Portfolio Optimization

John Furlong

April 2020

1 Abstract

Given a set of assets and limited funds, it can be difficult to determine the optimal way to distribute your budget. While high return is typically the goal, it also typically means accepting higher levels of risk in your investments. Mean-Variance optimization is a quadratic programming problem, which rests upon the linear algebra of minimizing quadratic functions. In this report, we will look at an example of a portfolio constructed of three assets, and their optimal distributions.

2 Introduction

Suppose you can invest your money into two businesses: a sunglasses company and a raincoat company. If you invest it all in the raincoat company, you will earn big if it is a rainy year; however, if you invest it all in the sunglasses company then you lose. If you invest half into both, then your portfolio of investments is sub-optimal. You won't lose as much, but you won't profit big either.

Modern Portfolio Theory (MPT) was developed by Harry Markowitz in his 1952 publication, "Portfolio Selection". Markowitz outlined a method known as Mean-Variance Optimization, based on the assumption that a rational investor should either maximize returns for a given level of risk, or minimize risk for a given level of expected return.

2.1 Risk and Return

Returns
Depending on the type of potential investments (short-term vs. long-term), an investor might want to calculate the rate of return using different periods. For the sake of example, let monthly returns for a stock be defined as:

$$r_{jt} = \frac{p_{jt} - p_{jt-1}}{p_{jt-1}}$$

where

$$r_{jt} = \text{return of stock j in month t}$$
$$p_{jt} = \text{price of stock j in month t}$$

Return on investment (ROI) is a stochastic quantity, therefore we will model it using a vector of random quantities:

$$\tilde{r} = \begin{bmatrix} \tilde{r_1} \\ \tilde{r_2} \\ ... \\ \tilde{r_n} \end{bmatrix} \quad \text{where } \tilde{r_j} \text{ is the random return for stock j}$$

Portfolio Allocations can also be modeled using a vector:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ ... \\ x_n \end{bmatrix} \quad \text{where } x_j \text{ is the amount \$ invested in stock j}$$

The total expected return for a portfolio is given by the standard Euclidian dot product between these two vectors; however, the return vector is a random

variable, therefore we need to use expected quantities:

$$E\big[\sum_{i=1}^{n} \tilde{r_i} x_i\big] = \sum_{i=1}^{n} E\big[\tilde{r_i}\big] x_i = \sum_{i=1}^{n} \bar{r_i} x_i = \langle \bar{r}, x \rangle$$

where $\bar{r_i}$ is the expected return for stock i

Our goal will be to minimize the total risk on a portfolio of investments, constrained to a minimum level of expected return. Let r_{min} be the minimum level of expected return:

$$\langle \bar{r}, x \rangle \geq r_{min}$$

Risk
We will use the volatility of a stock to model the risk associated with it. This quantity is calculated as the variance in the expected return:

$$\begin{aligned} Var\big[\sum_{i=1}^{n} \tilde{r_i} x_i\big] &= E\big[(\sum_{i=1}^{n} \tilde{r_i} x_i - \sum_{i=1}^{n} \bar{r_i} x_i)^2\big] \\ &= E\big[(\sum_{i=1}^{n} (\tilde{r_i} - \bar{r_i}) x_i)(\sum_{i=1}^{n} (\tilde{r_j} - \bar{r_j}) x_j)\big] \\ &= \sum_{i=1}^{n} \sum_{j=1}^{n} x_i x_j E\big[(\tilde{r_i} - \bar{r_i})(\tilde{r_j} - \bar{r_j})\big] \\ &= \sum_{i=1}^{n} \sum_{j=1}^{n} x_i x_j \sigma_{ij} \\ &= \vec{x}^T Q \vec{x} \end{aligned}$$

where Q is a covariance matrix

Note: We are able to express variance in terms of covariance, because for any 2 random variables X and Y:

$$Var(X) = Cov(X, X)$$

Covariance is also *symmetric*

$$Cov(X, Y) = Cov(Y, X)$$

Therefore, the portfolio variance modeled in quadratic matrix notation becomes:

$$\begin{aligned} Var[r(x)] &= \begin{pmatrix} x_1 & x_2 \end{pmatrix} \begin{pmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{21} & \sigma_2^2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \\ &= \vec{x}^T Q \vec{x} \end{aligned}$$

Note: The equation: $Var[r(x)] = \vec{x}^T Q \vec{x}$
holds true for $n \geq 2$

3 Mean-Variance Optimization

The complete Mean-Variance Optimization Problem can be formulated as a minimization of a quadratic:

$$\begin{aligned} \text{Minimize: } & \frac{1}{2}\vec{x}^T Q \vec{x} \\ & \text{such that:} \\ & \langle \vec{x}, \vec{r} \rangle = \vec{r_0} \\ & \langle \vec{x}, \vec{e} \rangle = 1 \end{aligned}$$

where
$\vec{e} = (1, 1, ..., 1)^T$
Q is the covariance matrix of the stocks
$\vec{r}$ is the vector of the expected returns
r_0 is the desired level of expected return

3.1 Solving the Optimization Model

In order to solve the optimization model, we can take advantage of the fact that our objective function is defined as a *convex function*, which is also constrained by other convex functions. Convex optimization problems fall under a subclass of quadratic programming problems. To solve the constrained optimization problem, we formulate what is known as the *Lagrangian* for the model:

$$L(x, \lambda_1, \lambda_2) = \frac{1}{2}\vec{x}^T Q \vec{x} - \lambda_1(r_{min} + \langle \vec{r}, \vec{x} \rangle) - \lambda_2(1 - \langle \vec{e}, \vec{x} \rangle)$$
$$s.t : \lambda_1, \lambda_2 \in R$$

Let $\vec{x}^*$ be a solution to the optimization problem. In order for $\vec{x}^*$ to be an optimal solution, we must ensure that the partial derivatives of the Lagrangian equation disappear when evaluated at λ_1 and λ_2:

$$\begin{aligned} \frac{\partial L}{\partial \vec{x}} &= Q\vec{x} - \lambda_1 \vec{r} - \lambda_2 \vec{e} = 0 \\ \frac{\partial L}{\partial \lambda_2} &= r_{min} - \langle \vec{x}, \vec{r} \rangle = 0 \\ \frac{\partial L}{\partial \lambda_2} &= 1 - \langle \vec{x}, \vec{e} \rangle = 0 \end{aligned}$$

Written in matrix form, the partial derivatives can be expressed as follows:

$$Q\vec{x} = \begin{pmatrix} Q^{-1}\vec{e} & Q^{-1}\vec{r} \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \end{pmatrix}$$

By multiplying both sides of the equation by Q^{-1}, we can find the optimal solution $\vec{x}^*$ in terms of λ_1 and λ_2:

$$\vec{x}^* = x(\lambda_1, \lambda_2) = \lambda_1 Q^{-1}\vec{r} + \lambda_2 Q^{-1}\vec{e}$$

In order to solve for the optimal values of λ_1 and λ_2, we use our optimality constraints, as well as the previous equation for $\vec{x}^*$ in terms of λ_1 and λ_2:

$$\begin{pmatrix} 1 \\ r_{min} \end{pmatrix} = \begin{pmatrix} 1 & 1 & ... & 1 \\ \bar{r_1} & \bar{r_2} & ... & \bar{r_n} \end{pmatrix} \begin{pmatrix} x_1(\lambda_1, \lambda_2) \\ x_2(\lambda_1, \lambda_2) \\ ... \\ x_n(\lambda_1, \lambda_2) \end{pmatrix}$$

$$\begin{pmatrix} 1 \\ r_{min} \end{pmatrix} = \begin{pmatrix} \vec{e}^T Q^{-1}\vec{e} & \vec{r}^T Q^{-1}\vec{e} \\ \vec{r}^T Q^{-1}\vec{e} & \vec{r}^T Q^{-1}\vec{r} \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \end{pmatrix}$$

$$\begin{pmatrix} \lambda_1 \\ \lambda_2 \end{pmatrix} = A^{-1} \begin{pmatrix} 1 \\ r_{min} \end{pmatrix}$$

$$\text{Where } A^{-1} = \begin{pmatrix} \vec{e}^T Q^{-1}\vec{e} & \vec{r}^T Q^{-1}\vec{e} \\ \vec{r}^T Q^{-1}\vec{e} & \vec{r}^T Q^{-1}\vec{r} \end{pmatrix}$$

Finally, once we've solved for the values of λ_1 and λ_2, we can plug them back into our equation and solve for $\vec{x}^*$:

$$\vec{x}^* = \lambda_1 Q^{-1}\vec{r} + \lambda_2 Q^{-1}\vec{e}$$

3.2 The Efficient Frontier

For a given level of desired return r_0, we can find the portfolio with minimal expected variance. By varying the value of r_0, we can construct what is known as an *efficient frontier* of optimal portfolios, which all minimize the variance for given levels of desired return.

The efficient frontier is constructed by plotting total portfolio risk – the standard deviation between returns, along the x-axis. The corresponding y-coordinate is the expected portfolio return. Portfolios that lie along the efficient frontier curve represent portfolios with minimal risk, while portfolios that lie below are sub-optimal.

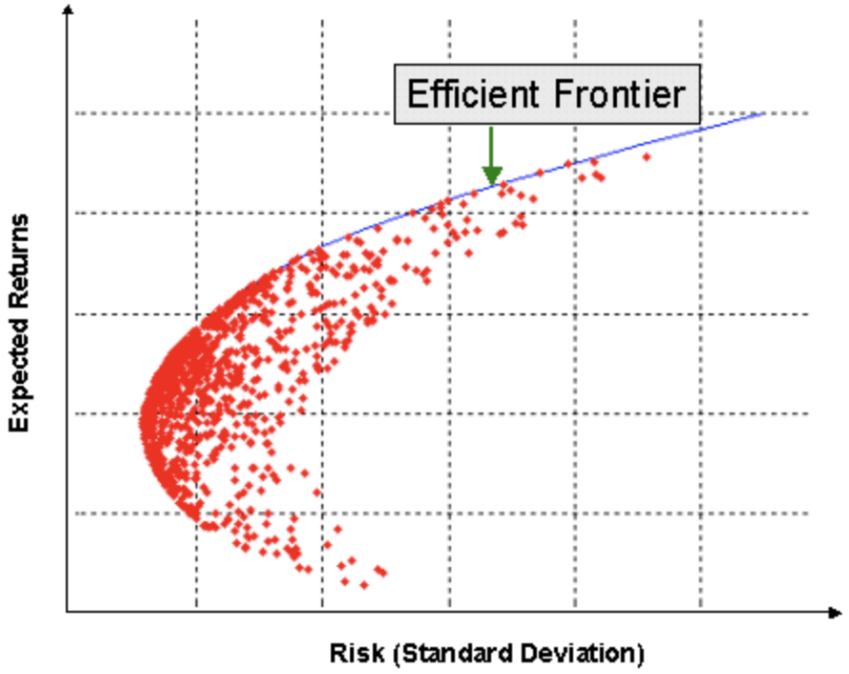


Figure 1: The Efficient Frontier

As we will see in the next section, calculating the volatility (standard deviation) of a 3-asset portfolio becomes lengthier, as we need to calculate the correlation between each pair of assets:

$$\sigma_p^2 = \sum_{i=1}^{3} x_i^2 \sigma_i^2 + \sum_{i=1}^{3} \sum_{j \neq i}^{3} x_i x_j \sigma_i \sigma_j p_{ij}$$

Where p_{ij} is the covariance between assets i and j

The standard deviation of the portfolio to be mapped along the x-axis of the efficient frontier is:

$$\sigma_p = \sqrt{\sigma_p^2}$$

Similarly, the returns to be mapped along the y-axis are calculated by the dot product between returns and asset allocations:

$$r = \langle \vec{x}^*, \vec{r} \rangle$$

3.3 Example

As an example, consider a portfolio of three assets: Apple (AAPL), Microsoft (MSFT), and Nike (NKE). We will calculate the expected returns for each company, via monthly close prices from 1/1/16 to 10/1/19.

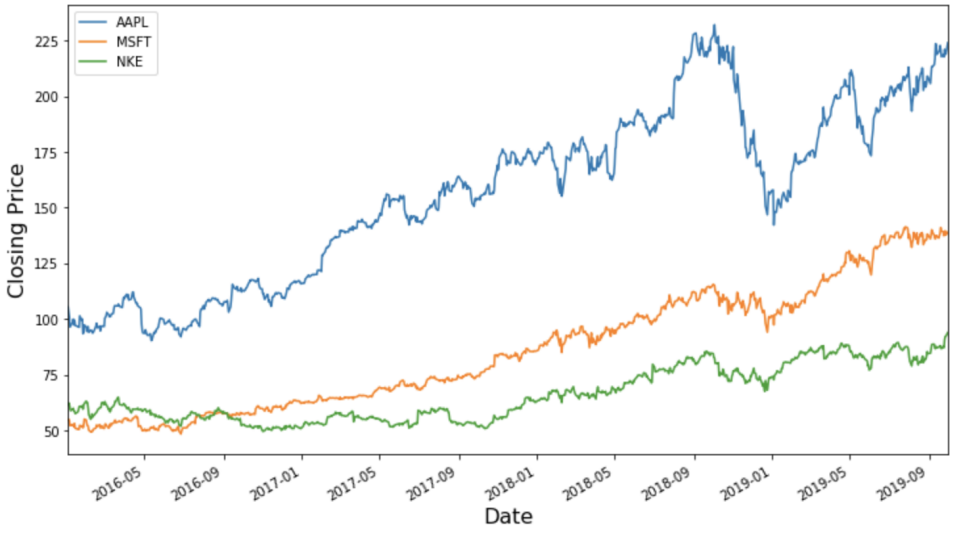


Figure 2: Monthly Closing Prices from 1/1/2016 to 10/1/2019

Using our portfolio of three assets, we can construct an efficient frontier for the portfolio by changing the values of r_{min}:

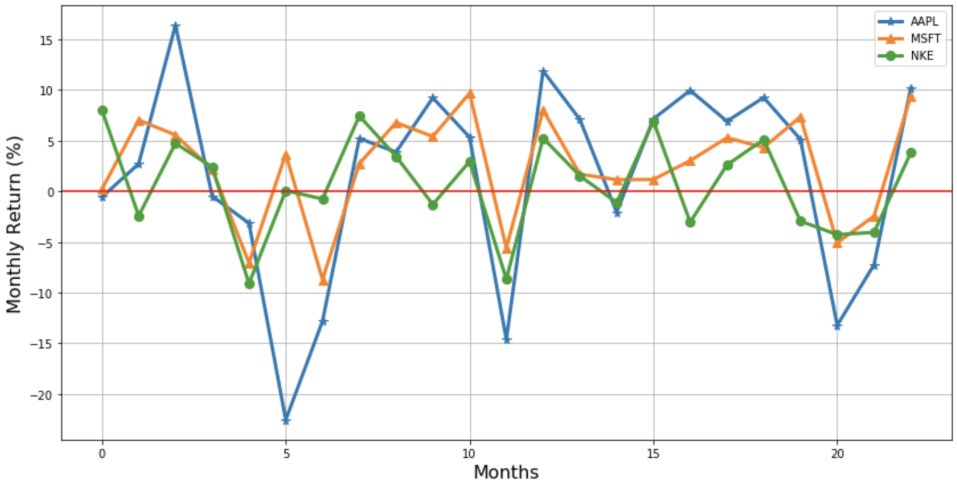


Figure 3: Monthly Returns from 1/1/2016 to 10/1/2019

Using the historical prices of the assets, we find the returns for each asset to be the following

```
Estimated Returns:
AAPL : [0.01464848]
MSFT : [0.02426521]
NKE : [0.00723904]
```

Figure 4: Estimated Returns

Next, we calculate the covariance matrix by using the estimated return values.

```
Covariance Matrix:
 [[0.00962834 0.00336847 0.00241048]
 [0.00336847 0.0027144  0.00129429]
 [0.00241048 0.00129429 0.00228191]]
```

Figure 5: Covariance Matrix

After we've solved the optimization for $\vec{x}^*$, we obtain the optimal portfolio distributions which minimize the expected risk.

```
Recommended Distribution:
AAPL : [-0.21449391]
MSFT : [0.88820909]
NKE : [0.32628482]

Expected Return:
0.02

Expected Risk:
0.05
```

Figure 6: Optimal Allocation of Assets

Note: Negative allocation values indicate short-sales – the process of selling assets that you do not own.

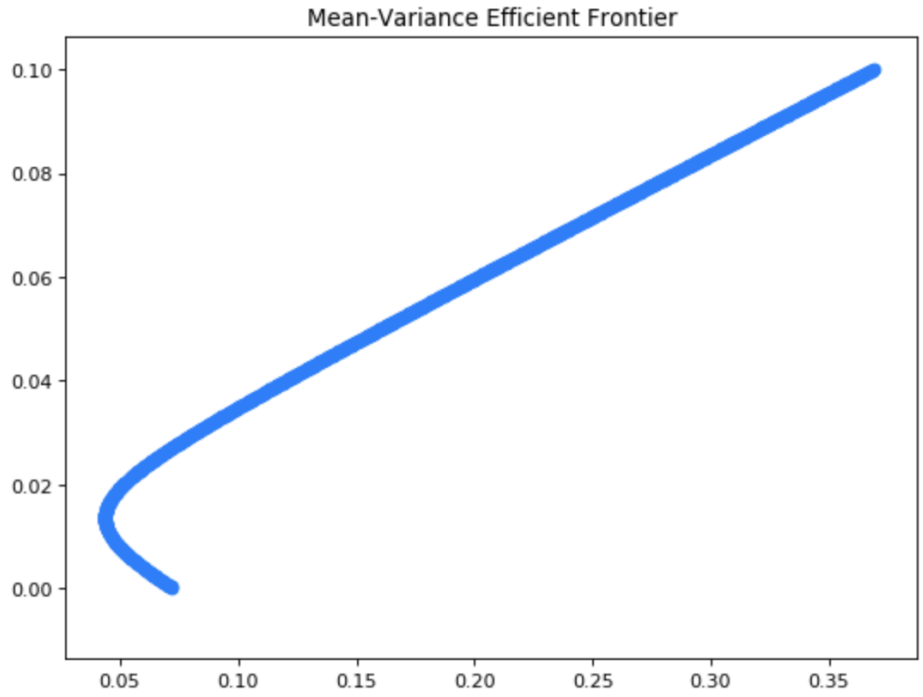


Figure 7: Efficient Frontier for AAPL, MSFT, and NKE

4 Conclusion

Mean-Variance optimization gives a rough approximation of the optimal asset allocation for a portfolio; however, the method still has its flaws. The solution to the mean-variance optimization problem involves the mean of historical asset returns as well as standard deviations. Some of the issues with this method is the over weighting of assets with high expected returns, as well as instability when it comes to errors or outliers in the data.

Extensions to the problem include methods such as the Sharpe ratio, which includes a risk-aversion factor, or the use of genetic algorithms which use repeated simulations to try and smooth the data. Thus, mean-variance optimization gives a good heuristic to the optimal portfolio; however, the problem could be improved by changing the risk/reward structure or by resampling methods such as genetic algorithms.

5 Appendix

5.1 Bibliography

References

[1] Harry Markowitz. Portfolio Selection. [*The Journal of Finance*]. 1952.

[2] Félix Roudier. Portfolio Optimization and Genetic Algorithms. [*Swiss Federal Institute of Technology*]. 2007.

[3] yfinance. Historical Stock Market Data. https://pypi.org/project/yfinance/. 2019.

5.2 Code

Portfolio Optimization
Name: John Furlong

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
import pandas as pd
from scipy import stats
%matplotlib inline
from matplotlib.pyplot import figure
import yfinance as yf
```

[2]:
```
# Download historical data using (<Stock Name>, <start_date>, <end_date>)
aapl_data = yf.download('AAPL','2016-01-01','2019-10-01')
msft_data = yf.download('MSFT', '2016-01-01', '2019-10-01')
nke_data = yf.download('NKE', '2016-01-01', '2019-10-01')
```

```
[*********************100%**********************]  1 of 1 downloaded
[*********************100%**********************]  1 of 1 downloaded
[*********************100%**********************]  1 of 1 downloaded
```

[209]:
```
class Portfolio:
    def __init__(self, stocks, start, end, period, interval):
        # self.stocks : Names of the stocks
        self.stocks = stocks
        self.start = start
        self.end = end
        self.period = period
        self.interval = interval
        # self.prices[i] : Used to plot prices
        self.prices1 = yf.download(self.stocks[0], start, end)
        self.prices2 = yf.download(self.stocks[1], start, end)
        self.prices3 = yf.download(self.stocks[2], start, end)
        # self.stock[i] : Used for calculations
        self.stock1 = yf.Ticker(self.stocks[0])
        self.stock2 = yf.Ticker(self.stocks[1])
        self.stock3 = yf.Ticker(self.stocks[2])
        # Compute Optimization Constants
```

```
        stock1 = self.stock1.history(self.period, self.interval).dropna()
        stock2 = self.stock2.history(self.period, self.interval).dropna()
        stock3 = self.stock3.history(self.period, self.interval).dropna()
        data = self.compute_roi(stock1, stock2, stock3)
        stock1 = np.asarray(data[self.stocks[0]])
        stock2 = np.asarray(data[self.stocks[1]])
        stock3 = np.asarray(data[self.stocks[2]])
        # Matrix of Returns (%) needed for covariance matrix
        roi_matrix = np.stack((stock1, stock2, stock3), axis = 0)
        self.r = np.asarray([np.mean(roi_matrix, axis = 1)]).T
        self.C = np.cov(roi_matrix)
        self.C_inv = np.linalg.inv(self.C)
        self.e = np.asarray([[1, 1, 1]]).T

    def plot_prices(self):
        plt.figure(figsize=(12,7))
        self.prices1.Close.plot(label=self.stocks[0])
        self.prices2.Close.plot(label=self.stocks[1])
        self.prices3.Close.plot(label=self.stocks[2])
        plt.ylabel('Closing Price', fontsize=16)
        plt.xlabel('Date', fontsize=16)
        plt.legend()
        plt.show()
        return

    def compute_roi(self, stock1, stock2, stock3):
        returns1 = []
        returns2 = []
        returns3 = []
        for i in range(1, 25):
            j = i - 1
            ret1 = (stock1.Close[i] - stock1.Close[j])/stock1.Close[i]
            ret2 = (stock2.Close[i] - stock2.Close[j])/stock2.Close[i]
            ret3 = (stock3.Close[i] - stock3.Close[j])/stock2.Close[i]
            returns1.append(ret1)
            returns2.append(ret2)
            returns3.append(ret3)
        data = pd.DataFrame({self.stocks[0] : returns1, self.stocks[1] :␣
↪returns2, self.stocks[2] : returns3})
        # Drop last row
        data.drop(data.tail(1).index,inplace=True)
        return data

    def plot_roi(self):
        stock1 = self.stock1.history(self.period, self.interval).dropna()
        stock2 = self.stock2.history(self.period, self.interval).dropna()
        stock3 = self.stock3.history(self.period, self.interval).dropna()
```

```
        data = self.compute_roi(stock1, stock2, stock3)
        # Plot the Monthly ROI %
        plt.figure(figsize=(14,7))
        plt.plot([i for i in range(0,23)], 100*data[self.stocks[0]], lw=3,␣
↪marker='*',markersize=8, label=self.stocks[0])
        plt.plot([i for i in range(0,23)], 100*data[self.stocks[1]], lw=3,␣
↪marker='^',markersize=8, label=self.stocks[1])
        plt.plot([i for i in range(0,23)], 100*data[self.stocks[2]], lw=3,␣
↪marker='o',markersize=8, label=self.stocks[2])
        plt.legend()
        plt.axhline(color='red')
        plt.xlabel('Months', fontsize=16)
        plt.ylabel('Monthly Return (%)', fontsize=16)
        plt.grid(True)
        plt.show()
        return

    def compute_risk(self, x_star, r, C):
        weights_times_deviations = [x_star[i]**2 * C[i][i] for i in range(0,3)]
        variance = sum(weights_times_deviations)
        for j in range(0, 3):
            for k in range(0, 3):
                if(j != k):
                    weight1 = x_star[j]
                    weight2 = x_star[k]
                    std1 = C[j][j]**0.5
                    std2 = C[k][k]**0.5
                    cov = C[j][k]
                    add_var = weight1 * weight2 * std1 * std2 * cov

                    variance += add_var
        standard_deviation = np.sqrt(variance)
        return standard_deviation

    def optimize(self, exp_ret):
        # Vector (Array) of expected returns
        r = self.r
        # C : Covariance Matrix, C_inv : Covariance Inverse
        C = self.C
        C_inv = self.C_inv
        # e : elementary vector of 1's
        e = self.e
        # Begin Solving Optimization Problem
        a = np.asscalar(r.T.dot(C_inv).dot(r))
        b = np.asscalar(r.T.dot(C_inv).dot(e))
        c = np.asscalar(e.T.dot(C_inv).dot(e))
        # Matrix A : used to solve for lambdas in the Lagrangian
```

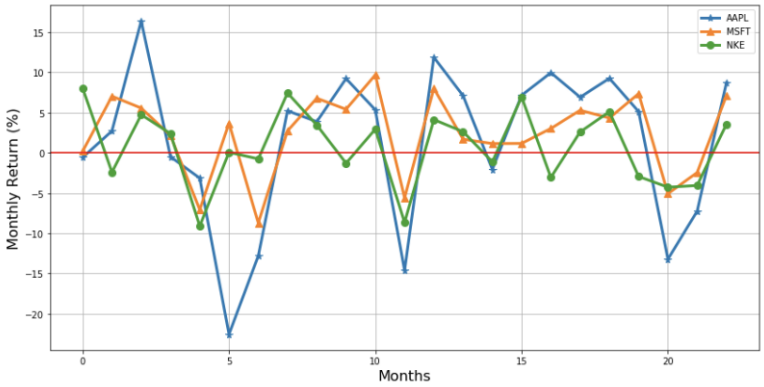
```
        A = np.array([[a, b], [b, c]])
        A_inv = np.linalg.inv(A)
        # Constraints
        constraint_vector = np.asarray([[exp_ret, 1]]).T
        # Solve for lambdas and convert from 1x1 "matrix" to a scalar
        lambdas = A_inv.dot(constraint_vector)
        lambda1 = np.asscalar(lambdas[0])
        lambda2 = np.asscalar(lambdas[1])
        # Substitute lambdas to solve for x_star
        x_star = (lambda1 * C_inv.dot(r)) + (lambda2 * C_inv.dot(e))
        # Expected Returns
        ret = x_star.T.dot(r)
        # Risk
        risk = self.compute_risk(x_star, r, C)
        return [x_star, r, C, risk]


    def plot_frontier(self):
        figure(num=None, figsize=(8, 6), dpi=80, facecolor='w', edgecolor='k')
        e = np.asarray([[1, 1, 1]]).T
        exp_ret = 0.0001

        for i in range(0, 1000):
            [weights, r, C, risk] = self.optimize(exp_ret)

            standard_deviation = risk
            expected_return = weights.T.dot(r)

            plt.title("Mean-Variance Efficient Frontier")
            plt.scatter(standard_deviation, expected_return, color='#007bff')

            exp_ret += 0.0001


        return
```

[210]:
```
# Initialize a Portfolio class
stocks = ['AAPL', 'MSFT', 'NKE']
start = '2016-01-01'
end = '2019-10-01'
period = "2y"
interval = "1mo"
portfolio = Portfolio(stocks, start, end, period, interval)
```

```
[**********************100%**********************]  1 of 1 downloaded
[**********************100%**********************]  1 of 1 downloaded
[**********************100%**********************]  1 of 1 downloaded
```

[211]:
```
portfolio.plot_prices()
```

[212]:
```
portfolio.plot_roi()
```

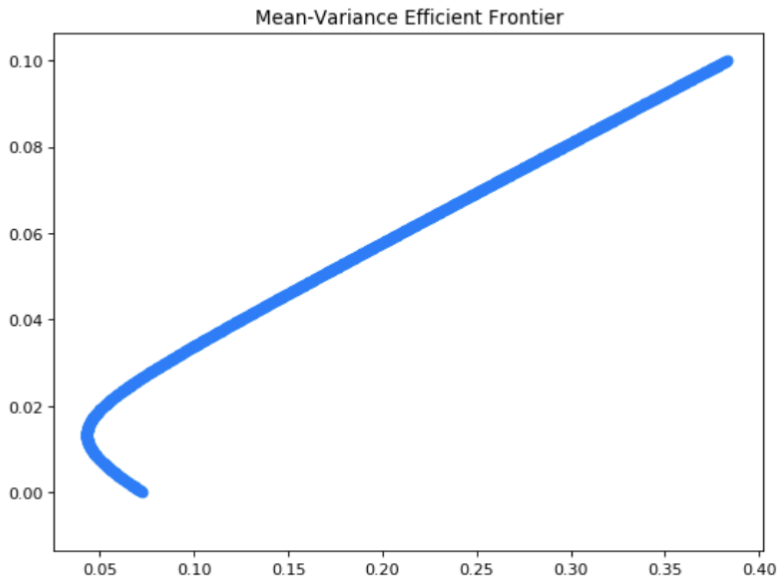


[213]:
```
x_star, r, C, risk = portfolio.optimize(exp_ret = 0.02)
print("Recommended Distribution:")
print(portfolio.stocks[0], ':', x_star[0])
print(portfolio.stocks[1], ':', x_star[1])
```

```
print(portfolio.stocks[2], ':', x_star[2], '\n')

ret = x_star.T.dot(r)
print("Expected Return: ")
print(round(np.asscalar(ret), 2), '\n')

print("Expected Risk: ")
print(round(np.asscalar(risk), 2))
```

```
Recommended Distribution:
AAPL : [-0.21449391]
MSFT : [0.88820909]
NKE : [0.32628482]

Expected Return:
0.02

Expected Risk:
0.05
```

[214]:
```
portfolio.plot_frontier()
```



[215]:
```
print("Covariance Matrix: \n", portfolio.C)
```

```
Covariance Matrix:
 [[0.00952038 0.00324639 0.00235684]
 [0.00324639 0.00259347 0.00122516]
 [0.00235684 0.00122516 0.00224856]]
```

[216]:
```
print("Estimated Returns: ")
print(portfolio.stocks[0], ':', portfolio.r[0])
print(portfolio.stocks[1], ':', portfolio.r[1])
print(portfolio.stocks[2], ':', portfolio.r[2])
```

```
Estimated Returns:
AAPL : [0.01400517]
MSFT : [0.02328152]
NKE : [0.00712619]
```