MATH5010 INTRO TO THE MATH IN FINANCE
Final Project | FALL 2019

# Optimal Sector Trading Portfolio Construction using Quadratic Programming

Juyeon Kang (jk4206)
Kebing Li (kl3185)

## 1. Introduction

The stock market as well as economy moves repeatedly in cycles over and over. However, investors would not like the radical change of their long-term plans and they would rather like to implement simple tweaks to their portfolio to help cushion catastrophic losses or extra gains. In that sense, our goal of this project is to get low risk portfolios by allocating optimal sector weights and to observe if the optimal sector weights allow us to have better market returns than benchmark.

We obtain historical returns of 11 sector ETFs and S&P 500, our benchmark, from 2009 to 2019 from Yahoo Finance. The sector ETFs are all value-weighted to emphasize an industry allocation effect rather than a stock selection effect. Through this project, we see if sector allocation portfolio can outperform the market.

## 2. Main objectives and plan

Our study is mainly focused on finding a low-risk portfolio while obtaining at least a pre-specified return level. We found the quadratic programming method would be one of the ways to solve the problem, in which it computes the weights that will minimize the variance of the daily returns while ensuring a return level. After finding the optimal portfolio allocation, we apply them to the market in our test period of 2009-2019. Their performance with respect to the benchmark in 2018 helps us determine the effects of our portfolio. We assume that a naked short-selling is allowed and there is no trading cost accompanied by rebalancing portfolios to adjust weights to our optimal weights every day.

## 3. Analysis description

### 3.1. Data description

#### 3.1.1 Data selection

While the traditional way of getting weight allocations of a portfolio would be the way by managing the weights among different asset classes such as stocks, fixed income and cash, we would like to consider the weight allocations among the sectors of the stock market. In convention, stock markets are divided into 11 sectors and in order to estimate the performance of each sector, we decided to use its corresponding sector ETF as a proxy.

An Exchange-Traded Fund (ETF) is a type of an investment fund. Just like stocks, it is

traded on exchanges. There could be different types of assets under an ETF, such as stocks, commodities and bonds, but for the sector ETFs we chose to use, the underlyings are all stocks within that sector.

The reason why we decided to use ETFs rather than select individual stocks is the sector ETFs could represent the sector's characteristics as a whole better, and many other previous sector-based trading strategies also used sector ETFs as their instruments due to its simplicity and accessibility. We collected 11 sector ETFs with the highest volumes from yahoo finance. We only use ETFs with the largest volumes because they tend to be more liquid, and hence we can avoid some fill costs in trading execution. The data ranges from 2009 to 2019.

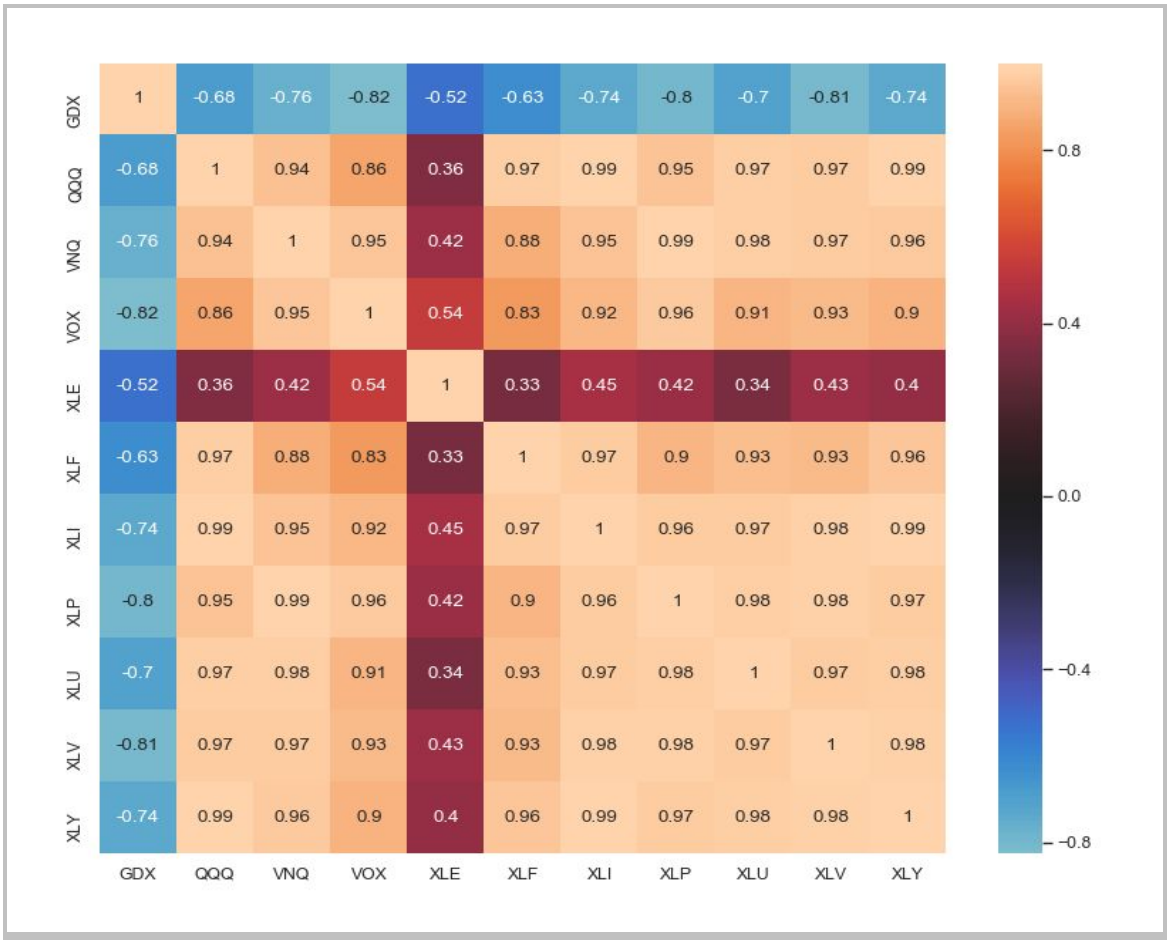
Specific description of data is as follows.

**Table 1: 11 Sectors of ETF**

| | |
|---|---|
| GDX | VanEck Vectors Gold Miners ETF |
| QQQ | Invesco QQQ |
| VNQ | Vanguard Real Estate Index Fund |
| VOX | Vanguard Communication Services ETF |
| XLE | Energy Select Sector SPDR Fund |
| XLF | Financial Select Sector SPDR Fund |
| XLI | Industrial Select Sector SPDR Fund |
| XLP | Consumer Staples Select Sector SPDR Fund |
| XLU | Utilities Select Sector SPDR Fund |
| XLV | Health Care Select Sector SPDR Fund |
| XLY | Consumer Discretionary Select Sector SPDR Fund |

### 3.1.2. Correlation matrix

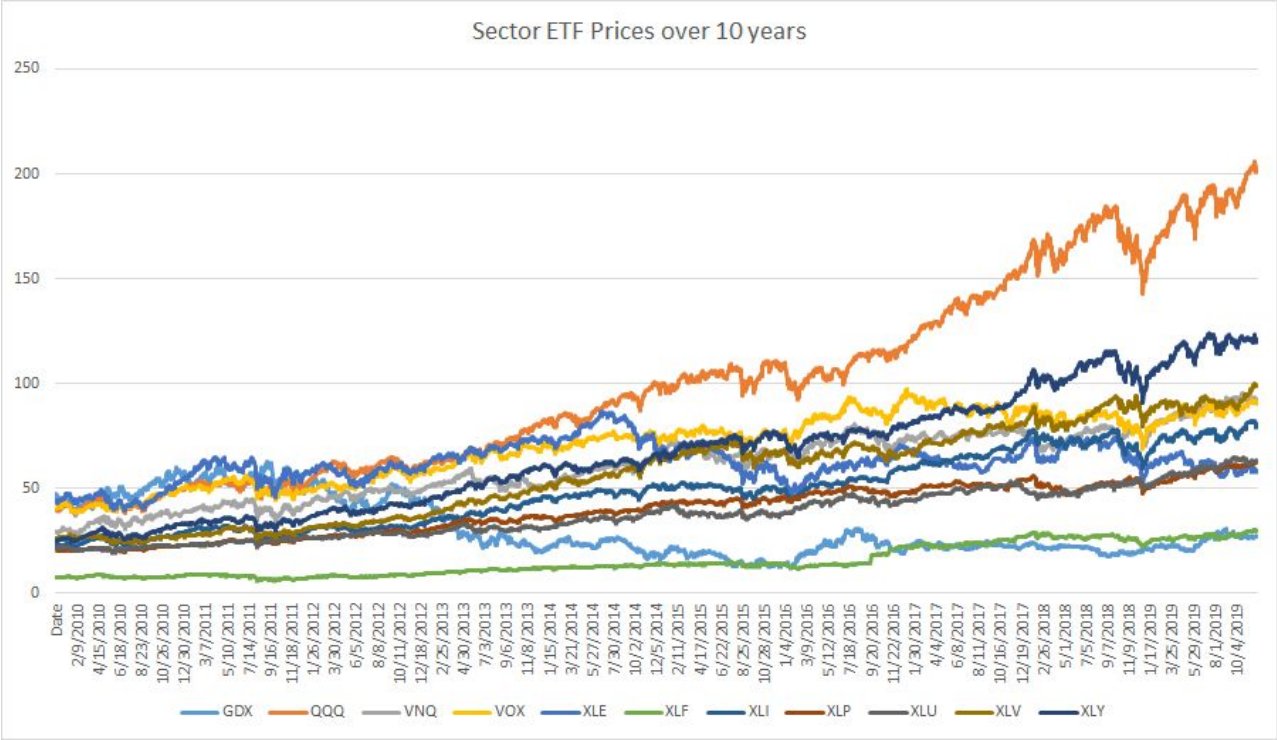
Below is the correlation matrix of our ETF data. We can see that most sector ETFs are positively correlated with each other except for GDX, the sector ETF for precious metals companies, which is negatively correlated with all other ETFs because Gold and other precious metals are considered as safe-haven investments and tend to have negative correlation with the general markets. One other might be worth noticing is that XLE, which is the energy sector ETF, appears to show moderate correlations with all other sector ETFs, which probably shows that the energy industry has some industry-specific risk factors that might be perpendicular to the general markets.




### 3.1.3. Sector ETF prices over 10 years

Below is a plot for the price changes over the past 10 years. We could see that QQQ, which is the technology sector ETF and is always considered as a proxy for Nasdaq, performs much better than all other sectors and its worth in 2019 is more than 3 times higher than it was in 2009. This is due to the rapid growth of the major technology firms like Apple, Google, Amazon, Facebook, and etc over the last ten years. However, GDX is negatively correlated to the general

market trend, so it has a downward shape.




### 3.2. Strategy: Weight allocation

#### 3.2.1. Quadratic programming

For the purpose of weight allocations among the sectors of the stock market, the quadratic programming is used to optimize the weight allocation by minimizing its risk while ensuring to achieve a certain amount of return. Through the quadratic programming computation, a portfolio with the lowest volatility can be achieved and it will gain at least a user-specified return level. As a default, we use the expected return of an equal weighting portfolio to be the return level if it is not explicitly specified, which means for example, if we want to use a 5-day look-back window in simulation, then we are essentially trying to compute a weight of allocation on Day 6, where the weights are calculated given the information (returns) of the previous 5 days such that the volatility of the previous 5 days is minimized under this weight and the expected return given this weight would be at least equal to the 5-day expected return assuming equal weighting.

In our implementation, we require the weights add up to one, so that we will have a net-long portfolio. However, if one sets the weights adding up to zero, then one may achieve a self-financing portfolio when ignoring the transaction costs and costs associated with short selling.

The optimization problem for optimal weights allocations is as follows:

$$minimize\ Var(\sum_{i=1}^{11}(x_i * R_i)) = X^T Cov(R)X$$

$$s.t.\ \sum_{i=1}^{11} x_i = 1\ and\ \forall_i : \left[E[r_{N,1}] \dots E[r_{N,11}]\right] * X \geq E[\ R \mid equal\ weighting]$$

Here, $R = \begin{bmatrix} r_{1,1} & \dots & r_{1,11} \\ \vdots & \ddots & \vdots \\ r_{N,1} & \dots & r_{N,11} \end{bmatrix}$ is the log-return matrix for the last N periods and $r_{i,j}$ is the daily log return of the j-th ETF in the previous i-th day. Our objective here is to minimize the variance of the portfolio taking into account the log-return matrix R, where $x_i$ is the weight assigned on the i-th ETF, and we can rewrite the expression as $X^T Cov(R)X$, where X is the 11*1 vector of sector allocation. At the same time, we also want the weighted returns greater than the equal-weight return and the weights add up to 1.

### 3.2.2. Construction

The key assumption for our strategy is that the price movement (or the return movement) of the selected ETFs will roughly keep the same in the next N trading days as the previous N trading days. Therefore, we can compute the optimal weights every N trading days and only rebalance the portfolio each time having new weights. This is saying that if we choose the 5-day look-back window again, then we are using the information from $t_1$ to $t_5$ to compute the weight allocations for $t_6$ to $t_{10}$, and we rebalance the weights on Day 11 using the new information from $t_6$ to $t_{10}$, and etc.

Our strategy uses signal weighting which is the weights we computed by Quadratic Programming. Our portfolio is also implicitly hedged already since every time we hold both long and short positions in our portfolio and the market exposure is fixed as the sum of the weights equals to 1. The coding would be available in the appendix and attached as a separate file.

## 4. Results of statistical analysis and interpretation

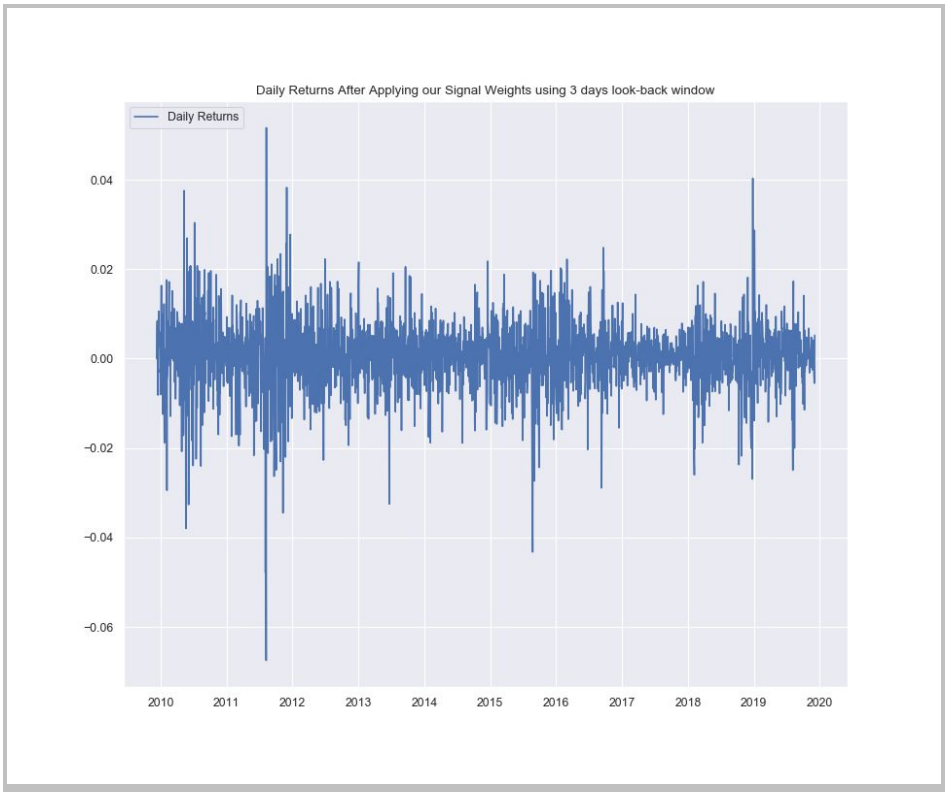
### 4.1. Results from different look-back window length

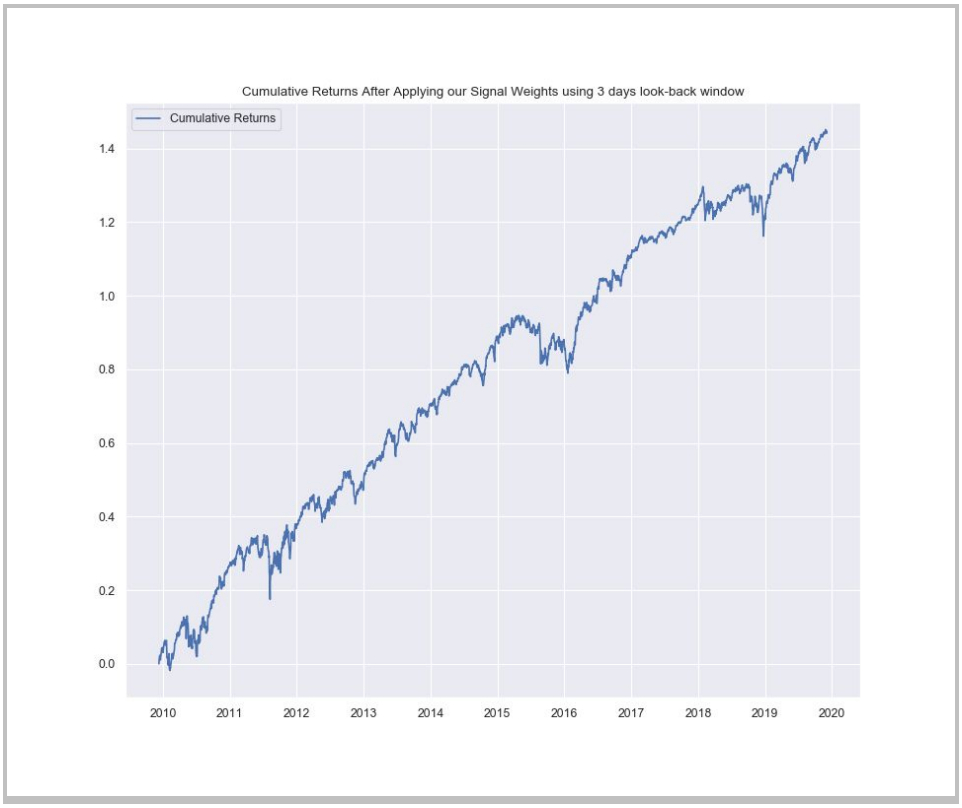
In order to see the effect of different look-back window lengths on our strategy performance, we ran the simulations on 11 different windows. The table below presents all the key statistics for each window length.

| Look-back Window Length | Expected Daily Return | Annualized Return | Cumulative Return | Daily Vols | Annuals Vols | Sharpe Ratio |
|---|---|---|---|---|---|---|
| 2 | 0.0559% | 15.1249% | 140.6072% | 0.7702% | 12.2268% | 1.24 |
| 3 | 0.0575% | 15.5959% | 144.6264% | 0.7792% | 12.3689% | 1.26 |
| 4 | 0.0453% | 12.0837% | 113.7849% | 0.7883% | 12.5142% | 0.97 |
| 5 | 0.0483% | 12.9357% | 121.2918% | 0.7968% | 12.6492% | 1.02 |
| 7 | 0.0429% | 11.4189% | 107.7206% | 0.7936% | 12.5976% | 0.91 |
| 11 | 0.0424% | 11.2799% | 106.3071% | 0.7964% | 12.6422% | 0.89 |
| 12 | 0.0394% | 10.4249% | 98.5945% | 0.8012% | 12.7183% | 0.82 |
| 14 | 0.0398% | 10.5541% | 99.6782% | 0.8019% | 12.7295% | 0.83 |
| 15 | 0.0393% | 10.4180% | 98.4147% | 0.8008% | 12.7123% | 0.82 |
| 21 | 0.0376% | 9.9323% | 93.8104% | 0.8079% | 12.8250% | 0.77 |
| 28 | 0.0389% | 10.2864% | 96.7246% | 0.8076% | 12.8198% | 0.80 |

From the table, we can see that the best-performer is when we select a 3-day look-back window length, and it generates a cumulative return of 140.61% over ten years and a decent Sharpe Ratio of 1.26. In general, strategies using shorter window lengths outperforms strategies using longer window lengths, which is quite what we would expect because the return patterns are more likely to maintain in a short period and it's getting really hard to continue when it comes to 2 or 3 weeks (14days or 21days). Interestingly, from the table, we can see that the number 11 seems to be the cut point between short and long window length, since the performance drops significantly when N>11.

The daily return and cumulative return plots when N = 3 is as follows.






### 4.2. Comparing to the benchmark

Since our benchmark is the S&P 500, we want to compare our strategy results with the S&P's performance over the last ten years. The table below shows a comparison between our best-performer and SPX.

| Strategy | Expected Daily Return | Annualized Return | Cumulative Return | Daily Vols | Annuals Vols | Sharpe Ratio |
|---|---|---|---|---|---|---|
| SPX | 0.04% | 11.26% | 112.60% | 0.78% | 12.45% | 0.87 |
| N=3 | 0.06% | 15.60% | 144.63% | 0.78% | 12.37% | 1.26 |

We see that our strategy outperforms the S&P 500 (SPX) by achieving greater return (4% per year) while having roughly the same risk level. Hence, our strategy is basically successful and reaches our expectations.

## 5. Conclusions

Even though our strategy achieves our objectives, it has certain constraints and there is also room for future improvements, which we will talk a little bit here.

### 5.1 Constraints

The first constraint is that we assume the future security price movements are unchanged for at least a short period of time, which makes our strategy a momentum-like strategy in nature. Therefore, if the momentum pattern doesn't hold, then our strategy may fail.

The second constraint is that even though we choose to only include the most liquid ETFs in the markets, short-selling in ETFs can still be competitive and incur some additional costs since short-selling ETFs is a widely-adopted hedging method.

Lastly, we do not take into account any costs involved, so our real performance might be slightly lower than what we calculated above.

### 5,2 Future Improvements

For the simplicity of our implementation, we rebalance our portfolio every time a new weight is calculated. However, even if our look-back window length is fixed, we may want to find out an optimal trading frequency. For example, if we use the 5-day look-back window, we can still do rebalancing on a daily basis, where we use the information from $t_1$ to $t_5$ to compute the weights on Day 6 and use the information from $t_2$ to $t_6$ to update the weights on Day 7, and so forth. In such a way, we may further improve our strategy performance.

Next, we can do some stress testing on extreme market conditions. Since we only look at the market after the 2008-2009 financial crisis, we can use that period for example as a stress test study.

This strategy employs the methodology of Quadratic Programming; however, there are many other mean-variance optimization tools or algorithms available to be incorporated in portfolio construction setting.

## 6. Reference


https://www.mathworks.com/help/optim/examples/using-quadratic-programming-on-portfolio-optimization-problems.html

http://support.sas.com/documentation/cdl/en/ormpug/63975/HTML/default/viewer.htm#ormpug_qpsolver_sect013.htm

https://www.morningstar.com/indexes/spi/spx/risk

https://scaron.info/blog/quadratic-programming-in-python.html

https://ycharts.com/indicators/10_year_treasury_rate

https://www.macrotrends.net/2488/sp500-10-year-daily-chart

https://etfdb.com/

https://finance.yahoo.com/


## 7. Appendix (Codes)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import qpsolvers as qp
from statistics import mean
from numpy import array, dot
import math

sns.set()

# open the DATA.xlsx file
data = pd.read_excel("DATA.xlsx")

# drop the first column
da = data.drop(data.columns[0], axis=1)

# Correlation Plot
plt.figure(figsize=(12,10), dpi=80)
corr_plot = sns.heatmap(da.corr(), xticklabels=da.corr().columns,
        yticklabels= da.corr(),
        center=0, annot='True')
plt.show()
```

```
# save the correlation plot to the directory
corr_plot.figure.savefig('correlation_plot.png')

# plot the price movements over the past ten years
plt.rcParams["figure.dpi"] = 140
data.plot('Date', list(data.columns[1:]), kind = 'line')

# name of the securities
header = list(data.columns)[1:]

# convert to numpy array
df = data.to_numpy()

# get the dates
date = df[:, 0]

# all prices numpy matrix
prices = df[:, 1:]

# compute the log returns from the price matrix
log_returns = np.diff(np.log(prices.astype('float')), axis=0)

def strategy(log_returns, rolling_window = 11, return_level = None):

    """takes in a log return matrix, look-back window,
       and a return level and returns a sector weight"""

    # use the selected part of the return matrix
    selected_period = log_returns[:rolling_window, :]

    # compute the average for each ETF over last N days
    r = np.mean(selected_period, axis=0)

    # compute the equal weighting expected returns
    naive_average = np.mean(r)

    # ensure the user can give his/her own return level
    if return_level == None:
        H = np.negative(np.asarray(naive_average))
    else:
        H = np.negative(np.asarray(return_level))
```

```
    # the arguments for the QP problem
    R = np.cov(selected_period.T)
    R = R + np.identity(R.shape[0])*(1e-3)
    A = np.ones((11, ))
    Z = np.zeros((11, ))
    B = np.ones((1,))
    g = np.negative(r)

    # compute the weights
    weights = qp.solve_qp(P=R, q=Z, G=g, h=H, A=A, b=B)
    return weights.tolist()

#strategy(log_returns, rolling_window=11, return_level=None)

def run_simulation(window_len, returnlevel=None):

    """ run the simulation for our strategy given a window
        length and a return level"""

    weights = []

    # compute a list of weights
    for i in range(math.floor(log_returns.shape[0] / window_len)):
        weights.append(strategy(log_returns[i * window_len:
                                            (i + 1) * window_len, :]
                                , window_len, returnlevel))

    # fill the list of weights to align the size with
    # the shrunk price matrix
    full_weights = np.repeat(weights, window_len, axis=0)

    shrunk_data = prices[window_len:, :]

    while full_weights.shape[0] > shrunk_data.shape[0]:
        full_weights = np.delete(full_weights, -1, 0)

    # create return matrix ignoring the first "window length" days
    log_returns_lag1 = np.diff(np.log(shrunk_data.astype('float')),
                               axis=0)
    log_returns_lag1 = np.vstack((np.zeros((1, 11)), log_returns_lag1))

    # calculate the weighted returns
```

```
    weighted_returns = np.multiply(full_weights, log_returns_lag1)

    # get the total returns
    total_returns = np.sum(weighted_returns, axis=1)

    total_returns.tolist()

    # plot daily return over the entire period
    plt.figure(0, figsize=(12, 10), dpi=80)
    plt.plot(date.tolist()[window_len:], total_returns)
    plt.legend(('Daily Returns',), loc='upper left')
    plt.title("Daily Returns After Applying our Signal Weights using "
              + str(window_len) + " days look-back window")
    plt.savefig('Daily Returns with ' + str(window_len)
                                  + ' days look-back window.png')

    # get the summary statistics
    expected_return = np.average(total_returns)
    print("Expected Return is: ", expected_return*100, "%")
    annual_return = (math.pow(expected_return + 1, 252) - 1)
    print("Annualized Return is: ", annual_return*100, "%")

    cumulative_returns = np.cumsum(total_returns)
    print("Cumulative return is: ", cumulative_returns.tolist()[-1]
          *100, "%")

    # plot cumulative returns
    plt.figure(1, figsize=(12, 10), dpi=80)
    plt.plot(date.tolist()[window_len:], cumulative_returns)
    plt.legend(('Cumulative Returns',), loc='upper left')
    plt.title("Cumulative Returns After Applying our Signal Weights using "
              + str(window_len) + " days look-back window")
    plt.savefig('Cumulative Returns with ' + str(window_len)
                + ' days look-back window.png')
    # calculate performance measures:
    # Information Ratio:
    std = np.std(total_returns)
    print("Daily vol is: ", std*100, "%")
    annual_std = std * math.sqrt(252)
    print("Annual vol is:", annual_std*100, "%")
    Sharpe_Ratio = annual_return / annual_std
    print("Sharpe Ratio is: ", Sharpe_Ratio)
```

```
run_simulation(window_len=3)
```