Advanced Investment Science – Final Report

20161342 Keonwoo Kim

1. Introduction

According to the Table 1 summarizing the pension fund's 2016 to 2018 returns, South Korea's National Pension Fund had lower returns to national pensions in advanced countries such as United States, Japan, Norway, and Canada. Also, in the first quarter of 2020, when COVID-19 arrived, the loss of the National Pension Fund was as large as the 2008 financial crisis. The loss of National Pension Fund was large due to a large loss occurred in stocks. So in this project, I'll set South Korea's National Pension Fund as a client, an institutional investor. Assuming that we were return to 2020.01.01, we will discuss about how to minimize the loss of National Pension Fund by setting up a strategy at that time.

[해외연기금의 수익률 현황]

(단위: %)

		GPIF (일본)	GPFG (노르웨이)	국민연금 (한국)	ABP (네덜란드)	CalPERS (미국)	CPP (캐나다)
연도별 수익률	2016	5.9	6.9	4.7	9.5	0.6	11.8
	2017	6.9	13.7	7.3	7.6	11.2	11.5
	2018	1.5	−6.1	−0.9	−2.3	8.6	9.0
5년 평균 수익률		4.4	4.7	4.2	6.2	8.1	10.7
10년 평균 수익률		5.0	8.3	5.5	8.7	5.7	11.3

Table 1

2. Financial Planning

Planning period was set to 2020.01.02 to 2020.12.31 which is a chaotic time when COVID-19 was the most serious. And the rebalancing periods was set every quarter which is April 1, July 1 and October 1, also we should also consider first investment on January 1. The investment is based on the historical data from 2018~2019, and every time a rebalance is made, new historical data from that period will also be taken into account. There are five main assumptions in financial planning. First of all, portfolio manager and investors are now on 2020.01.01. So stock prices since 2020.01.01 are completely unknown. Second of all, the National Pension Fund's investment is a considerable amount of 700 trillion Won, but to put it more concisely, the project uses the investment at 1 million won. Third of all, The National Pension Funds actually invests in various products including real estate, bonds, stocks and etc. But in this project, we assume to invest only in bonds and stocks because there is a limit to considering all products. Moreover, considering all domestic and foreign bonds and stocks is practically difficult to handle. Only six domestic stocks and six domestic bonds were used. Lastly, both stocks and bonds are designated as ETF related products. In addition to the five assumptions mentioned, there are some minor assumptions in this project, and they will be treated later.

I used robust portfolio selection method to optimize National Pension Fund's portfolio. This is because results of robust optimization have characteristics that are insensitive to small deviations. Also, box uncertainty set was chosen while solving out robust optimization. Thus, I can optimize optimal portfolio weights under considering worst case. Given the nature of a client's conservative investment orientation, setting the financial plans with considering worst case would be appropriate on investment strategy on COVID-19 time.

I made a strategy by considering the share of stocks and bonds in the actual Korea's National Pension Fund's usual portfolio. Due to stable investment tendency of the National Pension Fund, the portion of domestic bonds was 1.7 times more than domestic stocks, describe on Table 2 and Table 3. So I will insert these properties into the optimization constraint.

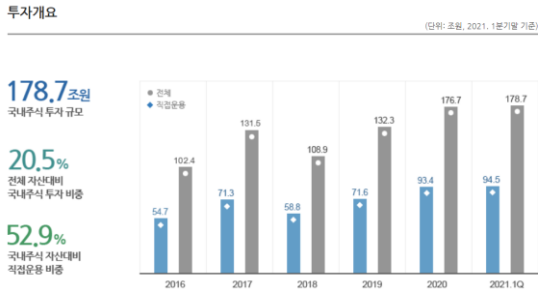


Table 2

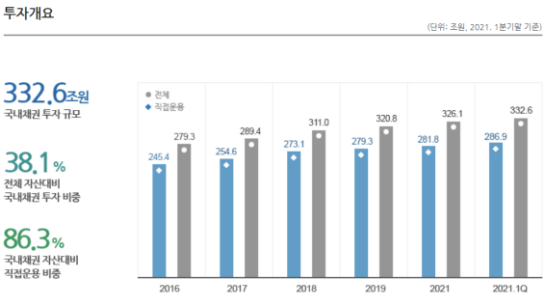


Table 3

To make sure our suggested strategy is okay, I plan to run 1000 different simulations and show them to a client. First, I show the yield of 1000 samples using a multi-variate normal distribution. And optimize each sample quarterly. Every time we optimize here, we update the expected return quarterly. This is the simulation result for one sample. I will repeat this process a total of 1000 times, and save the degree of the final asset for each turn, and visualize it like a distribution.

Fig 1 explains the overview of simulation investment. First, we calculate estimate parameters based on 2018 to 2019 historical data. Then, I solve out weights by using robust optimization. That would be a first investment in 2020.01.01. With buy-and-hold method, we assume to keep our money to stocks and bonds that period, and sell all of them on the last day of first quarter which means changing stocks and bonds into cash. Second, we solve out new estimate parameters based on first quarter history data and calculate new weights by robust optimization. Then, reinvesting stocks and bonds with money taken into account from past investments to returns. The important assumption in here is that, the period which is considered while calculating new estimators, I do not only consider first quarter because it has small number of data. I used 'moving window' method to keep same length of period to 2 years in order to calculate parameters. This same process is done for each quarter, and it would be on iteration of simulation. So 1000 times are done to make divers kinds of simulations.

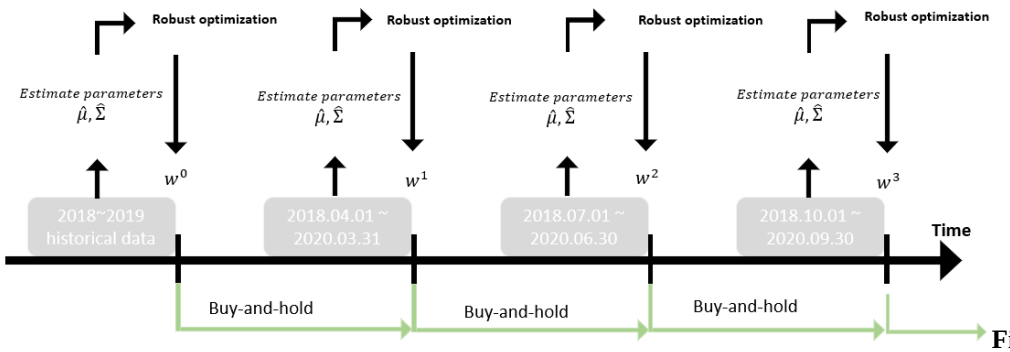


Fig 1

Table 4 explains the average expected return and standard deviation for each rebalanced period. Also, I calculated overall average of samples' total remaining money as $10496 at the end of 2002, and distribution of it which is illustrated on Fig 2. It is shown that the proportion of distribution in

10000~11000$ section is the largest, it is highly likely that the amount of money in this section will come out as the expected amount of money. And with these simulation results, I will suggest to Korean Pension Fund to invest money on stocks and bonds with following strategy.

Rebalanced period		
First (01.01~03.31)	Average expected return	5.79%
	Average expected std	3.62%
Second (04.01~06.30)	Average expected return	-0.04%
	Average expected std	1.72%
Third (07.01~9.30)	Average expected return	-0.26%
	Average expected std	1.72%
Fourth (10.01~12.31)	Average expected return	-0.02
	Average expected std	1.72%

Table 4

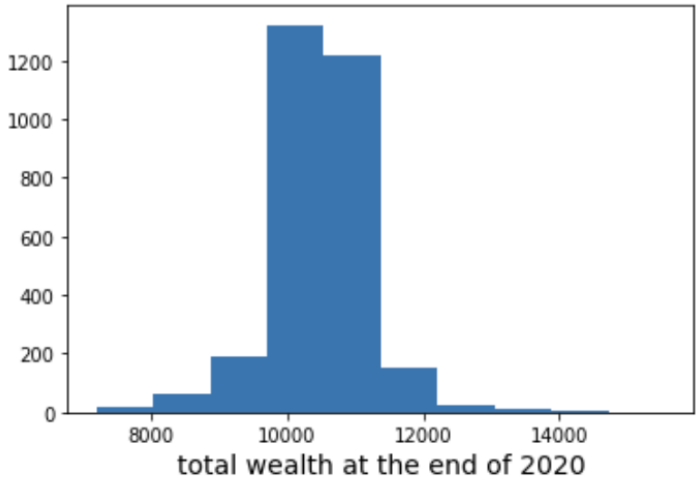


Fig 2

3. Progress Report

Progress report is made halfway through the total investment planning period, which is on the end of second quarter in 2020, illustrated on Fig 3.

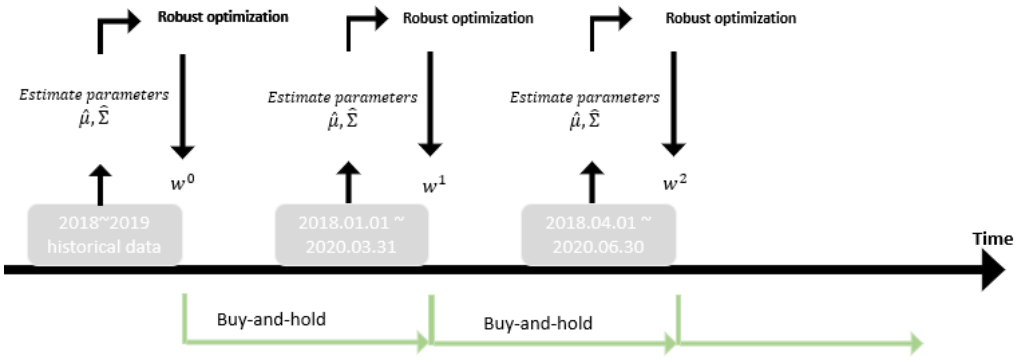


Fig 3

And Table 5 and Table 6 show the actual result on investment with suggested strategy in first quarter and second quarter. At the end of first quarter, the movement of total wealth is from $10000 to $9528. For second quarter, total wealth increase from $9528 to $9978. And on this progress report, I calculated the portfolio evaluation, conducted by the time weighted rate of return, on this portfolio. This

geometric mean return is calculated from January 1 to June 30, and the monthly interval was not set but quarter was set as one period in calculation. The result of it is about -1.13% which shows poor performance.

Type	Period: 2020.01.01~2020.03.31	Weight
Bond	KODEX 10Y F-LKTB	0.000
	KODEX Treasury Bond 3Y	0.091
	KODEX KRW Cash	0.311
	KOSEF Enhance Cash	-0.050
	TIGER Money Market	-0.049
	KOSEF 10Y KTB	0.365
Stock	KODEX 200	0.533
	KODEX KOSDAQ 150	0.000
	KODEX LEVERAGE	-0.049
	KODEX KOSDAQ LEVERAGE 150	-0.049
	KODEX SAMSUNG	-0.050
	TIGER TOP 10	-0.049
	Total bond Weight / Total stock weight	66.7% / 33.3%
	Expected Return / Standard Deviation	1.77% / 0.03%

Table 5

Type	Period: 2020.04.01~2020.06.30	Weight
Bond	KODEX 10Y F-LKTB	0.000
	KODEX Treasury Bond 3Y	-0.05
	KODEX KRW Cash	0.677
	KOSEF Enhance Cash	-0.049
	TIGER Money Market	-0.05
	KOSEF 10Y KTB	0.140
Stock	KODEX 200	0.533
	KODEX KOSDAQ 150	0.000
	KODEX LEVERAGE	-0.050
	KODEX KOSDAQ LEVERAGE 150	-0.049
	KODEX SAMSUNG	-0.050
	TIGER TOP 10	-0.049
	Total bond Weight / Total stock weight	66.7% / 33.3%
	Expected Return / Standard Deviation	2.16% / 0.03%

Table 6

To achieve the goals that were initially set, the strategy was revised. I revised its plan to increase the portion of stocks and reduce the portion of bonds, depicted on equation (1), as it has to achieve its goal even if it sits on higher risk.

$$W_{Stock} : W_{bond} = 1 : 1.2 \qquad (1)$$

After changing strategy, in the way increasing the portion of stocks, it seems to show have total wealth amount as $10689 at the end of 2020. Thus, due to better results in Table 7 and stable distribution in Fig 4.new suggested strategy seems to be attractive one to a client.

Rebalanced period		
First (07.01~09.30)	Average expected return	9.67%
	Average expected std	4.82%
Second (10.01~12.31)	Average expected return	-0.28%
	Average expected std	2.44%

Table 7

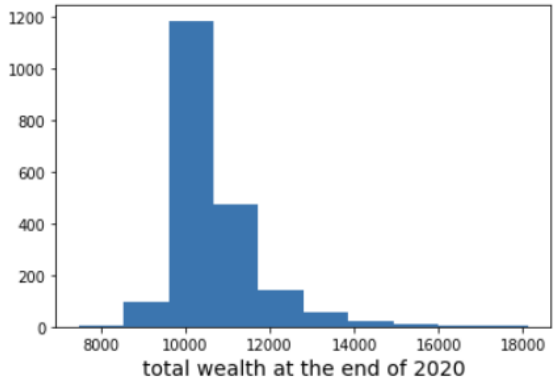


Fig 4

4. Additional Works

The strategy first proposed on a client on 2020.01.01 was evaluated only through June 30 and a progress repot was written. However, assuming that it was carried out by December 31, I evaluate the portfolio yield throughout the year in this paragraph. With this strategy, if I rebalance and optimize a client's portfolio quarterly from the beginning of the yare to the end of the year, $10391 remains ate the end of the year. In addition, the value of portfolio evaluation by the time weighted rate of return is -0.02%. So it was a better decision to change strategy in 'Progress Report'.

5. Conclusion

To sum up, although I tried portfolio optimization with robust optimization method, the last result shows bad performance due to COVID-19 in 2020. The financial crisis of countries caused by global disaster, COVID-19, could not be overcome by simple optimization strategy. Next time, I think better results would come out if we draw up a new optimization strategy linked to Artificial Intelligence and work on more concrete and realization of what we assumed in this project.

6. Python Code Explanation

```
import pandas as pd
import numpy as np
import pandas_datareader.data as web
import matplotlib.pyplot as plt
from tqdm import tqdm
import cvxpy as cp
import warnings
warnings.filterwarnings(action='ignore')
```

I used several libraries to facilitate the project. Pandas, Numpy and tqdm are used for data handling, matplotlib.pyplot is used for visualization, and pandas_datareader.data is used for loading historical data on each stock and bond. Also, cvxpy is used in optimization process.

I made several modules that are often used in project, making code easier to understand and work with. Now explanation for several modules, and codes with utilizing those modules will be continued on next page.

```
def price_data(start,end):
    #Korea stock
    KODEX_200 = web.DataReader('069500.KS',data_source='yahoo',start=start,end=end)
    KOSDAQ_150 = web.DataReader('229200.KS',data_source='yahoo',start=start,end=end)
    KODEX_LVERAGE = web.DataReader('122630.KS',data_source='yahoo',start=start,end=end)
    KODEX_KOSDAQ_150_LVERAGE = web.DataReader('233740.KS',data_source='yahoo',start=start,end=end)
    KODEX_SAMSUNG = web.DataReader('102780.KS',data_source='yahoo',start=start,end=end)
    TIGER_TOP10 = web.DataReader('292150.KS',data_source='yahoo',start=start,end=end)


    #Korea bond
    KODEX_10Y = web.DataReader('152380.KS',data_source='yahoo',start=start,end=end)
    KODEX_TBS_3Y = web.DataReader('114260.KS',data_source='yahoo',start=start,end=end)
    KODEX_KRW_CASH = web.DataReader('153130.KS',data_source='yahoo',start=start,end=end)
    KODEX_KRW_CASH_PLUS = web.DataReader('130730.KS',data_source='yahoo',start=start,end=end)  # kosef-130730
    TIGER_MONEY_MARKET = web.DataReader('157450.KS',data_source='yahoo',start=start,end=end)
    KIWOOM_10Y = web.DataReader('148070.KS',data_source='yahoo',start=start,end=end)

    #price_data
    close_df = pd.DataFrame({'bond_1':KODEX_10Y['Adj Close'],
                             'bond_2':KODEX_TBS_3Y['Adj Close'],
                             'bond_3':KODEX_KRW_CASH['Adj Close'],
                             'bond_4':KODEX_KRW_CASH_PLUS['Adj Close'],
                             'bond_5':TIGER_MONEY_MARKET['Adj Close'],
                             'bond_6':KIWOOM_10Y['Adj Close'],
                             'stock_1':KODEX_200['Adj Close'],
                             'stock_2':KOSDAQ_150['Adj Close'],
                             'stock_3':KODEX_SAMSUNG['Adj Close'],
                             'stock_4':KODEX_KOSDAQ_150_LVERAGE['Adj Close'],
                             'stock_5':KODEX_LVERAGE['Adj Close'],
                             'stock_6':TIGER_TOP10['Adj Close']})

    close_df.drop(index=close_df.iloc[:1, :].index.tolist(), inplace=True)
    return close_df
```

This module, price_data, is responsible for loading historical data from stocks and bonds and making them into a data frame. And parameters, ‘start’ and ‘end’, refer to the start and end of the period in which the data is retrieved. I used pandas_datareader.data while loading a historical data.

```
def calculate_gain(investment_amount, weight, close_df):

    portfolio_amount = []
    for w in weight:
        portfolio_amount.append(investment_amount*w)

    gain_list = []
    for i in range(12):
        gap_price = close_df.iloc[-1][i] - close_df.iloc[0][i]
        rtn_pct = gap_price / close_df.iloc[0][i]
        gain_list.append((1+rtn_pct)*portfolio_amount[i])
    all_gain = 0
    for rtn in gain_list:
        all_gain += rtn

    return all_gain
```

This module, calculate_gian, is responsible for calculating total wealth. There are three parameters on this module which are ‘investement_amount’, ‘weight’ and ‘close_df’. Investment_amount just refers to amount of investment. Close_df is a price of stock and bonds in data frame type which is made from a module ‘price_data’ and weight is portions in each product from a optimization result.

20161342 Keonwoo Kim

```
def calculate_mu_sigma(start,end):
    #Korea stock
    KODEX_200 = web.DataReader('069500.KS',data_source='yahoo',start=start,end=end)
    KOSDAQ_150 = web.DataReader('229200.KS',data_source='yahoo',start=start,end=end)
    KODEX_LVERAGE = web.DataReader('122630.KS',data_source='yahoo',start=start,end=end)
    KODEX_KOSDAQ_150_LVERAGE = web.DataReader('233740.KS',data_source='yahoo',start=start,end=end)
    KODEX_SAMSUNG = web.DataReader('102780.KS',data_source='yahoo',start=start,end=end)
    TIGER_TOP10 = web.DataReader('292150.KS',data_source='yahoo',start=start,end=end)

    #Korea bond
    KODEX_10Y = web.DataReader('152380.KS',data_source='yahoo',start=start,end=end)
    KODEX_TBS_3Y = web.DataReader('114260.KS',data_source='yahoo',start=start,end=end)
    KODEX_KRW_CASH = web.DataReader('153130.KS',data_source='yahoo',start=start,end=end)
    KODEX_KRW_CASH_PLUS = web.DataReader('130730.KS',data_source='yahoo',start=start,end=end)
    TIGER_MONEY_MARKET = web.DataReader('157450.KS',data_source='yahoo',start=start,end=end)
    KIWOOM_10Y = web.DataReader('148070.KS',data_source='yahoo',start=start,end=end)


    # change into monthly data

    #Korea stock
    KODEX_200 = pd.DataFrame(KODEX_200.groupby(pd.Grouper(freq='M'))['Adj Close'].mean())
    KOSDAQ_150 = pd.DataFrame(KOSDAQ_150.groupby(pd.Grouper(freq='M'))['Adj Close'].mean())
    KODEX_LVERAGE = pd.DataFrame(KODEX_LVERAGE.groupby(pd.Grouper(freq='M'))['Adj Close'].mean())
    KODEX_KOSDAQ_150_LVERAGE = pd.DataFrame(KODEX_KOSDAQ_150_LVERAGE.groupby(pd.Grouper(freq='M'))['Adj Close'].mean())
    KODEX_SAMSUNG = pd.DataFrame(KODEX_SAMSUNG.groupby(pd.Grouper(freq='M'))['Adj Close'].mean())
    TIGER_TOP10 = pd.DataFrame(TIGER_TOP10.groupby(pd.Grouper(freq='M'))['Adj Close'].mean())

    #Korea bond
    KODEX_10Y = pd.DataFrame(KODEX_10Y.groupby(pd.Grouper(freq='M'))['Adj Close'].mean())
    KODEX_TBS_3Y = pd.DataFrame(KODEX_TBS_3Y.groupby(pd.Grouper(freq='M'))['Adj Close'].mean())
    KODEX_KRW_CASH = pd.DataFrame(KODEX_KRW_CASH.groupby(pd.Grouper(freq='M'))['Adj Close'].mean())
    KODEX_KRW_CASH_PLUS = pd.DataFrame(KODEX_KRW_CASH_PLUS.groupby(pd.Grouper(freq='M'))['Adj Close'].mean())
    TIGER_MONEY_MARKET = pd.DataFrame(TIGER_MONEY_MARKET.groupby(pd.Grouper(freq='M'))['Adj Close'].mean())
    KIWOOM_10Y = pd.DataFrame(KIWOOM_10Y.groupby(pd.Grouper(freq='M'))['Adj Close'].mean())
```

```
    ### calculate return
    KODEX_200_logret = np.log(KODEX_200['Adj Close']/KODEX_200['Adj Close'].shift(1))
    KOSDAQ_150_logret = np.log(KOSDAQ_150['Adj Close']/KOSDAQ_150['Adj Close'].shift(1))
    KODEX_LVERAGE_logret = np.log(KODEX_LVERAGE['Adj Close']/KODEX_LVERAGE['Adj Close'].shift(1))
    KODEX_KOSDAQ_150_LVERAGE_logret = np.log(KODEX_KOSDAQ_150_LVERAGE['Adj Close']/KODEX_KOSDAQ_150_LVERAGE['Adj Close'].shift(1))
    KODEX_SAMSUNG_logret = np.log(KODEX_SAMSUNG['Adj Close']/KODEX_SAMSUNG['Adj Close'].shift(1))
    TIGER_TOP10_logret = np.log(TIGER_TOP10['Adj Close']/TIGER_TOP10['Adj Close'].shift(1))


    KODEX_10Y_logret = np.log(KODEX_10Y['Adj Close']/KODEX_10Y['Adj Close'].shift(1))
    KODEX_TBS_logret = np.log(KODEX_TBS_3Y['Adj Close']/KODEX_TBS_3Y['Adj Close'].shift(1))
    KODEX_KRW_CASH_logret = np.log(KODEX_KRW_CASH['Adj Close']/KODEX_KRW_CASH['Adj Close'].shift(1))
    KODEX_KRW_CASH_PLUS_logret = np.log(KODEX_KRW_CASH_PLUS['Adj Close']/KODEX_KRW_CASH_PLUS['Adj Close'].shift(1))
    TIGER_MONEY_MARKET_logret = np.log(TIGER_MONEY_MARKET['Adj Close']/TIGER_MONEY_MARKET['Adj Close'].shift(1))
    KIWOOM_10Y_logret = np.log(KIWOOM_10Y['Adj Close']/KIWOOM_10Y['Adj Close'].shift(1))

    # concat all data as dataframe
    stock_data = pd.DataFrame({'KODEX':KODEX_200_logret,
                           'KOSDAQ':KOSDAQ_150_logret,
                           'KODEX_SAMSUNG':KODEX_SAMSUNG_logret,
                           'KODEX_KOSDAQ_150_LVERAGE':KODEX_KOSDAQ_150_LVERAGE_logret,
                           'KODEX_LVERAGE':KODEX_LVERAGE_logret,
                           'TIGER_TOP10':TIGER_TOP10_logret})
```

```
    bond_data = pd.DataFrame({'KODEX_10Y':KODEX_10Y_logret,
                              'KODEX_TBS':KODEX_TBS_logret,
                              'KODEX_KRW_CASH':KODEX_LVERAGE_logret,
                              'KODEX_KRW_CASH_PLUS':KODEX_KOSDAQ_150_LVERAGE_logret,
                              'TIGER_MONEY_MARKET':KODEX_SAMSUNG_logret,
                              'KIWOOM_10Y':TIGER_TOP10_logret})

    bond_data.loc[0:2,list(bond_data.columns)]=0.0 # shift1하면서 생긴 none 데이터 결측치 처리

    stock_bond_return = pd.concat([stock_data,bond_data],axis=1)

    stock_bond_return.drop(index=stock_bond_return.iloc[:2, :].index.tolist(), inplace=True)
    stock_bond_return.dropna(axis=0, inplace=True)

    mu = stock_bond_return.mean()*3    # 분기별 데이터 사용
    sigma = stock_bond_return.cov()*3
    #mu = mu.values
    #mu = mu.reshape((8,))
    mu = mu.values.T
    sigma = sigma.values

    return mu, sigma
```

This module, calculate_mu_sigma, is responsible for calculating mu and sigma from the values of log returns on each bond and stock. Since this project is based on quarterly frequency not monthly, I processed loaded data by using ‘groupby’ method in order to only use monthly data. While calculating the log return, since null data occurs due to shifting, I changed null data to 0. Also, the important thing in here is that since we are using quarterly data, returns would be also based on quarterly data. I adjusted the time frequency of my parameters to set the length of sample returns, then, I multiply 3 on log return data while calculating both ‘mu’ and ‘sigma’. And parameters, ‘start’ and ‘end’, refer to the start and end of the period in which the data is retrieved.

```
def robust_optimizing(mu,sigma):
    w = cp.Variable(12)
    w_plus = cp.Variable(12)
    w_minus = cp.Variable(12)

    # box uncertainty set
    delta = np.array([0.005,0.005,0.005,0.005,0.005,0.005,0.01,0.01,0.01,0.01,0.01,0.01]).reshape(12,1)

    lambda0 = .1

    ret = mu.T*w
    aux = delta.T * (w_plus + w_minus)
    risk = cp.quad_form(w,sigma)

    objective = cp.Minimize(risk - lambda0*ret + lambda0*aux)

    prob = cp.Problem(objective,
                      [sum(w)==1,
                       w == w_plus - w_minus,
                       w_plus >=0,
                       w_minus>=0,
                       w >= -0.05,    # 공매도는 조금만
                       sum(w[:6]) >= sum(w[6:])*1.7   # 채권 비중이 2배 이상 높다
                      ])

    prob.solve()
```

```
    print('₩n Robust portfolio model₩n')
    print('₩n Robust portfolio model₩n')

    print('expected return',mu.dot(w.value))
    print('standard deviation:',np.sqrt(w.value.dot(sigma).dot(w.value)))
    print('optimal weight₩n')
    print('1_bond',w.value[0])
    print('2_bond',w.value[1])
    print('3_bond',w.value[2])
    print('4_bond',w.value[3])
    print('5_bond',w.value[4])
    print('6_bond',w.value[5])

    print('1_stock',w.value[6])
    print('2_stock',w.value[7])
    print('3_stock',w.value[8])
    print('4_stock',w.value[9])
    print('5_stock',w.value[10])
    print('6_stock',w.value[11])

    print('₩nbond:',sum([w.value[i] for i in range(6)]))
    print('stock:',sum([w.value[i+6] for i in range(6)]))

    return w.value
```

This module, robust_optimizing, is responsible for calculating optimal weights with robust optimization strategy. I set the value of delta on bond as 0.005 and stock as 0.01. Also, unlike usual robust optimization constraint, I added two more constraints that making threshold on short selling weight as -0.05 and setting the ratio of stocks and bonds as 1.7:1 in a realistic manner which is real proportion of National Pension Fund’s investment. And parameters, ‘mu’ and ‘sigma’, is calculated from calculate_mu_sigma module. The return value would be weights on each bond and stock.

1st portfolio selection : investing money on 2020/01/01 with past dat

```
mu_0, sigma_0 = calculate_mu_sigma(start='1/1/2018',end='12/31/2019')
weight_0 = robust_optimizing(mu_0,sigma_0)     # 2020/01/01 투자 비중
```

```
 Robust portfolio model


 Robust portfolio model

expected return 0.024700812325026047
standard deviation: 0.025472203739669824
optimal weight

1_bond -3.241073478541443e-22
2_bond -0.05
3_bond 0.680061009578863
4_bond -0.05
5_bond -0.050000000000000024
6_bond 0.13660565708780392
1_stock 0.5333333333333333
2_stock 9.011191830703347e-22
3_stock -0.050000000000000024
4_stock -0.05000000000000001
5_stock -0.050000000000000024
6_stock -0.05

bond: 0.6666666666666667
stock: 0.33333333333333326
```

First, calculate mu and sigma by using 2018~2019 historical data. Then, by using that mu_0 and sigma_0, we put them on robust_optimizing module. The reason for calulating mu_0 and sigma_0 first is we use them on simulation part.

Simulation

```
# random returns on bonds and stocks based on 2018,2019 historical data

num_samples = 3
sample_returns = pd.DataFrame()

for i in range(1000):

    np.random.seed(i)
    random_returns = np.random.multivariate_normal(mu_0,sigma_0,num_samples).T
    tmp = pd.DataFrame(random_returns).T
    sample_returns = pd.concat([sample_returns,tmp],axis=0)
sample_returns.columns = ['bond1','bond2','bond3','bond4','bond5','bond6',
                          'stock1','stock2','stock3','stock4','stock5','stock6']
```

```
sample_returns.head()
```

	bond1	bond2	bond3	bond4	bond5	bond6	stock1	stock2	stock3	stock4	stock5	stock6
0	-0.098712	-0.258657	-0.119050	-0.536912	-0.171205	-0.110114	0.031085	0.011948	-0.171205	-0.536912	-0.119050	-0.110114
1	-0.050586	-0.147664	-0.059276	-0.318028	-0.095534	-0.051278	0.034136	0.012245	-0.095534	-0.318028	-0.059276	-0.051278
2	-0.170757	-0.259878	-0.176803	-0.560687	-0.347736	-0.193687	0.055195	0.011990	-0.347736	-0.560687	-0.176803	-0.193687
0	-0.134826	-0.227204	-0.106971	-0.460820	-0.240097	-0.115198	0.051997	0.017479	-0.240097	-0.460820	-0.106971	-0.115198
1	-0.013541	-0.032300	-0.023655	-0.069512	-0.012316	0.004452	0.012628	0.003642	-0.012316	-0.069512	-0.023655	0.004452

This code generates 3000 number of returns since we rebalanced the data for 3 times and do the whole process for 1000 times. I used multi-variate normal distribution method while generating sample returns.

```
each_sample_wealth = []
num_samples = len(sample_returns)
rebalancing_period =4

mu_list = []
volatility_list = []

for i in tqdm(range(num_samples)):

    investing_amount = 10000
    mu = (sample_returns.to_numpy()[i])

    for rebalance in range(rebalancing_period):
        investing_amount = investing_amount

        w = cp.Variable(12)
        w_plus = cp.Variable(12)
        w_minus = cp.Variable(12)

        # box uncertainty set
        delta = np.array([0.005,0.005,0.005,0.005,0.005,0.005,0.01,0.01,0.01,0.01,0.01,0.01]).reshape(12,1)

        lambda0 = .1
        ret = mu.T*w

        aux = delta.T * (w_plus + w_minus)
        risk = cp.quad_form(w,sigma_0)

        objective = cp.Minimize(risk - lambda0*ret + lambda0*aux)

        prob = cp.Problem(objective,
                          [sum(w)==1,
                           w == w_plus - w_minus,
                           w_plus >=0,
                           w_minus>=0,
                           w >= -0.05,    # 공매도는 조금만
                           sum(w[:6]) >= sum(w[6:])*1.7   # 채권 비중이 1.7배 이상 높다
                           ])
        prob.solve()
```

```
        volatility_list.append(np.sqrt(w.value.dot(sigma_0).dot(w.value)))
        mu_list.append(mu.dot(w.value))

        investing_amount = np.dot(mu,w.value*investing_amount) + investing_amount
        mu = np.array([np.dot(mu[0],w.value[0]) for i in range(12)])   # rebalancing할 때마다, update

    each_sample_wealth.append(investing_amount)
```

I used two ‘for’ structures on this code. First ‘for’ structure which uses ‘i’, it refers the number of simulation, 3000, so it means that every time ‘i’ is updated, the simulation runs once. And second ‘for’ structure which uses ‘rebalance’ refers the rebalancing for each quarter. It goes four times in total, since April 1, July 1 and October 1 are rebalancing periods including Jan 1 as first investment based on 2018~2019 data. In this structure, I used robust optimization method with same constraints with suggested financial planning depicted on robust_optimizing module.‘mu’ and ‘investing_amount’ are updated while runs for once each from initial values mu_0 and 10000. So I saved all the expected return and standard deviation on each quarter as list format to find out average values on each of them. Also, total wealth on each simulation is saved as list format to gather them together at once to determine their distribution.

```
first_rebalancing_mu = []
second_rebalancing_mu = []
third_rebalancing_mu = []
fourth_rebalancing_mu = []

first_rebalancing_sg = []
second_rebalancing_sg = []
third_rebalancing_sg = []
fourth_rebalancing_sg = []

for i in range(len(mu_list)):

    if i%4 == 0:
        first_rebalancing_mu.append(mu_list[i])
        first_rebalancing_sg.append(volatility_list[i])

    elif i%4 == 1:
        second_rebalancing_mu.append(mu_list[i])
        second_rebalancing_sg.append(volatility_list[i])

    elif i%4 == 2:
        third_rebalancing_mu.append(mu_list[i])
        third_rebalancing_sg.append(volatility_list[i])

    elif i%4 == 3:
        fourth_rebalancing_mu.append(mu_list[i])
        fourth_rebalancing_sg.append(volatility_list[i])
```

```
# rebalancing 기간별 평균
first_mu = sum(first_rebalancing_mu)/len(first_rebalancing_mu)
second_mu = sum(second_rebalancing_mu)/len(second_rebalancing_mu)
third_mu = sum(third_rebalancing_mu)/len(third_rebalancing_mu)
fourht_mu = sum(fourth_rebalancing_mu)/len(fourth_rebalancing_mu)

first_sg = sum(first_rebalancing_sg)/len(first_rebalancing_sg)
second_sg = sum(second_rebalancing_sg)/len(second_rebalancing_sg)
third_sg = sum(third_rebalancing_sg)/len(third_rebalancing_sg)
fourth_sg = sum(fourth_rebalancing_sg)/len(fourth_rebalancing_sg)
```

```
print('first-period average expected return:',first_mu)
print('first-period average standard deviation:',first_sg)
print('₩n')

print('second-period average expected return:',second_mu)
print('second-period average standard deviation:',second_sg)
print('₩n')

print('third-period average expected return:',third_mu)
print('third-period average standard deviation:',third_sg)
print('₩n')

print('fourth-period average expected return:',fourht_mu)
print('fourth-period average standard deviation:',fourth_sg)
```

```
first-period average expected return: 0.05792412921447482
first-period average standard deviation: 0.0361550796295948


second-period average expected return: -0.00445583114329965
second-period average standard deviation: 0.017225055626246558


third-period average expected return: -0.002633963576228031
third-period average standard deviation: 0.017225055626246558


fourth-period average expected return: -0.001557007861783189
fourth-period average standard deviation: 0.017225055626246558
```

This code distributes quarterly expected returns and standard deviation values for each simulation. Then, I calculated average from 1000 samples on each expected return and standard deviation quarterly.

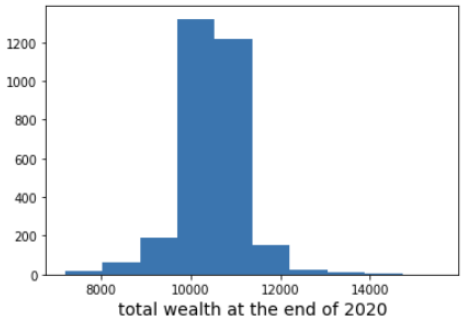
```
average_wealth = sum(each_sample_wealth)/len(each_sample_wealth)
print(average_wealth)
```

```
10496.921248752109
```

```
import matplotlib.pyplot as plt

plt.hist(each_sample_wealth)
plt.xlabel('total wealth at the end of 2020',fontsize=14)
```

```
Text(0.5, 0, 'total wealth at the end of 2020')
```



This code shows us the distribution on 3000 simulation's total wealth at the end of 2020 and average value of it.

2nd portfolio selection : investing money on 2020/04/01 with past dat

```
mu_1, sigma_1 = calculate_mu_sigma(start='4/1/2018',end='3/31/2020')
weight_1 = robust_optimizing(mu_1,sigma_1)    # 2020/04/01 투자 비중
```

```
 Robust portfolio model


 Robust portfolio model

expected return 0.021578889964799744
standard deviation: 0.036038933305931706
optimal weight

1_bond -4.848169115503968e-22
2_bond -0.05
3_bond 0.6771309507969863
4_bond -0.04999999999999998
5_bond -0.05
6_bond 0.1395357158696807
1_stock 0.5333333333333333
2_stock 7.796586064554329e-22
3_stock -0.05000000000000002
4_stock -0.049999999999999996
5_stock -0.05000000000000001
6_stock -0.049999999999999996

bond: 0.666666666666667
stock: 0.3333333333333333
```

I selected start date as 2018.04.01 not as 2020.01.01 since I used moving window method which utilizes same period length as first one due to small number of data between 2020.01.01~2020.03.31. Then, after mu_1 and sigma_1 are calcualted, weight_1 is also calculated by using them.

3rd portfolio selection : investing money on 2020/07/01 with past dat

```
mu_2, sigma_2 = calculate_mu_sigma(start='7/1/2018',end='6/30/2020')
weight_2 = robust_optimizing(mu_2,sigma_2)     # 2020/07/01 투자 비중
```

```
 Robust portfolio model


 Robust portfolio model

expected return 0.01768972456366152
standard deviation: 0.031291004436371374
optimal weight

1_bond -1.233993543390191e-20
2_bond 0.09090272453326077
3_bond 0.3106490354896836
4_bond -0.04999999999999998
5_bond -0.05
6_bond 0.3651149066437224
1_stock 0.5333333333333333
2_stock -4.969454210691046e-21
3_stock -0.049999999999999996
4_stock -0.04999999999999999
5_stock -0.050000000000000024
6_stock -0.049999999999999996

bond: 0.6666666666666667
stock: 0.3333333333333333
```

4th portfolio selection : investing money on 2020/010/01 with past dat

```
mu_3, sigma_3 = calculate_mu_sigma(start='10/1/2018',end='9/30/2020')
weight_3 = robust_optimizing(mu_3,sigma_3)    # 2020/10/01 투자 비중
```

```
 Robust portfolio model


 Robust portfolio model

expected return 0.03305467141005887
standard deviation: 0.03375473607624913
optimal weight

1_bond -0.049999999999999996
2_bond 0.11624447691820985
3_bond -0.05000000000000002
4_bond -0.04999999999999999
5_bond -0.04999999999999999
6_bond 0.7504221897484568
1_stock 0.5189655040764579
2_stock 1.4722131221687718e-23
3_stock -0.049999999999999996
4_stock -0.05000000000000001
5_stock -0.050000000000000024
6_stock -0.03563217074312466

bond: 0.6666666666666666
stock: 0.33333333333333326
```

Although these third and fourth portfolio rebalancing are not used in progress report, I solved out them for ‘Additional Works’. The calculation methods are same with first and second one.

Open, Close datafrmae

```
closed_df_1 = price_data(start='1/1/2020',end='3/31/2020')
closed_df_2 = price_data(start='4/1/2020',end='6/30/2020')
closed_df_3 = price_data(start='7/1/2020',end='9/30/2020')
closed_df_4 = price_data(start='10/1/2020',end='12/31/2020')
```

```
gain_1 = calculate_gain(10000,weight_0,closed_df_1)   # 2020/03/31 wealth
gain_2 = calculate_gain(gain_1,weight_1,closed_df_2)  # 2020/06/30 wealth
gain_3 = calculate_gain(gain_2,weight_2,closed_df_3)  # 2020/09/30 wealth
gain_4 = calculate_gain(gain_3,weight_3,closed_df_4)  # 2020/12/31 wealth
```

```
gain_1
```

9528.01876410589

```
gain_2
```

9778.639577110458

```
gain_3
```

9992.173033956738

```
gain_4
```

10391.013022123934

I solved out total wealth on each quarter by utilizing modules I made. Here the important thing is we should use $10000 for first period but second, third and fourth period's investment_amount parameter is used by the outcome values from just before total gain. And gain_3 and gian_4 are also calculated are not used in progress report but I solved out them for using in 'additional works'

Portfolio evaluation

Time-weighted rate of return

```
cash_list = [10000,gain_1,gain_2]
return_list = []
for i in range(2):
    return_list.append((cash_list[i+1]-cash_list[i])/cash_list[i])
time_weighted_return = ((1+return_list[0])*(1+return_list[1]))**(1/2)-1
time_weighted_return
```

```
-0.011129959139702827
```

```
cash_list = [10000,gain_1,gain_2,gain_3,gain_4]
return_list = []
for i in range(4):
    return_list.append((cash_list[i+1]-cash_list[i])/cash_list[i])
```

```
return_list
```

```
[-0.09087781827534745,
 -0.2305499742838632,
 0.10032175704855088,
 0.20186868084609702]
```

```
time_weighted_return = ((1+return_list[0])*(1+return_list[1])*(1+return_list[2])*(1+return_list[3]))**(1/4)-1
time_weighted_return
```

```
-0.019280415905349346
```

This code depicts the method of calculating time-weighted rate of return for portfolio evaluation. Two kinds of time-weighted rate of returns are made which each for progress report and additional works. A part where first cash_list which only uses three values in list are for progress report and the other where uses five values in list are for additional work.

20161342 Keonwoo Kim

Change Strategy on 2020.06.01

```
# random returns on bonds and stocks based on 2018,2019 historical data

num_samples = 2
sample_returns = pd.DataFrame()

for i in range(1000):

    np.random.seed(i+100000)
    random_returns = np.random.multivariate_normal(mu_2,sigma_2,num_samples).T
    tmp = pd.DataFrame(random_returns).T
    sample_returns = pd.concat([sample_returns,tmp],axis=0)
sample_returns.columns = ['bond1','bond2','bond3','bond4','bond5','bond6',
                          'stock1','stock2','stock3','stock4','stock5','stock6']


aaa = []
each_sample_wealth = []
num_samples = len(sample_returns)
rebalancing_period = 2

mu_list = []
volatility_list = []

for i in tqdm(range(num_samples)):

    investing_amount = gain_2
    mu = (sample_returns.to_numpy()[i])
```

```
    for rebalance in range(rebalancing_period):
        investing_amount = investing_amount

        w = cp.Variable(12)
        w_plus = cp.Variable(12)
        w_minus = cp.Variable(12)

        # box uncertainty set
        delta = np.array([0.01,0.01,0.01,0.01,0.01,0.01,0.03,0.03,0.03,0.03,0.03,0.03]).reshape(12,1)

        lambda0 = .1
        ret = mu.T*w

        aux = delta.T * (w_plus + w_minus)
        risk = cp.quad_form(w,sigma_0)

        objective = cp.Minimize(risk - lambda0*ret + lambda0*aux)

        prob = cp.Problem(objective,
                          [sum(w)==1,
                           w == w_plus - w_minus,
                           w_plus >=0,
                           w_minus>=0,
                           w >= -0.05,    # 공매도는 조금만
                           sum(w[:6]) >= sum(w[6:])*1.2  # 채권 비중이 1.5배 이상 높다
                           #sum(w[6:])>=0.4,   # 채권 비중이 2배 이상 높다
                           #sum(w[6:])<=0.6
                           ])
        prob.solve()

        #print('expected return',mu.dot(w.value))
        #print('standard deviation:',np.sqrt(w.value.dot(sigma_0).dot(w.value)))

        volatility_list.append(np.sqrt(w.value.dot(sigma_0).dot(w.value)))
        mu_list.append(mu.dot(w.value))

        investing_amount = np.dot(mu,w.value*investing_amount) + investing_amount
        mu = np.array([np.dot(mu[0],w.value[0]) for i in range(12)])   # rebalancing할 때마다, update

    each_sample_wealth.append(investing_amount)
```

```
first_rebalancing_mu = []
second_rebalancing_mu = []

first_rebalancing_sg = []
second_rebalancing_sg = []

for i in range(len(mu_list)):

    if i%4 == 0:
        first_rebalancing_mu.append(mu_list[i])
        first_rebalancing_sg.append(volatility_list[i])

    elif i%4 == 1:
        second_rebalancing_mu.append(mu_list[i])
        second_rebalancing_sg.append(volatility_list[i])
```

,

```
# rebalancing 기간별 평균
first_mu = sum(first_rebalancing_mu)/len(first_rebalancing_mu)
second_mu = sum(second_rebalancing_mu)/len(second_rebalancing_mu)

first_sg = sum(first_rebalancing_sg)/len(first_rebalancing_sg)
second_sg = sum(second_rebalancing_sg)/len(second_rebalancing_sg)
```

```
print('first-period average expected return:',first_mu)
print('first-period average standard deviation:',first_sg)
print('₩n')

print('second-period average expected return:',second_mu)
print('second-period average standard deviation:',second_sg)
print('₩n')
```

```
first-period average expected return: 0.09669400454486762
first-period average standard deviation: 0.0481591942050389


second-period average expected return: -0.0028259510703429725
second-period average standard deviation: 0.024411833282680814
```

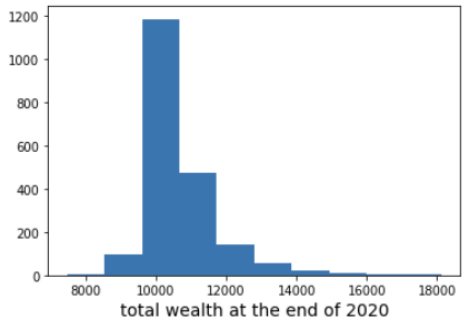
```
average_wealth = sum(each_sample_wealth)/len(each_sample_wealth)
print(average_wealth)
```

```
10689.73481267045
```

```
import matplotlib.pyplot as plt

plt.hist(each_sample_wealth)
plt.xlabel('total wealth at the end of 2020',fontsize=14)
```

```
Text(0.5, 0, 'total wealth at the end of 2020')
```



This code is for simulation with new strategy which increases the weight on stock. The difference between it and a simulation code part described earlier is constraint on robust optimization where change the ratio as 1.2: 1. Also, since the strategy was made after the progress report, the number of quarterly periods for rebalancing has changed from four to two. And the rest is all done in the same way.