by_2019'].tolist()
df4 = df3.copy().drop(['evaluation_by_2020','evaluation_by_2019'], axis=1)

df4
```

num_of_m	eval_2020_new_method	eval_2019_new_method
5	0.777362	0.499259
10	0.626495	0.299563
20	0.530670	0.160711
30	0.474911	0.106094
40	0.407061	0.081338
50	0.411480	0.064503
60	0.381149	0.052260
70	0.366804	0.047588
80	0.370600	0.045227
90	0.368682	0.044911
100	0.368682	0.044911

```
plt.figure(figsize=(10,5))
plt.plot(df4)
plt.legend(['eval_2020_method_2', 'eval_2019_method_2'])
plt.xlim([0, 105])
plt.ylim([0, 0.8])
```

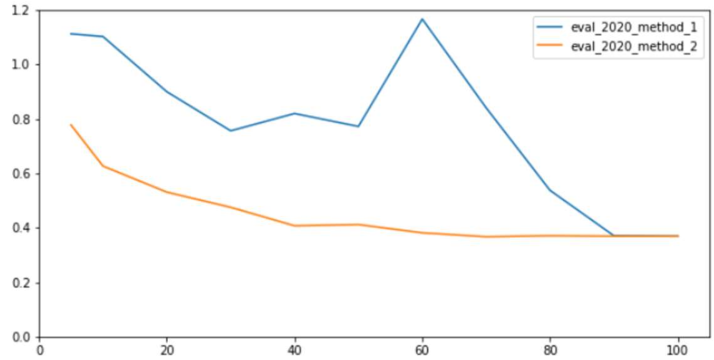
(0.0, 0.8)



C.4 Comparison between Method 1 and Method 2

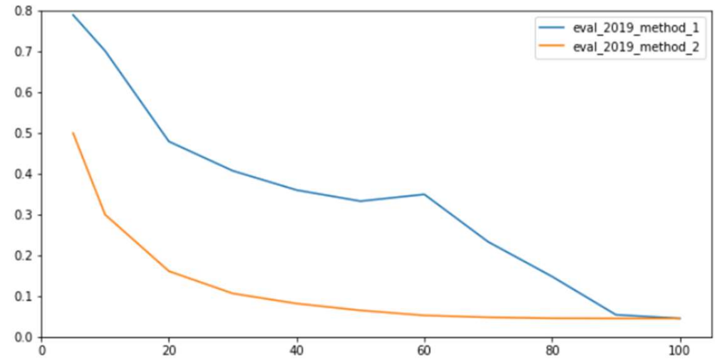
```
df5 = df3.copy().drop(['evaluation_by_2019','eval_2019_new_method'], axis=1)

plt.figure(figsize=(10,5))
plt.plot(df5)
plt.legend(['eval_2020_method_1', 'eval_2020_method_2'])
plt.xlim([0, 105])
plt.ylim([0, 1.2])
```

```
(0.0, 1.2)
```




```
df6 = df3.copy().drop(['evaluation_by_2020','eval_2020_new_method'], axis=1)

plt.figure(figsize=(10,5))
plt.plot(df6)
plt.legend(['eval_2019_method_1', 'eval_2019_method_2'])
plt.xlim([0, 105])
plt.ylim([0, 0.8])
```

```
(0.0, 0.8)
```



Recommendations

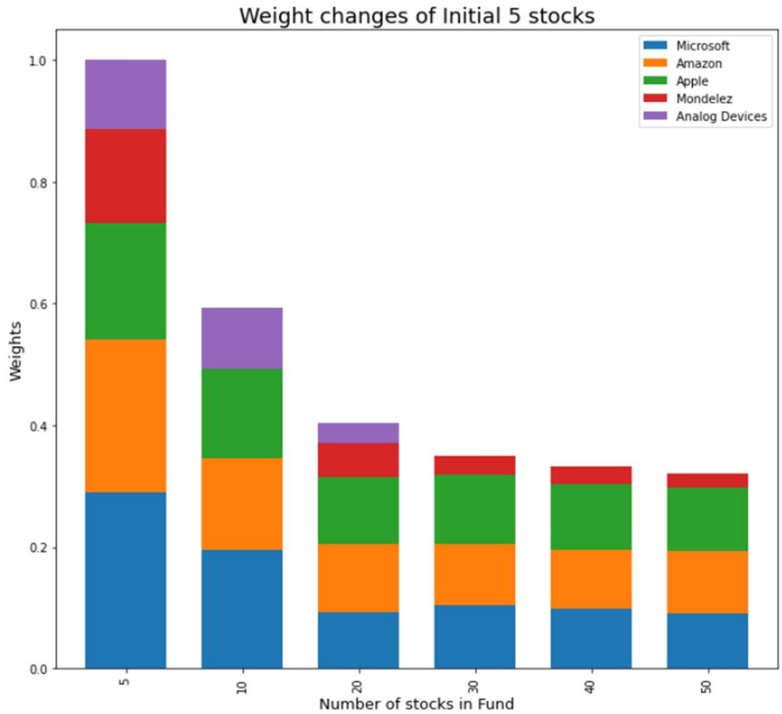
```
weights = pd.read_csv('weights_new_method.csv')
```

```
chosen_stocks = weights.apply(lambda row: row[row != 0].index, axis=1)[4]
chosen_stocks = list(chosen_stocks)

w = []
for i in weights.loc[4].to_list():
    if i > 0:
        w.append(i)

chosen_weights = pd.Series(w,name='Chosen Weights')
chosen_stocks = pd.Series(chosen_stocks,name='Chosen Stocks')
df = pd.concat([chosen_stocks,chosen_weights],axis=1)
df = df.sort_values(by=['Chosen Weights'], ascending=False)
df = df.reset_index()
df = df.drop(['index'], axis=1)
df['Chosen Weights'] = df['Chosen Weights']
df['Chosen Weights'] = df['Chosen Weights']
df['Chosen Weights'] = df['Chosen Weights'].apply('{:.2%}'.format)

df
```

	Chosen Stocks	Chosen Weights
0	AAPL	10.82%
1	MSFT	9.81%
2	AMZN	9.79%
3	GOOG	7.88%
4	FB	5.45%
5	INTC	3.30%
6	MDLZ	2.84%
7	CSCO	2.78%
8	ADBE	2.76%
9	TXN	2.73%
10	PAYX	2.65%
11	CTXS	2.12%

```
num_stocks = pd.Series(['5','10', '20','30', '40','50'], name='Number of stocks')
amz = pd.Series(weights['AMZN'][0:6],name='Amazon')
df = pd. concat([num_stocks, amz], axis=1)
ana = pd.Series(weights['ADI'][0:6],name='Analog Devices')
app = pd.Series(weights['AAPL'][0:6],name='Apple')
mic = pd.Series(weights['MSFT'][0:6],name='Microsoft')
mon = pd.Series(weights['MDLZ'][0:6],name='Mondelez')
df = pd.concat([num_stocks,mic,amz,app,mon,ana],axis=1)
df
df = df.set_index('Number of stocks')

df.plot.bar(stacked=True, figsize=(11,10), width = .7)
plt.xlabel('Number of stocks in Fund', fontsize = 13)
plt.ylabel('Weights', fontsize = 13)
plt.title('Weight changes of Initial 5 stocks', fontsize = 18)
```

```
Text(0.5, 1.0, 'Weight changes of Initial 5 stocks')
```




```
x = [5,10,20,30,40,50,60,70]
plt.figure(figsize=(15,10))
plt.plot(x,weights['AMZN'][0:8], label = 'Amazon')
plt.plot(x,weights['ADI'][0:8], label = 'Analog Devices')
plt.plot(x,weights['AAPL'][0:8], label = 'Apple')
plt.plot(x,weights['MSFT'][0:8], label = 'Microsoft')
plt.plot(x,weights['MDLZ'][0:8], label = 'Mondelez')
plt.plot(x,weights['GOOGL'][0:8], label = 'Alphabet Class A')
plt.plot(x,weights['CHTR'][0:8], label = 'Charter Communications')
plt.plot(x,weights['CSCO'][0:8], label = 'Cisco Systems')
plt.plot(x,weights['GILD'][0:8], label = 'Gilead Sciences')
plt.plot(x,weights['NFLX'][0:8], label = 'Netflix')
plt.xlabel('Number of stocks included in Fund', fontsize = 14)
plt.ylabel('Weights', fontsize = 14)
plt.title('Weight changes of Initial 10 stocks', fontsize = 18)
plt.legend()
```

```
<matplotlib.legend.Legend at 0x14e9a3fce80>
```