Optimization Project 2
Stochastic Control and Optimization – Integer Programming

Charan Musunuru(cm59982)
Harsh Mehta (hdm564)
Karthick Ramasubramanian (kr33733)
Yashaswini Kalva (yk8348)

Problem Statement

The objective is to construct an index fund to track the returns of the NASDAQ-100. We build our portfolio by using returns of the NASDAQ-100 constituents and the overall index returns for the year of 2019. Our algorithm identifies the best combination of stocks along with their weights in the portfolio to minimize the difference of returns between the index and the portfolio. We implement this optimal tracking portfolio using several different approaches including Linear Programming and Mixed Integer Programming and compare the results from these approaches on returns for the year 2020 (test set) before making a final recommendation.

Process Summary

1. Stock Selection: Formulate a linear program that picks m out of 100 NASDAQ-100 stocks for the portfolio. Input for this program is a similarity matrix, with individual elements of matrix ρij, which represents correlation between stocks i and j.

2. Portfolio Weights Calculation: To solve another linear program and decide number(weights) of chosen stocks to buy for the portfolio

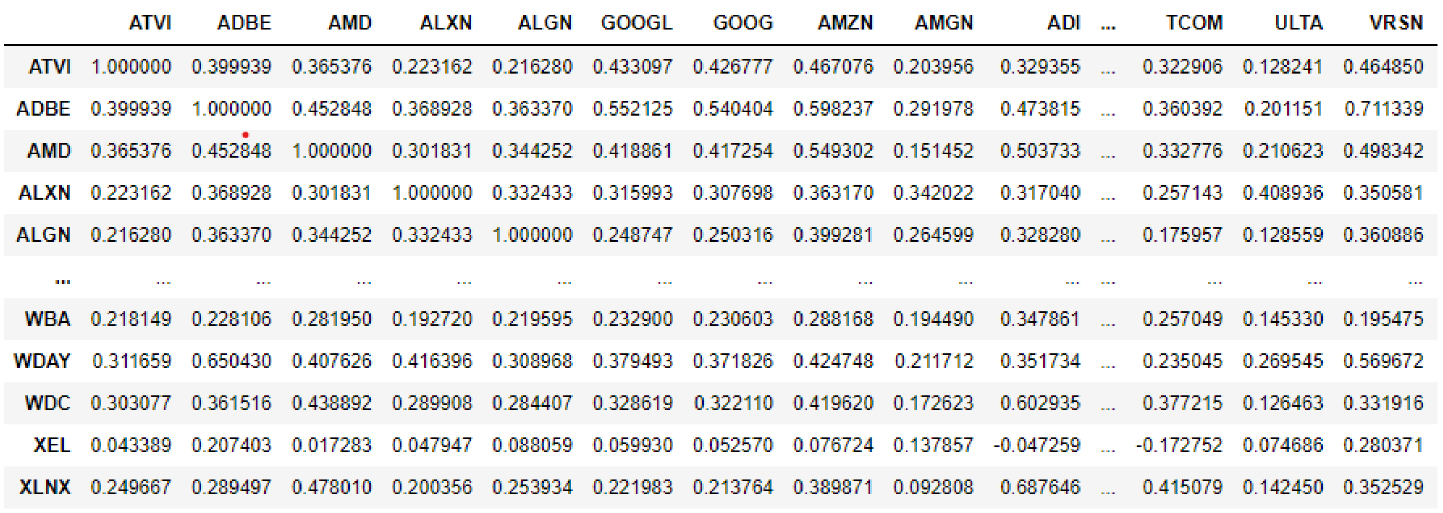
3. Performance Evaluation: To evaluate performance of index funds compared to NASDAQ-100 index. And to compare performance for different numbers of stocks chosen in the portfolio.

4. Two Approaches: Linear Programming(LP) & Mixed Integer Programming (MIP)

Stock Selection

The correlation matrix was used as a measure of similarity between stocks to find the optimal tracking portfolio.

Correlation Matrix



	ATVI	ADBE	AMD	ALXN	ALGN	GOOGL	GOOG	AMZN	AMGN	ADI	...	TCOM	ULTA	VRSN
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
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
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
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
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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
WBA	0.218149	0.228106	0.281950	0.192720	0.219595	0.232900	0.230603	0.288168	0.194490	0.347861	...	0.257049	0.145330	0.195475
WDAY	0.311659	0.650430	0.407626	0.416396	0.308968	0.379493	0.371826	0.424748	0.211712	0.351734	...	0.235045	0.269545	0.569672
WDC	0.303077	0.361516	0.438892	0.289908	0.284407	0.328619	0.322110	0.419620	0.172623	0.602935	...	0.377215	0.126463	0.331916
XEL	0.043389	0.207403	0.017283	0.047947	0.088059	0.059930	0.052570	0.076724	0.137857	-0.047259	...	-0.172752	0.074686	0.280371
XLNX	0.249667	0.289497	0.478010	0.200356	0.253934	0.221983	0.213764	0.389871	0.092808	0.687646	...	0.415079	0.142450	0.352529

Decision Variables:

- yj - stocks j from the index are present in the fund
- xij - stock j in the index is the best representative of stock i
- ρij -correlation between stock's i and j

Constraints and Objective function for Stock Selection:

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

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

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

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

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

Preparation of the problem:

1.Exactly m stocks are picked.

2. Each stock is represented by only one other stock in the tracking portfolio.

3. The representative for each stock must be present in the index fund.

The objective function maximizes the sum of correlations (of returns) of the selected stocks.

```
#Creating the constraint matrix, objective function and bvec for optimization
A = np.zeros((n*n+1+n,n*n+n))
b = np.zeros(n*n+1+n)

#\sum \rho_{ij} x_{ij} is the objective function
obj = np.append(np.array([0]*n),np.array(np.matrix(cor).flatten(order='F')).flatten())

# Initialize the Amat matrix, the bVec.
# There are nStocks^2 + nStocks + 1 constraints while nStocks + nStocks^2 variables.

# The constraint \sum_{j=1}^n y_j = q
A[0,0:n] = [1]*n

# The constraints \sum_{j = 1}^n x_{ij} = 1 for i = 1, ..., n
for i in range(0,n):
    for j in range(1,n+1):
        A[i+1,j*n + i] = 1

# The constraints x_{ij} <= y_{j} for i = 1, ..., n; j = 1, ..., n
for j in range(1,n+1):
    A[(1 + n*j):(1 + n*(j+1)), j-1] = -1
    np.fill_diagonal(A[(1 + n*j):(1 + n*(j+1)), (n*j):(n*(j+1))], 1)

b[1:n+1] = [1]*n
b[n+1:n+n*n+1] = 0
sense = np.array(['=']*(n+1)+['<']*(n*n))
```

Running the optimization and selecting the stocks for the portfolio

```
def choose_stocks(m,b,A,sense,n,obj,plot_check):
    b[0] = m
    ojModel = gp.Model() # initialize an empty model

    ojModX = ojModel.addMVar(n*n+n, vtype=gp.GRB.BINARY) # tell the model how many variables there are
    # must define the variables before adding constraints because variables go into the constraints
    ojModCon = ojModel.addMConstrs(A, ojModX, sense, b) # add the constraints to the model
    ojModel.setMObjective(None,obj,0,sense=gp.GRB.MAXIMIZE) # add the objective to the model

    ojModel.Params.OutputFlag = 0 #

    ojModel.optimize() # solve the LP
    sol = ojModX.x[0:n]
    stocks = ['NDX']
    for i in range(0, n):
        if(sol[i] == 1.0):
            stocks.append(returns.columns[i+1])
    weights, error_2019 = find_w(returns[stocks])

    port_returns = returns_2020[stocks].iloc[:,1:].mul(weights,axis=1).sum(axis=1)+1
    index_returns = returns_2020.iloc[:,0]+1
    index_returns_2019 = returns.iloc[:,0]+1
    port_returns_2019 = returns[stocks].iloc[:,1:].mul(weights,axis=1).sum(axis=1)+1
```

Portfolio Weights:

To get the portfolio weights we need to match the returns of the index as closely as possible. We need to minimize the sum of absolute differences between the return of index and the portfolio to match the index returns closely daily

Objective Function: To minimize the below equation

$$\sum_{t=1}^{T} |q_{\mathrm{t}} - \sum_{i=1}^{m} w_{\mathrm{i}} r_{\mathrm{it}}|$$

r_{it} → return of stock i at time period t,

Q_t → return of the index at time t

w_i → weight of stock i in the portfolio

Portfolio Weight Constraints:

1. Converting Non-linear to Linear the two constraints are

$$z_{\mathrm{i}} \geq -(\textstyle\sum_{t=1}^{T} q_{\mathrm{t}} - \sum_{i=1}^{m} w_{\mathrm{i}} r_{\mathrm{it}})$$

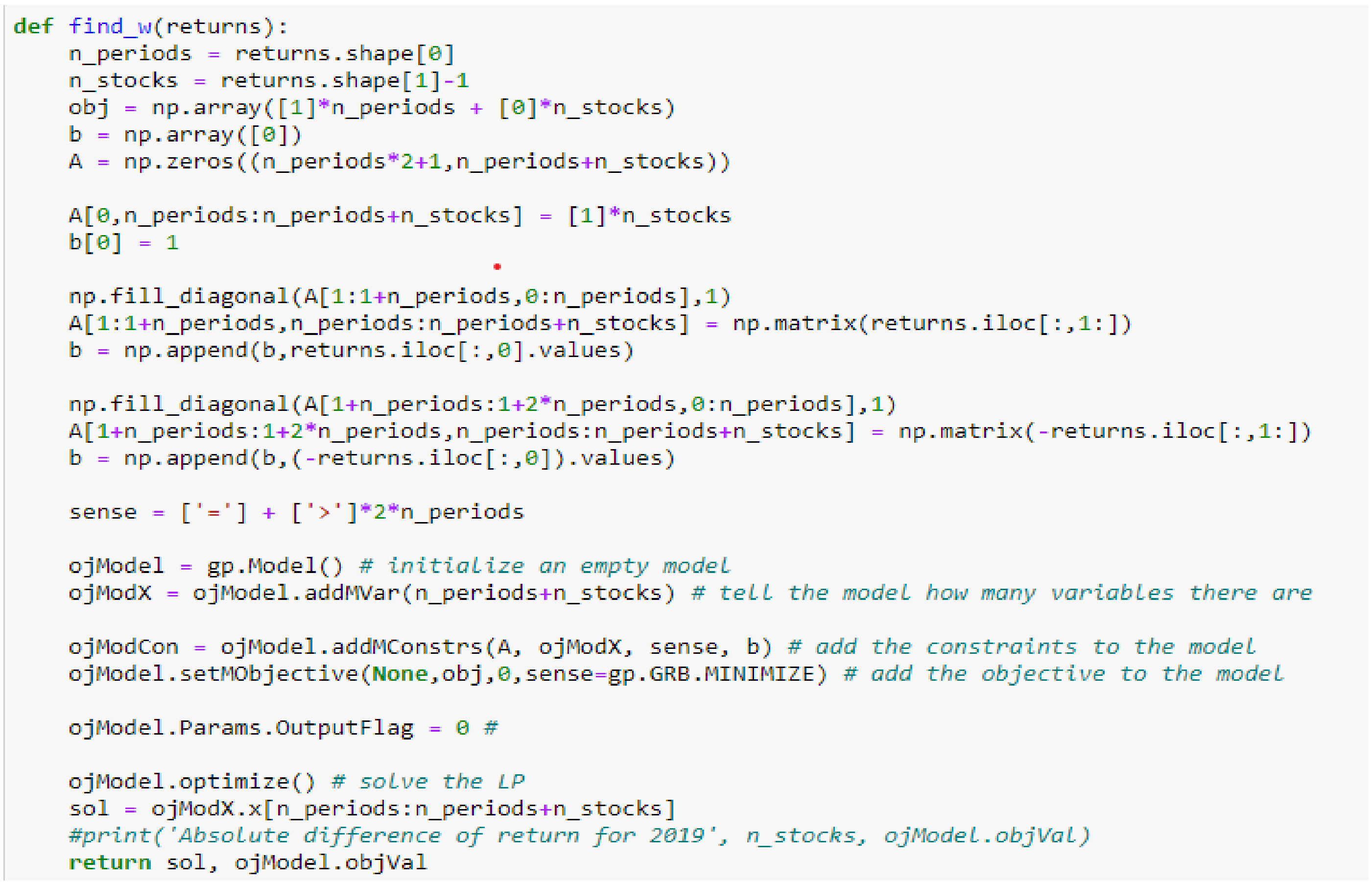
$$z_{\mathrm{i}} \geq \textstyle\sum_{t=1}^{T} q_{\mathrm{t}} - \sum_{i=1}^{m} w_{\mathrm{i}} r_{\mathrm{it}}$$

2. Sum of weights of all stocks in fund adds up to 1

$$\sum_{i=1}^{m} w_{\mathrm{i}} = 1$$

The objective function minimizes the absolute difference between the return of the stock times its respective weight, and the return of the index at a certain time t.

Code snippet for calculating portfolio weights using Linear programming.



```python
def find_w(returns):
    n_periods = returns.shape[0]
    n_stocks = returns.shape[1]-1
    obj = np.array([1]*n_periods + [0]*n_stocks)
    b = np.array([0])
    A = np.zeros((n_periods*2+1,n_periods+n_stocks))

    A[0,n_periods:n_periods+n_stocks] = [1]*n_stocks
    b[0] = 1

    np.fill_diagonal(A[1:1+n_periods,0:n_periods],1)
    A[1:1+n_periods,n_periods:n_periods+n_stocks] = np.matrix(returns.iloc[:,1:])
    b = np.append(b,returns.iloc[:,0].values)

    np.fill_diagonal(A[1+n_periods:1+2*n_periods,0:n_periods],1)
    A[1+n_periods:1+2*n_periods,n_periods:n_periods+n_stocks] = np.matrix(-returns.iloc[:,1:])
    b = np.append(b,(-returns.iloc[:,0]).values)

    sense = ['='] + ['>']*2*n_periods

    ojModel = gp.Model() # initialize an empty model
    ojModX = ojModel.addMVar(n_periods+n_stocks) # tell the model how many variables there are

    ojModCon = ojModel.addMConstrs(A, ojModX, sense, b) # add the constraints to the model
    ojModel.setMObjective(None,obj,0,sense=gp.GRB.MINIMIZE) # add the objective to the model

    ojModel.Params.OutputFlag = 0 #

    ojModel.optimize() # solve the LP
    sol = ojModX.x[n_periods:n_periods+n_stocks]
    #print('Absolute difference of return for 2019', n_stocks, ojModel.objVal)
    return sol, ojModel.objVal
```

Performance evaluation:

We are using the sum of the absolute differences of daily returns of the Portfolio and the Index to evaluate the performance of our portfolio.

Method 1 - Index vs Portfolio Returns Stock - Correlation Method.

Incrementing the total number of stocks in our Portfolio, from 10 to 100 in steps of 10, we observed the performance of the portfolio in the year 2019 and 2020 based on the sum of the absolute difference of the returns between the index and the fund.

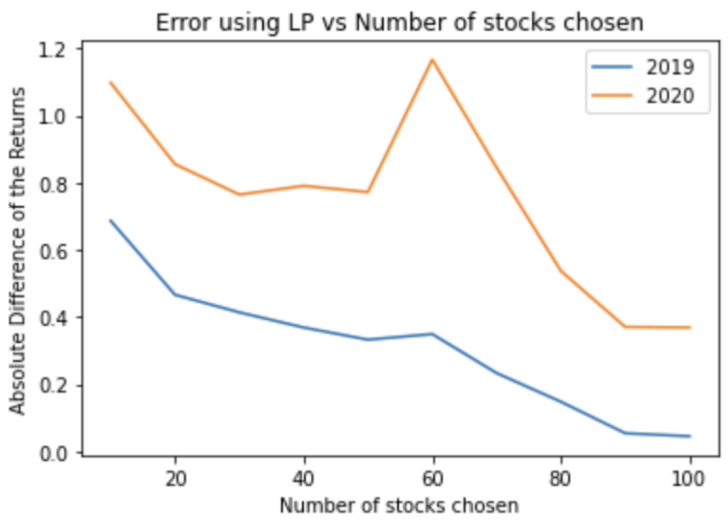
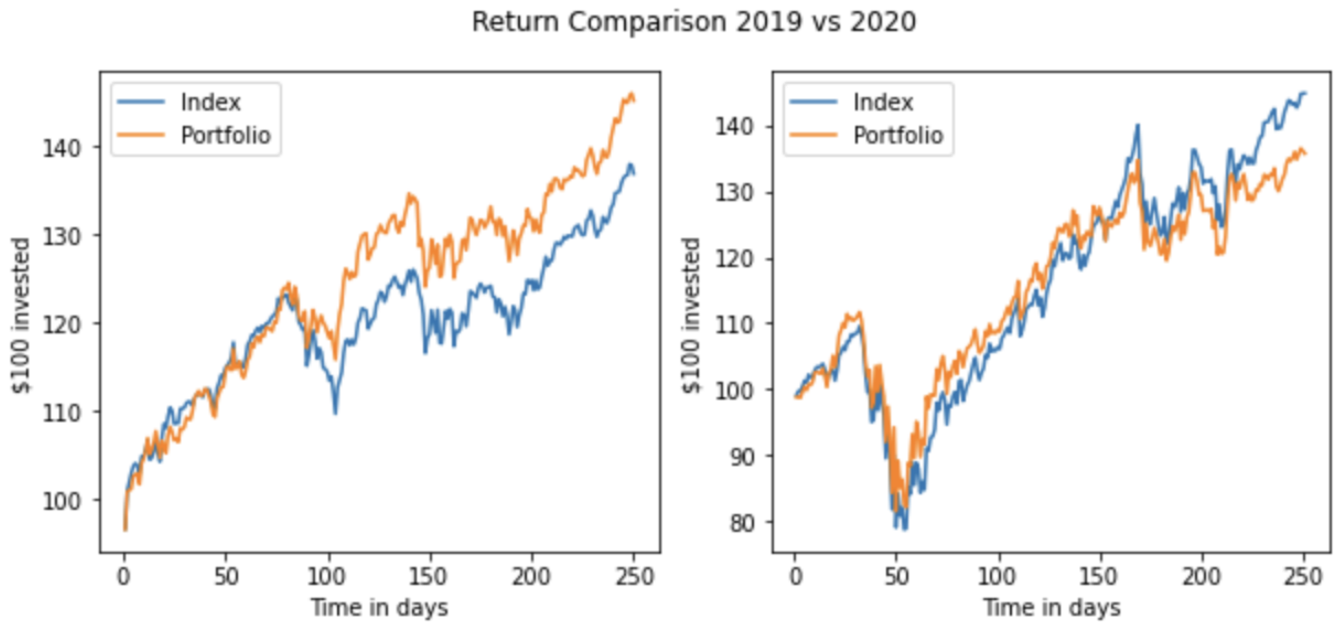


Figure 1 : Return Error - Correlation Method

For 2019, the trend is continuously decreasing. There is a large decrease in difference of return when moving from 10 to 20 and then there is an almost consistent decrease as the number of stocks are increasing. On the other hand, we see that the 2020 return although a decreasing trend is not very smooth and has a spike at 60 stocks. Assuming this spike is due to idiosyncratic events happening with a few stocks, the ideal number of stocks should be somewhere near 40 or near 80 as that is where the most performance gains. The best position would depend on other factors like rebalancing costs and market spreads.

For M = 5 (5 stocks chosen to be in the portfolio)



```
In sample error (Using 2019 Returns) 0.7891782824631473
Out of sample error (Using 2020 Returns) 1.1124373455076464
```

Figure 2 : Cumulative Returns for Portfolio of 5 Stocks vs Index using Stock Correlation Method for 2019 and 2020.

Method 2 - Index vs Portfolio Returns Stock - Multiple Integer Programming (Direct Weight Optimization)

We made use of an alternate approach i.e., using mixed integer programming.

We evaluated another approach where we completely ignored the stock selection IP and reformulated the weight selection problem to be an MIP that constrains the number of non-zero weights to be an integer. To do this take the weight selection problem and replace m with n so that you are optimizing over ALL weights:

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

In this method, we define slack variables yi's which is the difference of the returns between the Portfolio and the Index. The objective is to keep the slack as low as possible. Since the number of stocks are pre-defined, we introduce another variable, yi, one for each stock, which is a binary variable whose value is 1 when weights can be non-zero. The sum of non-zero y is the total number of stocks in our portfolio.

Lets see how the model is able to capture the index for m=10. The fund is able to track the index for the year 2019 and 2020 is depicted in the below figures.

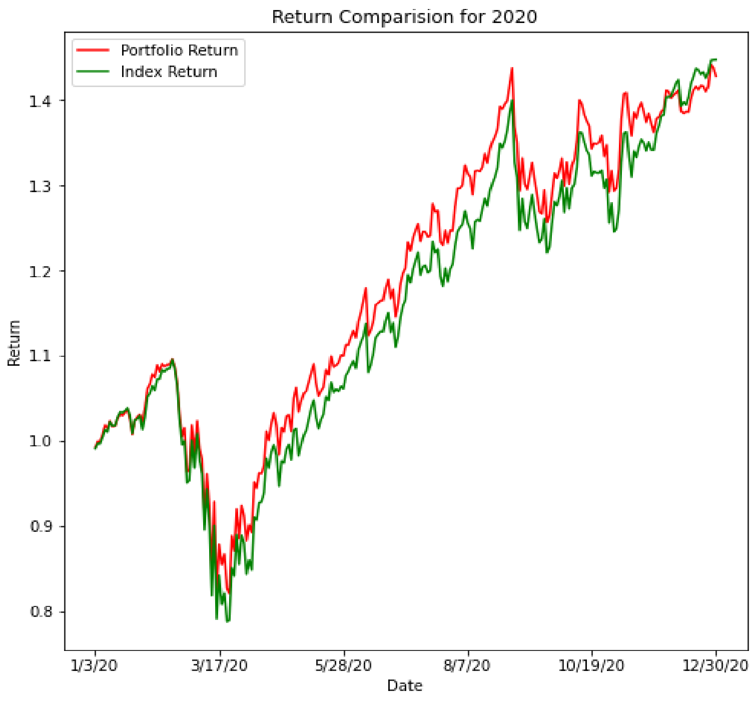


Figure 3 : Cumulative Returns for Portfolio of 10 Stocks vs Index using Mixed Integer programming

The graph shown below shows the absolute difference of the Returns(error) for 2019 and 2020 using the mixed integer programming method. The x-axis shows the number of stocks chosen and y-axis denotes the error. As seen from the graph below, we obtain an optimal solution at m = 60 stocks.

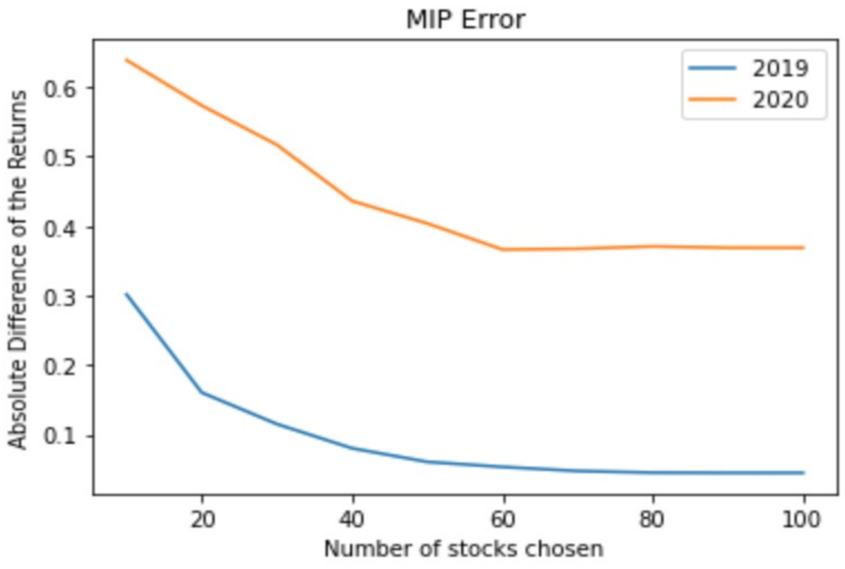


Figure 4 : Return Error - Multiple Integer Programming

Recommendations

- Based on the overall analysis, we recommend the second method (Multiple integer programming) to pick the most optimal number of component stocks in our fund from the index fund and its weights going forward.

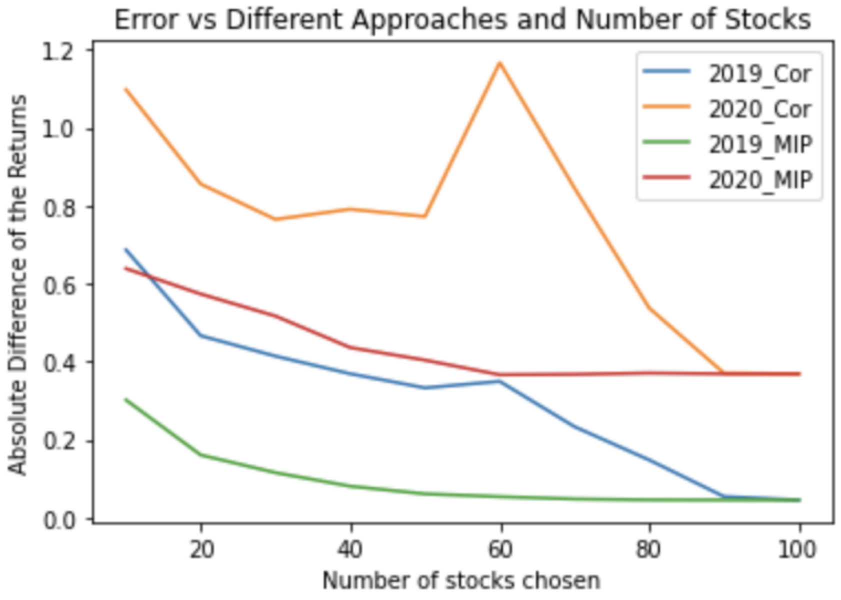
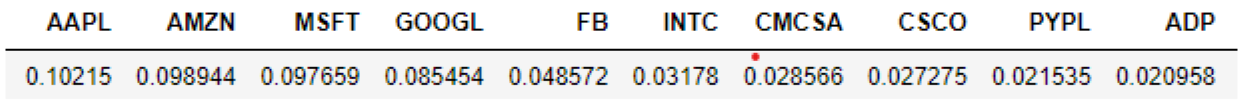


Figure 5 : Return Error - Comparison

- This is because the Direct weight optimization (mixed integer programming) performs better by giving us a lower tracking error when compared to the first method as seen in the graph (figure 5) above. Since, computation power is not a big issue in the industry, this would be ideal. We can even further decrease the error by letting the optimizer run longer.
- Also when you compare figure 2 and figure 3, the fit for the portfolio returns with the index returns for the mixed integer programming method seems much better than the correlation method as described in the report

- The optimal number of component stocks in our fund to track 2020 returns are **60** as the errors after 60 component stocks seem to flatten out. The corresponding weights for the top 10 stocks ordered by weights is given below.



AAPL	AMZN	MSFT	GOOGL	FB	INTC	CMCSA	CSCO	PYPL	ADP
0.10215	0.098944	0.097659	0.085454	0.048572	0.03178	0.028566	0.027275	0.021535	0.020958