Introduction to mathematical finance and investment theory mandatory assignment

Orlando Closs
Student № 674785
STK-MAT3700

Exercise 1

Please note that the data used in exercise is from 13/09/2023.

See Appendix ex1 for the python script used in this exercise. Libraries used were pandas [1], matplotlib [2] and seaborn [3] in visualisation of the following graphs.

(a): Plotting time series

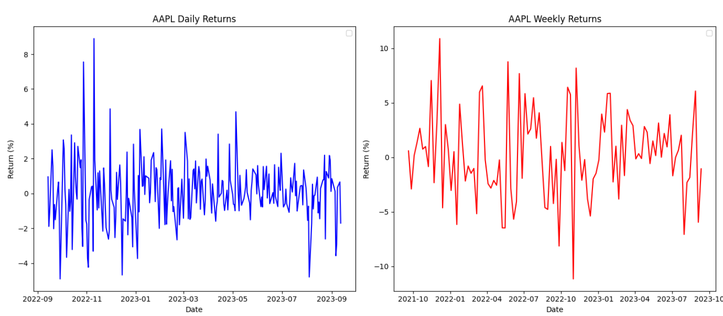

Figure 1: Time series to show Apple's (AAPL) weekly returns over two years and daily returns over one year (-13/09/2023).

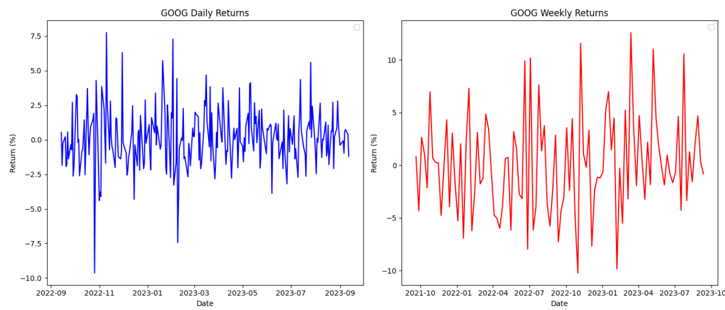

Figure 2: Time series to show Google's (GOOG) weekly returns over two years and daily returns over one year (-13/09/2023).

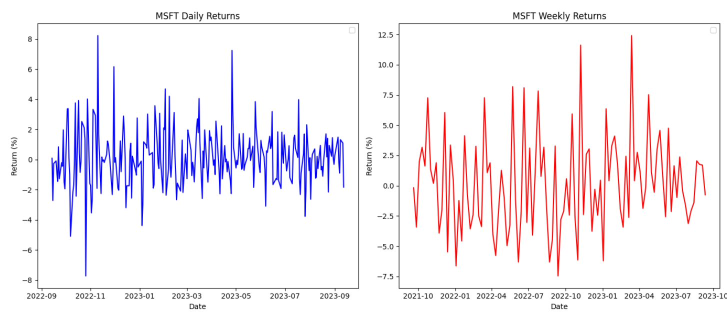

Figure 3: Time series to show Microsoft's (MSFT) weekly returns over two years and daily returns over one year (-13/09/2023).

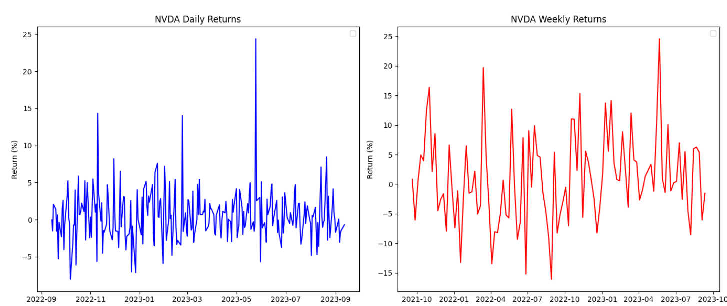

Figure 4: Time series to show NVIDIA's (NVDA) weekly returns over two years and daily returns over one year (-13/09/2023).

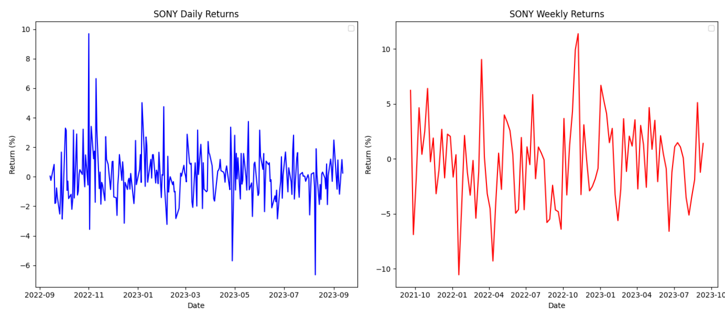

Figure 5: Time series to show Sony's (SONY) weekly returns over two years and daily returns over one year (-13/09/2023).

(b): Mean and volatility of returns

Stock Name	Type	Mean (%)	Volatility
AAPL	Daily	0.0703	1.78
AAPL	Weekly	0.259	3.97
GOOG	Daily	0.126	2.17
GOOG	Weekly	0.0718	4.72
MSFT	Daily	0.129	1.94
MSFT	Weekly	0.174	3.97
NVDA	Daily	0.551	3.48
NVDA	Weekly	0.958	7.4
SONY	Daily	0.0834	1.75
SONY	Weekly	-0.173	3.93

Figure 6: Table to show mean and volatility of the 5 selected assets for weekly returns over two years and daily returns over one year (-13/09/2023) these values are rounded to 3.s.f.

(c): Empirical density vs normal distribution

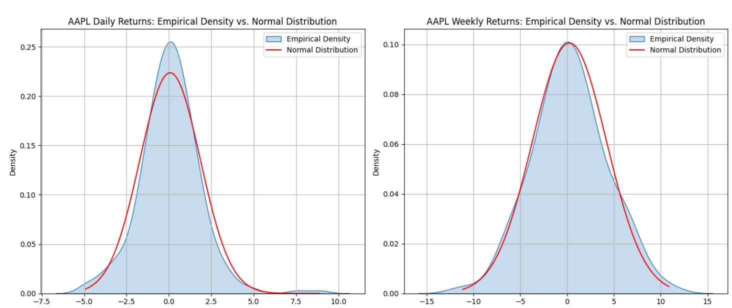

Figure 7: Graphs to show Apple's (AAPL) empirical density fitted with its normal distribution with weekly and daily returns.

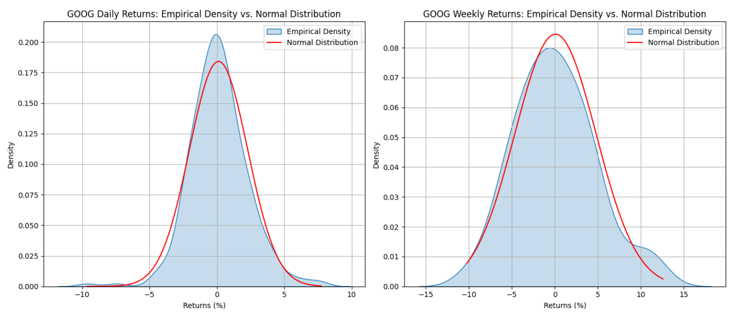

Figure 8: Graphs to show Google's (GOOG) empirical density fitted with its normal distribution with weekly and daily returns.

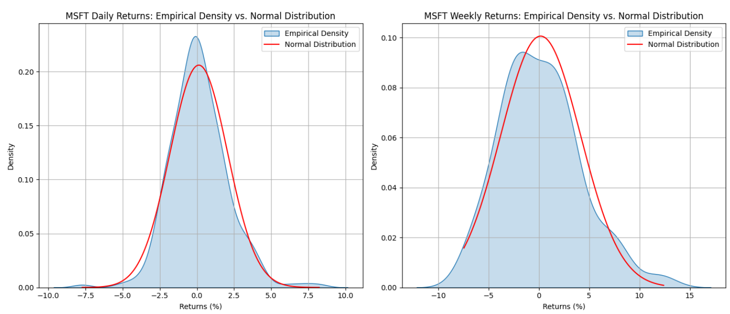

Figure 9: Graphs to show Microsoft's (MSFT) empirical density fitted with its normal distribution with weekly and daily returns.

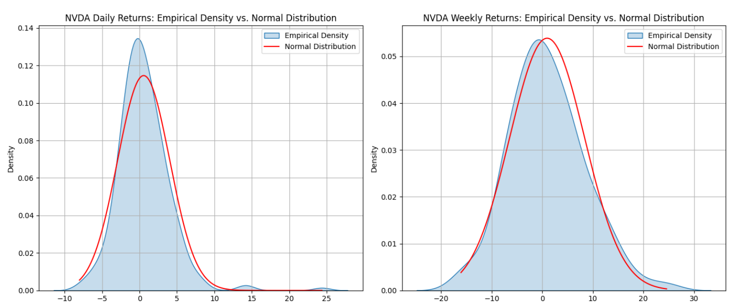

Figure 10: Graphs to show NVIDIA's (NVDA) empirical density fitted with its normal distribution with weekly and daily returns.

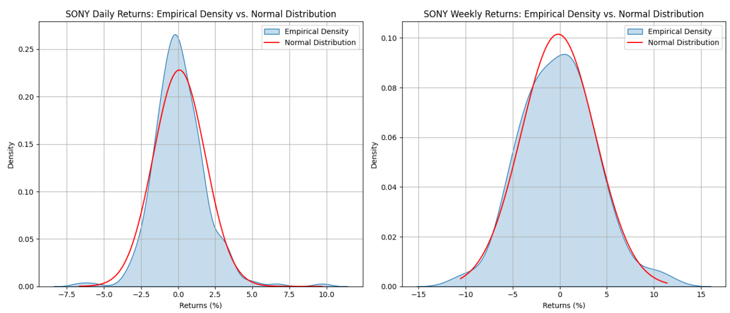

Figure 11: Graphs to show Sony's (SONY) empirical density fitted with its normal distribution with weekly and daily returns.

Discussion
The **normal distribution hypothesis** states that asset returns are normally distributed. Consistently the basic shape of our empirical density graphs shows similarity to its corresponding normal distribution; however, this shape is not perfect. One can observe heavier tails in the empirical density graphs in all the figures. This indicates that extreme returns are more frequent than what the normal distribution hypothesis would suggest. This is because the real-world financial market often involves shocks, news events and other factors that lead to more extreme movements than what would be expected under a normal distribution.

Additionally, the empirical distributions for daily returns tend to have sharper peaks than the normal distributions. These sharper peaks suggest that returns tend to cluster more closely around the mean. However, weekly returns show a more broadening peak suggesting that the daily average gets evened out over the week, due to factors like news events, making returns vary more broadly from the average.

Exercise 2

Please note that the data used in exercise is from 13/09/2023.

See Appendix ex2 for the python script used in this exercise.

(a): Variance-covariance matrices

The variance-covariance matrix $\mathbf{V}$ for n assets is given by:

$$\mathbf{V} = \begin{bmatrix} \sigma_1^2 & \rho_{12}\sigma_1\sigma_2 & \dots & \rho_{1n}\sigma_1\sigma_n \\ \rho_{21}\sigma_2\sigma_1 & \sigma_2^2 & \dots & \rho_{2n}\sigma_2\sigma_n \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{n1}\sigma_n\sigma_1 & \rho_{n2}\sigma_n\sigma_2 & \dots & \sigma_n^2 \end{bmatrix}$$

Where:

σ_i^2 = Variance of the returns of asset i

ρ_{ij} = Pearson correlation coefficient between the returns of assets i and j

σ_i = Standard deviation of the returns of asset i

This will be achieved in implementation using the Python library panda's covariance function .cov() [1].

	AAPL	GOOG	MSFT
AAPL	15.733255	11.694736	11.234691
GOOG	11.694736	22.276352	13.308659
MSFT	11.234691	13.308659	15.731894

(1)

	AAPL	GOOG	MSFT	NVDA
AAPL	15.733255	11.694736	11.234691	17.597943
GOOG	11.694736	22.276352	13.308659	19.053782
MSFT	11.234691	13.308659	15.731894	20.372411
NVDA	17.597943	19.053782	20.372411	54.811841

(2)

	AAPL	GOOG	MSFT	NVDA	SONY
AAPL	15.733255	11.694736	11.234691	17.597943	6.628200
GOOG	11.694736	22.276352	13.308659	19.053782	7.470190
MSFT	11.234691	13.308659	15.731894	20.372411	6.837558
NVDA	17.597943	19.053782	20.372411	54.811841	17.906442
SONY	6.628200	7.470190	6.837558	17.906442	15.453067

(3)

(b): Minimum variance portfolio

The primary constraints of the Markowitz portfolio theory can be expressed as:

$$\mathbf{x}^{*T}\mathbf{1} = 1 \tag{4}$$

$$\mathbf{x}^{*T}\mathbf{r} = r_p \tag{5}$$

Where:

- $\mathbf{x}^*$ is a vector of portfolio weights.
- $\mathbf{1}$ is a vector of ones.
- $\mathbf{r}$ is a vector of expected asset returns.
- r_p is the expected portfolio return.

The variance of the portfolio, σ_p^2, can be expressed in terms of the portfolio weights and the variance-covariance matrix $\mathbf{V}$ of asset returns:

$$\sigma_p^2 = \mathbf{x}^{*T}\mathbf{V}\mathbf{x}^* \tag{6}$$

To find the minimum-variance portfolio, we seek to minimize σ_p^2 with respect to the portfolio weights $\mathbf{x}^*$, subject to the constraints (4) and (5). In this scenario a Python implementation is used, utilising theoretical formula to compute the weights, expected return, and volatility of the minimum variance portfolio.

$$\sigma_m = \frac{1}{\sqrt{c}} \tag{7}$$

$$\mathrm{r}_m = \frac{a}{c} \tag{8}$$

$$\mathbf{x}^*{}_m = \frac{1}{c}\mathbf{V^{-1}1} \tag{9}$$

Where:

$$a = \mathbf{r^T V^{-1} 1} \tag{10}$$

$$c = \mathbf{1^T V^{-1} 1} \tag{11}$$

Where:

- σ_m is the volatility of the minimum variance portfolio.
- r_m is the return of the minimum variance portfolio.
- $\mathbf{x}^*{}_m$ is the weights of the minimum variance portfolio.
- $\mathbf{V^{-1}}$ is the inverse of the variance-covariance matrix.

3-Asset Portfolio (AAPL, GOOG, MSFT)

- **Weights**:

$$\text{AAPL} : 46.99\%$$
$$\text{GOOG} : 9.38\%$$
$$\text{MSFT} : 43.63\%$$

- **Expected Portfolio Return**: 0.204%
- **Portfolio Volatility**: 3.659%

4-Asset Portfolio (AAPL, GOOG, MSFT, NVDA)

- **Weights**:

$$\text{AAPL} : 50.28\%$$
$$\text{GOOG} : 10.38\%$$
$$\text{MSFT} : 58.21\%$$
$$\text{NVDA} : -18.86\%$$

- **Expected Portfolio Return**: 0.058%
- **Portfolio Volatility**: 3.513%

5-Asset Portfolio (AAPL, GOOG, MSFT, NVDA, SONY)

- **Weights**:

$$\text{AAPL} : 29.12\%$$
$$\text{GOOG} : 2.25\%$$
$$\text{MSFT} : 43.69\%$$
$$\text{NVDA} : -28.97\%$$
$$\text{SONY} : 53.91\%$$

- **Expected Portfolio Return**: -0.217%
- **Portfolio Volatility**: 2.868%

Discussion

Apple (AAPL) and Microsoft (MSFT) stand out as optimal for the minimum-variance portfolio and consistently make up a significant portion of the portfolios. From the variance-covariance matrix (3) we can see that AAPL and MSFT have a relatively low covariance and therefore are a beneficial combination as they do not move strictly in tandem. This balance means if one is under-performing the other may offset its losses which is key to the minimum variance portfolio.

Noteworthy is Nvidia's (NVDA) introduction to the portfolio leading to it taking short positions in a minimum variance portfolio. Its variance is significantly high and has high covariances with the other assets (3) therefore moves often in tandem with the other stocks. This is because it manufactures computer GPU's for technology and artificial intelligence companies and therefore is somewhat reliant on the success of technology companies. It is high risk with high variance and has high covariance with the other assets which is why the minimum-variance portfolio does not favour it.

Sony (SONY) gained a large portion and was of high importance when introduced to the portfolio. This is because it has a very low covariance with AAPL, GOOG and MSFT (3) and therefore is a diverse stock to have in a minimum-variance portfolio which has the potential to offset losses.

Google (GOOG) does not gain a large portion throughout the different asset combinations. This can be attributed to Google's relatively high variance and covariances with other assets. These factors diminished its significance in a minimum variance portfolio.

In the 5-asset portfolio, NVDA's large short position (-28.96 %) in order to acheive minimal variance leads to a negative portfolio expected return of -0.217 %. This is due to NVDA's significantly high expected return of 0.958 %.

(c): Efficient portfolio frontier

To construct the efficient frontier, we find the portfolios of minimum risk for a given level of expected return. This involves iterating through a range of target returns and determining the corresponding minimum variance for each. For each asset combination, the maximum target return is set to the highest expected return among the assets. Using Python, we implement the theoretical formulas to compute and plot [2] the efficient frontier.

Given a target return r:

$$\sigma^2(r) = c\left(\frac{(r-\frac{a}{c})^2}{bc-a^2}\right) + \frac{1}{c} \tag{12}$$

Where:

$$b = \mathbf{r^T V^{-1} r} \tag{13}$$

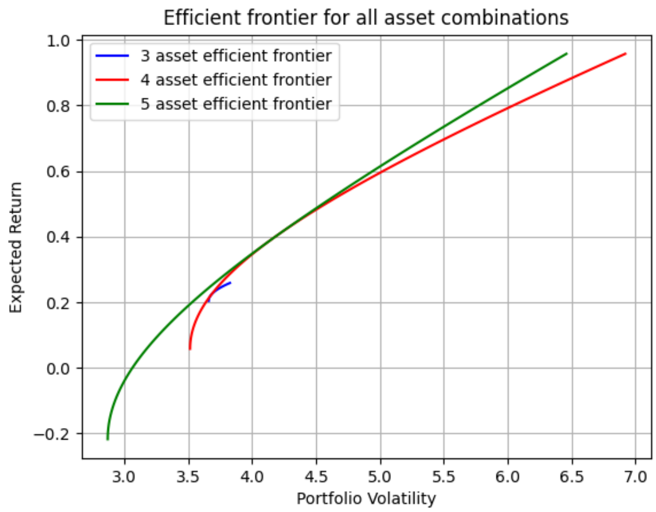

Figure 12: Graph to show 3, 4 and 5-asset efficient portfolio frontiers.

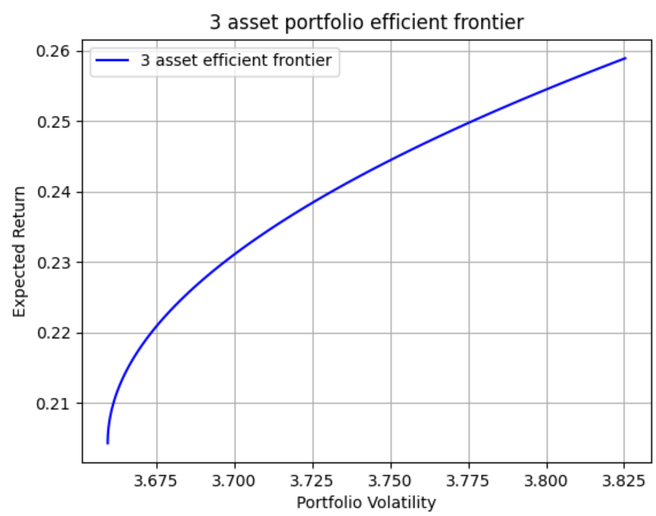

Figure 13: Graph to show the 3-asset efficient portfolio frontier.

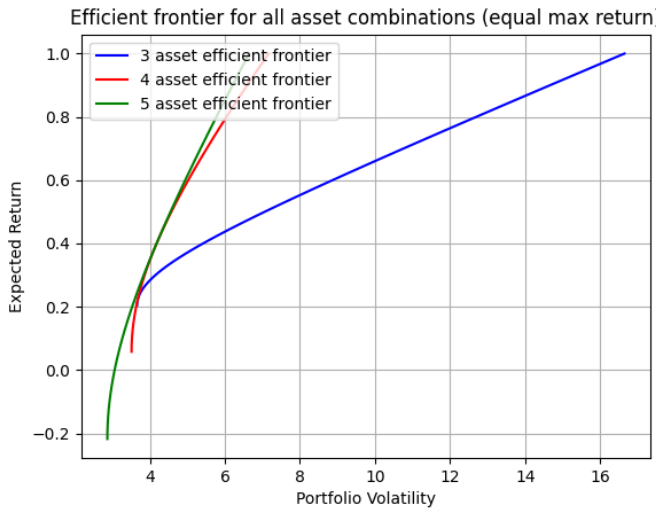

Figure 14: Graph to show 3, 4 and 5-asset efficient portfolio frontiers with equal max returns.

Discussion

When all frontiers are plotted on the same graph, with the maximum target return set to the maximum of the assets expected returns, the 3-asset frontier occupies a relatively small portion. This is primarily due to the scale being influenced by the significantly high return and risk introduced by NVDA in the 4 and 5-asset portfolios.

The efficient frontier for the 3-asset portfolio displays a convex curve. Starting from the leftmost point, which represents the minimum risk, the curve rises steeply, showing rapid gains in expected return for small increases in risk. As we move further to the right, the curve begins to flatten, suggesting that additional returns require taking on proportionally more risk.

This description is also the case for the 4 and 5-asset efficient frontier's. However, these curves seem to flatten at a lesser rate suggesting less risk is needed for the excess return (above 0.26 %) in comparison to the 3-asset portfolio. This can be seen in Figure 14 when maximum returns are equal for all asset combinations. This happens because the 3-asset portfolio does not contain the anomalous high return asset NVDA.

When comparing the 4-asset to the 5-asset frontier, the 4-asset portfolio shows slightly higher returns for each level of risk across the curve. This can be attributed to NVDA's pronounced impact in the 4-asset mix. Given that NVDA carries both high risks and high returns, its influence is dominant. When an additional asset is introduced in the 5-asset portfolio, NVDA's contribution is reduced, leading to slightly diluted returns for the same levels of risk.

(d): Tangent portfolio

The tangent portfolio, resides on the efficient frontier and represents the portfolio with the highest Sharpe ratio [4] defined as:

$$\text{Sharpe Ratio} = \frac{r_p - r_f}{\sigma_p} \tag{14}$$

Where:

- r_p is the expected return of the optimal portfolio.
- r_f is the risk-free rate.
- σ_p is the or volatility of the optimal portfolio.

In our Python implementation, the maximal Sharpe ratio from the points of the efficient frontier is derived. The theoretical formula used to determine the weights of the assets in the tangent portfolio given its optimal return is:

$$\mathbf{x}^* = \frac{\mathbf{V^{-1}}(\mathbf{r} - r_f\mathbf{1})}{\mathbf{1^T V^{-1}}(\mathbf{r} - r_f\mathbf{1})} \tag{15}$$

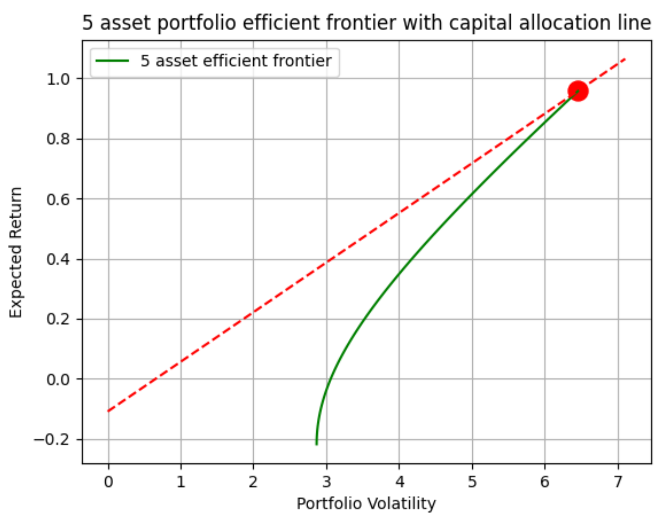

Figure 15: Graph to show the 5-asset efficient portfolio frontier with tangent portfolio and capital allocation line.

Optimal 5-Asset Portfolio (AAPL, GOOG, MSFT, NVDA, SONY)

- **Weights**:

$$\begin{aligned}
\text{AAPL} &: 94.62\% \\
\text{GOOG} &: -21.42\% \\
\text{MSFT} &: 10.54\% \\
\text{NVDA} &: 65.27\% \\
\text{SONY} &: -49.02\%
\end{aligned}$$

- **Expected Portfolio Return**: 0.957%
- **Portfolio Volatility**: 6.459%

Discussion

By optimally mixing a bank investment with a Markowitz portfolio, you can achieve a range that spans from the low risk and return profile of the bank investment to the higher risk and return profile of the Markowitz portfolio. However in this scenario for the 5 asset portfolio the optimal portfolio simply chooses the largest risk-return portfolio on the efficient frontier.

This is due to the anomalous high return NVDA stock, who's return is set to the efficient frontier's maximum value. The efficient frontier has minimised risk on NVDA's return by diversifying its portfolio with short positions and a large long position on AAPL and this portfolio in turn has the maximal sharpe ratio.

Additionally the risk-free rate (r_m / 2) in this case is negative. This means that even doing nothing has a small cost. Because of this, the extra return from our portfolio looks even bigger. So when we're calculating the Sharpe ratio, portfolios with large returns become even more attractive.

Exercise 3

Please note that the data used in this exercise is from 08/10/2023.

See Appendix ex3 for the python script used in this exercise.

(a): Black Scholes calculator for 6 options

The risk free rate used in this exercise is 3.4% this is the average risk free rate in 2023 in Norway, taken from Statista [6].

For different combinations of strike and exercise time, and using the stock price and volatility from 08/10/2023 for AAPL; these values are entered into the Black Scholes formula:

$$
\begin{aligned}
C &= S_t \cdot \Phi(d_1) - K \cdot e^{-rT} \cdot \Phi(d_2) \\
d_1 &= \frac{\ln\left(\frac{S_t}{K}\right) + \left(r - \frac{1}{2}\sigma^2\right) \cdot T}{\sigma\sqrt{T}} \\
d_2 &= d_1 - \sigma\sqrt{T}
\end{aligned}
$$

Where:

- C : call option price
- S_t : current stock price
- K : strike price
- r : risk-free rate
- σ : volatility
- T : time to expiration
- $\Phi()$: cumulative distribution function for a standard normal distribution

AAPL 6 Call Option Prices in USD with different combinations of strike and time period

	1 Month	3 Months
Current Stock Price	**5.76**	**10.22**
Current Stock Price + 10%	**0.84**	**3.88**
Current Stock Price - 10%	**18.69**	**21.57**

Table 1: Black-Scholes Calculated Call Option Prices for AAPL

(b): Market price vs Black-Scholes price

The risk free rate used in this exercise is the same as part (a) 3.4%. The options selected were from 08/10/2023 with an exercise time on 10/11/2023 totalling 33 days.

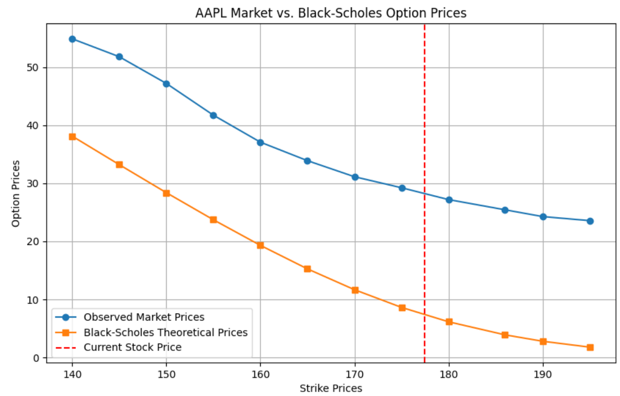

Figure 16: Graph to show market option price vs Black Scholes option price for AAPL.

(c): Plotting implied volatility as a function of strike price

The ready made routine used to find implied volatility is from the python library: py_vollib [5].

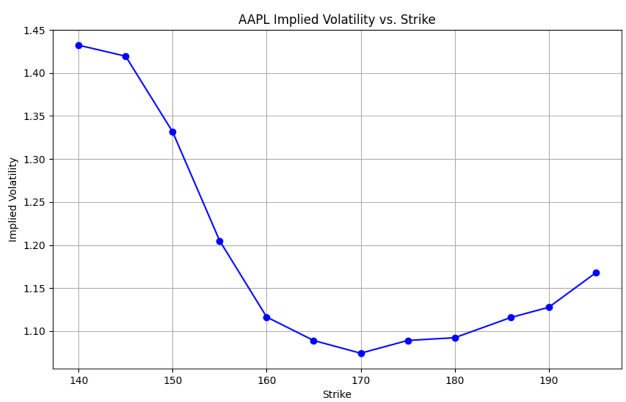

Figure 17: Graph to show implied volatility against strike price for AAPL options.

References

[1] Pandas Documentation. *pandas.DataFrame.cov.* https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.cov.html

[2] Matplotlib Documentation. *matplotlib.pyplot.* https://matplotlib.org/3.5.3/api/_as_gen/matplotlib.pyplot.html

[3] Seaborn Documentation. *seaborn.kdeplot.* https://seaborn.pydata.org/generated/seaborn.kdeplot.html

[4] Corporate Finance Institute. *Capital Allocation Line (CAL) and Optimal Portfolio.* https://corporatefinanceinstitute.com/resources/career-map/sell-side/capital-markets/capital-allocation-line-cal-and-optimal-portfolio/

[5] Python Vollib Library Documentation. *py_vollib implied volatility.* https://vollib.org/documentation/1.0.3/autoapi/py_vollib/black_scholes/implied_volatility/index.html#module-py_vollib.black_scholes.implied_volatility

[6] Statista. *Average risk free rate in europe..* https://www.statista.com/statistics/885915/average-risk-free-rate-europe/

Appendix

ex1

```
'''
Author: Orlando Closs
Description: Code to calculate returns, mean and volatility (with table), plot time series
             and plot empirical vs normal graph
Date: 14/09/2023
'''


import pandas as pd
import matplotlib.pyplot as plt
import sigfig
import seaborn as sns
import numpy as np
from scipy.stats import norm


data_list=['AAPL-daily.csv', 'AAPL-weekly.csv', 'GOOG-daily.csv', \
           'GOOG-weekly.csv', 'MSFT-daily.csv', 'MSFT-weekly.csv', \
```

```
             'NVDA-daily.csv', 'NVDA-weekly.csv', 'SONY-daily.csv', 'SONY-weekly.csv']

def compute_returns(file_path):
    data = pd.read_csv(file_path) #read csv file
    data['Date'] = pd.to_datetime(data['Date']) #change date format
    data = data.sort_values(by='Date').reset_index(drop=True) #sorts list by date
    data['Returns'] = data['Close'].pct_change() *100 #makes new column in data and \
    #pct_change calulates percentage change
    return data

def plot_time_series(data, weekly_data, stock_name):
    #plots daily returns over time
    plt.figure(figsize=(14, 6))

    plt.subplot(1, 2, 1)
    title='{} Daily Returns'.format(stock_name)
    plt.plot(data['Date'], data['Returns'], color='blue')
    plt.title(title)
    plt.xlabel('Date')
    plt.ylabel('Return (%)')
    plt.legend()

    plt.subplot(1, 2, 2)
    title='{} Weekly Returns'.format(stock_name)
    plt.plot(weekly_data['Date'], weekly_data['Returns'], color='red')
    plt.title(title)
    plt.xlabel('Date')
    plt.ylabel('Return (%)')
    plt.legend()

    plt.tight_layout()

    # save the plot
    filename = "{}_returns.png".format(stock_name)
    plt.savefig(filename)

def plot_empirical_vs_normal(data, weekly_data, stock_name, daily_mean, \
                             daily_volatility, weekly_mean, weekly_volatility):
    daily_returns = data['Returns'].dropna() #gets returns data drops missing values
    weekly_returns = weekly_data['Returns'].dropna()

    plt.figure(figsize=(14, 6))
```

```

    plt.subplot(1, 2, 1)
    sns.kdeplot(daily_returns, label="Empirical Density", shade=True) #makes empirical \
    #density graph https://seaborn.pydata.org/generated/seaborn.kdeplot.html
    x_daily = np.linspace(daily_returns.min(), daily_returns.max(),\
                          1000) #empty data for x axis for normal distribution
    plt.plot(x_daily, norm.pdf(x_daily, daily_mean, daily_volatility), \
             'r-', label="Normal Distribution") #plots normal distribution
    plt.title(f"{stock_name} Daily Returns: Empirical Density vs. Normal Distribution")
    plt.xlabel("Returns (%)")
    plt.ylabel("Density")
    plt.legend()
    plt.grid(True) #adds gridlines - useful for this type of graph

    plt.subplot(1, 2, 2)
    sns.kdeplot(weekly_returns, label="Empirical Density", shade=True)
    x_weekly = np.linspace(weekly_returns.min(), weekly_returns.max(), 1000)
    plt.plot(x_weekly, norm.pdf(x_weekly, weekly_mean, weekly_volatility), 'r-', label="Normal Distribution")
    plt.title(f"{stock_name} Weekly Returns: Empirical Density vs. Normal Distribution")
    plt.xlabel("Returns (%)")
    plt.ylabel("Density")
    plt.legend()
    plt.grid(True)

    #plots these two graphs in one image
    plt.tight_layout()
    filename = f"{stock_name}_empirical_normal.png"
    plt.savefig(filename)

def mean_and_volatility(data):
    mean = data['Returns'].mean()
    volatility = data['Returns'].std()
    return mean, volatility


# empty table to store results
mean_volatility_table = pd.DataFrame(columns=['Stock Name',
```

```python
    'Type', 'Mean (%)', 'Volatility'])

#-------------------------perform actions
    -------------------------

for index,csv in enumerate(data_list):

    if (index%2==0): #every other file
        stock_name=csv[0:4] #first four letters
        daily_data=compute_returns(csv)
        weekly_data=compute_returns(data_list[index+1])
        plot_time_series(daily_data, weekly_data, stock_name
            )

        daily_mean, daily_volatility = mean_and_volatility(
            daily_data)
        weekly_mean, weekly_volatility = mean_and_volatility
            (weekly_data)

        plot_empirical_vs_normal(daily_data, weekly_data,
            stock_name, daily_mean,\
                                  daily_volatility,
                                      weekly_mean,
                                      weekly_volatility)

        daily_mean=sigfig.round(daily_mean,3) #round to 3
            significant figures
        daily_volatility=sigfig.round(daily_volatility,3)
        weekly_mean=sigfig.round(weekly_mean,3)
        weekly_volatility=sigfig.round(weekly_volatility,3)

        # add daily results to table
        index2 = len(mean_volatility_table)
        mean_volatility_table.loc[index2] = [stock_name, '
            Daily', daily_mean, daily_volatility]

        # add weekly results to table
        index2 = len(mean_volatility_table)
        mean_volatility_table.loc[index2] = [stock_name, '
            Weekly', weekly_mean, weekly_volatility]


#-------------------------make mean tables
    -------------------------

#makes table plot and saves table image
fig, ax = plt.subplots(figsize=(10, 4))
ax.axis('off')
ax.axis('tight')
ax.table(cellText=mean_volatility_table.values, colLabels=
```

```
mean_volatility_table.columns,\
            cellLoc = 'center', loc='center')
plt.savefig('mean_volatility_table.png')
plt.close()
```

ex2

```
'''
Author: Orlando Closs
Description: Code to calculate markowitz minimum variance, efficient frontier and tangent portfolio using
    theoretical formula
Date: 03/10/2023
'''



import pandas as pd
import matplotlib.pyplot as plt
import sigfig
import seaborn as sns
import numpy as np
from scipy.stats import norm
from scipy.optimize import minimize

#-------------------------markowitz class-------------------------

class MarkowitzPortfolio():
    def __init__(self, expected_returns, cov):
        self.expected_returns = expected_returns
        self.n = len(expected_returns)
        self.cov = cov
        self.incov = np.linalg.inv(cov)  # Inverse of the covariance matrix
        self.a = None
        self.b = None
        self.c = None
        self.ones_vector = np.ones(self.n)
        self.compute_abc()

    def compute_abc(self):
        self.a = self.expected_returns.T @ self.incov @ self.ones_vector
        self.b = self.expected_returns.T @ self.incov @ self.expected_returns
        self.c = self.ones_vector.T @ self.incov @ self.ones_vector

```

```python
    def minimum_variance(self):
        volatility = 1 / np.sqrt(self.c)
        return_mv = self.a / self.c
        weights = (1/self.c) * (self.incov @ self.ones_vector)
        self.return_mv = return_mv
        self.risk_mv = volatility
        self.r0 = return_mv/2
        return volatility, return_mv, weights

    def optimal_risk_formula(self, r):
        variance = (self.c * (((r - (self.a/self.c))**2) / (self.b*self.c - (self.a**2)))) + (1 / self.c)
        return variance

    def efficient_frontier(self, num_points=100):
        min_return = self.return_mv
        max_return = max(self.expected_returns) + 0.2
        target_returns = np.linspace(min_return, max_return, num_points)
        portfolio_volatilities = []
        for target_return in target_returns:
            variance = self.optimal_risk_formula(target_return)
            if variance is not None:
                portfolio_volatilities.append(np.sqrt(variance))
            else:
                break
        return (portfolio_volatilities, target_returns)

    def tangent_portfolio(self):
        portfolio_volatilities, target_returns = self.efficient_frontier()
        max_sharpe_ratio=float('-Inf')
        optimal_p_return=0
        optimal_p_risk=0
        for index, target_return in enumerate(target_returns):
            volatility = portfolio_volatilities[index]
            sharpe_ratio=(target_return-self.r0)/(volatility)
            if sharpe_ratio > max_sharpe_ratio:
                max_sharpe_ratio = sharpe_ratio
                optimal_p_return = target_return
                optimal_p_risk = volatility

        return optimal_p_return, optimal_p_risk

    def get_weights_for_return(self,r):
```

```
        x0 = (self.b - self.a * r) / (self.b * self.c - self.a ** 2)
        xr = (self.c * r - self.a) / (self.b * self.c - self.a ** 2)
        weights = self.incov @ (x0 * self.ones_vector + xr * self.expected_returns)
        return weights

    def plot_efficient_frontier(self, color = 'blue'):
        portfolio_volatilities, target_returns = self.efficient_frontier()
        plt.plot(portfolio_volatilities, target_returns[:len(portfolio_volatilities)], '-', color=color, label='{} asset efficient frontier'.format(self.n))
        plt.xlabel('Portfolio Volatility')
        plt.ylabel('Expected Return')
        title = '{} asset portfolio efficient frontier'.format(self.n)
        plt.title(title)
        plt.legend(loc='upper left')
        plt.grid(True)

    def plot_tangent_portfolio(self):
        optimal_p_return, optimal_p_risk = self.tangent_portfolio()
        weights = [0, 1, 1.1]
        x_values = []
        y_values = []

        for weight in weights:
            y = (weight * optimal_p_return) + ((1 - weight) * self.r0)
            x = weight * optimal_p_risk
            x_values.append(x)
            y_values.append(y)

        title = '{} asset portfolio efficient frontier with capital allocation line'.format(self.n)
        plt.title(title)

       # Highlight the tangent portfolio with a red dot at weight 1
        plt.scatter(optimal_p_risk, optimal_p_return, color='red', s=150, label='Tangent Portfolio')

        # Connect the points at the ends with a line to form the Capital Allocation Line
        plt.plot([x_values[0], x_values[2]], [y_values[0], y_values[2]], 'r--', label='Capital Allocation
```

```
            Line')


#-------------------------preprocessing data -------------------------

data_list=['AAPL-weekly.csv', 'GOOG-weekly.csv', 'MSFT-weekly.csv', \
           'NVDA-weekly.csv',  'SONY-weekly.csv']

def compute_returns(file_path):
    data = pd.read_csv(file_path) #read csv file
    data['Date'] = pd.to_datetime(data['Date']) #change date format
    data = data.sort_values(by='Date').reset_index(drop=True) #sorts list by date
    data['Returns'] = data['Close'].pct_change() *100 #makes new column in data and \
    #pct_change calulates percentage change
    return data

#make dictionary and add returns to prepare for dataframe
asset_returns={}
for csv in data_list:
    data=compute_returns(csv)
    stock_name=csv[0:4]
    asset_returns[stock_name] = data['Returns']

returns_dataframe = pd.DataFrame(asset_returns)

#get covariance matrices
cov_matrix_3 = returns_dataframe[['AAPL','GOOG', 'MSFT']].cov()
#covariance matrix function
# https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.cov.html
cov_matrix_4 = returns_dataframe[['AAPL', 'GOOG', 'MSFT', 'NVDA']].cov()
cov_matrix_5 = returns_dataframe[['AAPL', 'GOOG', 'MSFT', 'NVDA', 'SONY']].cov()

expected_returns_3 = returns_dataframe[['AAPL','GOOG', 'MSFT']].mean(axis=0).values #more direct way calculating mean values
expected_returns_4 = returns_dataframe[['AAPL', 'GOOG', 'MSFT', 'NVDA']].mean(axis=0).values
expected_returns_5 = returns_dataframe[['AAPL', 'GOOG', 'MSFT', 'NVDA', 'SONY']].mean(axis=0).values

cov_matrix_3 = cov_matrix_3.values #grabs values ready for
```

```
    calculation
cov_matrix_4 = cov_matrix_4.values
cov_matrix_5 = cov_matrix_5.values

#-------------------------perform actions -------------------------

three_asset = MarkowitzPortfolio(expected_returns_3, cov_matrix_3)
four_asset = MarkowitzPortfolio(expected_returns_4, cov_matrix_4)
five_asset = MarkowitzPortfolio(expected_returns_5, cov_matrix_5)

#three asset minimum variance and efficient frontier

volatility_mv_3, return_mv_3, weights_mv_3 = three_asset.minimum_variance()
print('\n-----MINIMUM VARIANCE 3 ASSET-----')
print('\nAAPL, GOOG, MSFT')
print('WEIGHTS: {}, {}, {}'.format(weights_mv_3[0], weights_mv_3[1], weights_mv_3[2]))
print('EXPECTED RETURN: {}'.format(return_mv_3))
print('VOLATILITY: {}'.format(volatility_mv_3))

three_asset.plot_efficient_frontier()
plt.savefig('efficient_frontier_3.png')

plt.clf()

#four asset minimum variance and efficient frontier

volatility_mv_4, return_mv_4, weights_mv_4 = four_asset.minimum_variance()
print('\n-----MINIMUM VARIANCE 4 ASSET-----')
print('\nAAPL, GOOG, MSFT, NVDA')
print('WEIGHTS: {}, {}, {}, {}'.format(weights_mv_4[0], weights_mv_4[1], weights_mv_4[2], weights_mv_4[3]))
print('EXPECTED RETURN: {}'.format(return_mv_4))
print('VOLATILITY: {}'.format(volatility_mv_4))

four_asset.plot_efficient_frontier(color='red')
plt.savefig('efficient_frontier_4.png')

plt.clf()

#five asset minimum variance and efficient frontier

volatility_mv_5, return_mv_5, weights_mv_5 = five_asset.minimum_variance()
```

```
print('\n-----MINIMUM VARIANCE 5 ASSET-----')
print('\nAAPL, GOOG, MSFT, NVDA, SONY')
print('WEIGHTS: {}, {}, {}, {}, {}'.format(weights_mv_5[0], weights_mv_5[1], weights_mv_5[2], weights_mv_5[3], weights_mv_5[4]))
print('EXPECTED RETURN: {}'.format(return_mv_5))
print('VOLATILITY: {}'.format(volatility_mv_5))

five_asset.plot_efficient_frontier(color='green')
plt.savefig('efficient_frontier_5.png')

#tangent portfolio

five_asset.plot_tangent_portfolio()

plt.savefig('efficient_frontier_5_tangent.png')

plt.clf()

optimal_p_return, optimal_p_risk = five_asset.tangent_portfolio()
optimal_weights = five_asset.get_weights_for_return(optimal_p_return)

print('\n-----OPTIMAL PORTFOLIO 5 ASSET-----')
print('AAPL, GOOG, MSFT, NVDA, SONY')
print('WEIGHTS: {}, {}, {}, {}, {}'.format(optimal_weights[0], optimal_weights[1], optimal_weights[2], optimal_weights[3], optimal_weights[4]))
print('EXPECTED RETURN: {}'.format(optimal_p_return))
print('VOLATILITY: {}'.format(optimal_p_risk))

# all asset frontier

three_asset.plot_efficient_frontier()
four_asset.plot_efficient_frontier(color='red')
five_asset.plot_efficient_frontier(color='green')
plt.title("Efficient frontier for all asset combinations (equal max return)")
plt.savefig('efficient_frontier_all.png')
```

ex3

```
'''
Author: Orlando Closs
Description: Code to calculate option prices with Black Scholes and other analysis
Date: 08/10/2023
'''
```

```python


import numpy as np
from scipy.stats import norm
import pandas as pd
import matplotlib.pyplot as plt
from py_vollib.black_scholes import implied_volatility

#https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.norm.html
#https://numpy.org/doc/stable/reference/generated/numpy.exp.html

#part a
#FIND REAL r value WITH EXPLANATION

class BlackScholesA():
    def __init__(self, file_path):
        self.file_path = file_path
        self.data = self.compute_returns()
        self.stock_price = self.current_stock_price()
        self.annualized_volatility = self.annualized_volatility()


    def current_stock_price(self):
        current_stock_price = self.data['Close'].iloc[-1]
        return current_stock_price

    def compute_returns(self):
        data = pd.read_csv(self.file_path) #read csv file
        data['Date'] = pd.to_datetime(data['Date']) #change date format
        data = data.sort_values(by='Date').reset_index(drop=True) #sorts list by date
        data['Returns'] = data['Close'].pct_change() #makes new column in data
        return data

    def annualized_volatility(self):
        daily_volatility = self.data['Returns'].std()
        annualized_volatility = daily_volatility * (np.sqrt(250))
        return annualized_volatility

    def black_scholes(self, S_t, K, T, r, sigma):
        d1 = (np.log(S_t / K) + (r - 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
        d2 = d1 - sigma * np.sqrt(T)
        c = S_t * norm.cdf(d1) - K * np.exp(-r * T) * norm.
```

```python
            cdf(d2)
        return c

    def calculate_six_options(self):
        strike_prices = [self.stock_price, self.stock_price
            * 1.1, self.stock_price * 0.9]
        exercise_times = [1/12, 3/12]
        r=0.034
        prices = []
        for K in strike_prices:
            for T in exercise_times:
                c = self.black_scholes(self.stock_price, K,
                    T, r, self.annualized_volatility)
                prices.append(c)

        print('Current Stock Price, 1 Month: {}'.format(
            prices[0]))
        print('Current Stock Price, 3 Months: {}'.format(
            prices[1]))
        print('Current Stock Price + 10%, 1 Month: {}'.
            format(prices[2]))
        print('Current Stock Price + 10%, 3 Months: {}'.
            format(prices[3]))
        print('Current Stock Price - 10%, 1 Month: {}'.
            format(prices[4]))
        print('Current Stock Price - 10%, 3 Months: {}'.
            format(prices[5]))


aapl = BlackScholesA('AAPL-updated.csv')
aapl.calculate_six_options()

#part b and c
#date today 08/10/2023
#Exercise Time-10/11/2023

class BlackScholesBC():
    def __init__(self, file_path, options_path):
        self.file_path = file_path
        self.data = self.compute_returns()
        self.stock_price = self.current_stock_price()
        self.annual_volatility = self.annualized_volatility
            ()
        self.T = 33/250
        self.options_path=options_path
        self.strikes, self.prices, _ = self.
            extract_options_data()
        self.r=0.034

    def extract_options_data(self):
```

```
        strikes = []
        ivs = []
        prices = []
        with open(self.options_path, 'r') as file:
            lines = file.readlines()
            for line in lines:
                strike, price, iv = line.strip().split('-')
                strikes.append(float(strike))
                prices.append(float(price))
                ivs.append(float(iv))
        return strikes, ivs, prices


    def current_stock_price(self):
        current_stock_price = self.data['Close'].iloc[-1]
        return current_stock_price

    def compute_returns(self):
        data = pd.read_csv(self.file_path) #read csv file
        data['Date'] = pd.to_datetime(data['Date']) #change date format
        data = data.sort_values(by='Date').reset_index(drop=True) #sorts list by date
        data['Returns'] = data['Close'].pct_change() #makes new column in data
        return data

    def annualized_volatility(self):
        daily_volatility = self.data['Returns'].std()
        annualized_volatility = daily_volatility * (np.sqrt(250))
        return annualized_volatility

    def black_scholes(self, K, r):
        d1 = (np.log(self.stock_price / K) + (r - 0.5 * self.annual_volatility**2) * self.T) / (self.annual_volatility * np.sqrt(self.T))
        d2 = d1 - self.annual_volatility * np.sqrt(self.T)
        c = self.stock_price * norm.cdf(d1) - K * np.exp(-r * self.T) * norm.cdf(d2)
        return c

    def get_black_scholes_prices(self):

        bs_prices=[]
        for K in self.strikes:
            c = self.black_scholes(K, self.r)
            bs_prices.append(c)

        return bs_prices
```

```python

    def plot_options(self):

        bs_prices = self.get_black_scholes_prices()

        plt.figure(figsize=(10, 6))
        plt.plot(self.strikes[0:12], self.prices[0:12], 'o-', label='Observed Market Prices')
        plt.plot(self.strikes[0:12], bs_prices[0:12], 's-', label='Black-Scholes Theoretical Prices')
        plt.axvline(x=self.stock_price, color='r', linestyle='--', label=f'Current Stock Price')
        plt.xlabel('Strike Prices')
        plt.ylabel('Option Prices')
        plt.title('AAPL Market vs. Black-Scholes Option Prices')
        plt.legend()
        plt.grid(True)

    def plot_implied_volatility(self):

        ivs=[]

        for index, K in enumerate(self.strikes):
            price = self.prices[index]
            iv = implied_volatility.implied_volatility(price, self.stock_price, K, self.T, self.r, 'c')
            ivs.append(iv)

        plt.figure(figsize=(10, 6))
        plt.plot(self.strikes[0:12], ivs[0:12], 'o-', color='blue')
        plt.xlabel('Strike')
        plt.ylabel('Implied Volatility')
        plt.title('AAPL Implied Volatility vs. Strike')
        plt.grid(True)

aapl_options = BlackScholesBC('AAPL-updated.csv', 'optionpriceaapl.txt')

aapl_options.plot_options()
plt.savefig('aapl-marketvsbs-2.png')

plt.clf()

aapl_options.plot_implied_volatility()
plt.savefig('aapl-ivsstrike-2.png')
```