STK–MAT3700

Mandatory assignment 1 of 1

For both Problem 1 and 2 I use Python as a programming language, implemented in VS Code and run on my terminal.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import yfinance as yf
import datetime as dt
import seaborn as sns
import scipy.stats as stats
from pypfopt import risk_models
from pypfopt import expected_returns
from pypfopt import EfficientFrontier
from pypfopt import risk_models
from pypfopt import expected_returns
from math import *
```

Listing 1: Packages

Problem 1

a)

In this task I retrieved 5 series of Norwegian stock prices using Yahoo Finance implemented in Python. The stocks I used were the investment company Arendals Fossekompani, the financial institution DNB, the energy company Equinor, the publishing and media company Schibsted, and the telecommunications company Telenor.

```
start = dt.datetime(2023,1,1)
end = dt.datetime(2024,1,1)

scha = yf.download('SCHA.OL', start, end)
tel = yf.download('TEL.OL', start, end)
afk = yf.download('AFK.OL', start, end)
dnb = yf.download('DNB.OL', start, end)
eqnr = yf.download('EQNR.OL', start, end)

stocks = [scha, tel, afk, dnb, eqnr]
stockn = ['SCHA', 'TEL', 'AFK', 'DNB', 'EQNR']
```

Listing 2: Stock Price Code

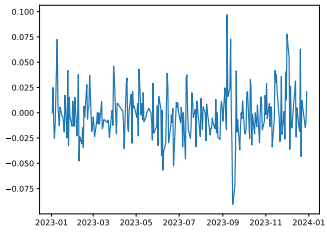
Under, I have plotted the returns of the companies over a one year period between `2023-01-01` and `2024-01-01`.

```
for i in stocks:
    i['ret'] = np.log(i['Adj Close']/i['Adj Close'].shift(1))
    i['ret'].iloc[0] = 0
    i['retmean'] = np.mean(i['ret'])
    i['vlt'] = np.std(i['ret'])

for i, j in enumerate(stocks):
```

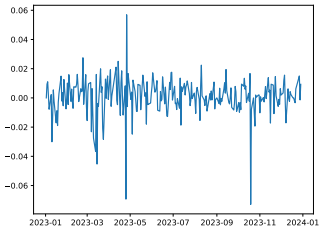
```
    print(f"{stockn[i]:<7} expected return: {np.mean(j['retmean']):<10.6f} volatility: {np.mean(j['vlt
        ']):<10.6f}")

for i, j in enumerate(stocks):
    plt.plot(j.index, j['ret'])
    plt.savefig(f'ret_{stockn[i]}.pdf')
    plt.close()

for i, j in enumerate(stocks):
    plt.plot(j.index, j['vlt'])
    plt.savefig(f'vlt_{stockn[i]}.pdf')
    plt.close()

for i, j in enumerate(stocks):
    mu = j['retmean']
    std = j['vlt']
    x = np.linspace(mu-3*std, mu+3*std, 100)
    j['ret'].plot.density()
    plt.plot(x, stats.norm.pdf(x, mu, std))
    plt.title(f"{stockn[i]} returns v. normal, density")
    # plt.show()
    plt.savefig(f'{stockn[i]}.pdf')
    plt.close()
```

Listing 3: Stock Price Code

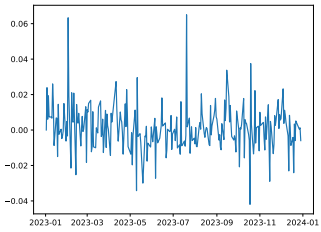
The stock prices does almost look like white noise, with some days of exceptional returns or negative returns.



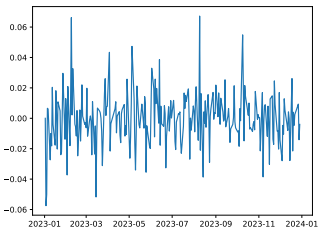
(a) Arendals Fossekompani



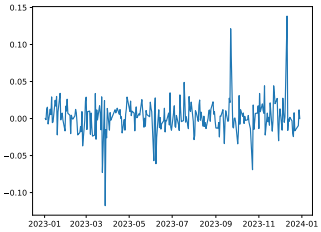
(b) DNB



(c) Equinor

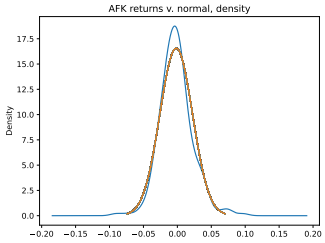


(d) Scibsted A

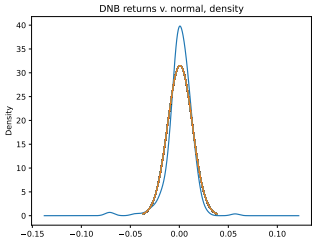


(e) Telenor

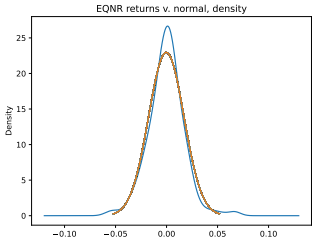
Next, I show density plots of the stocks returns, fitted together with the normal density. This is shown in the five plots below. I find that the stocks returns fits the normal density distribution quite well at times, where Arendals Fossekompani, Equinor and Telenor looks to fit the distribution well, and both Schibsted and DNB having somewhat less density in the mean.



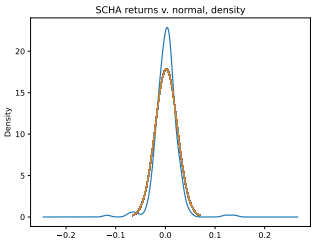
(a) Arendals Fossekompani



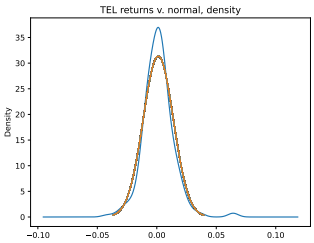
(b) DNB



(c) Equinor



(d) Schibsted



(e) Telenor

b)

In this task I find the correlation and covariance between the selected stocks. First, I aggregate the stocks in a dataframe.

```
cores = np.corrcoef(eqnr['ret'], scha['ret'])
print(cores)

aggregate = yf.download(stockn, start, end)['Adj Close']
for i in stockn:
    aggregate[i] = np.log(aggregate[i]/aggregate[i].shift(1)-1)
    # aggregate[i].iloc[0] = 0
    aggregate[i].fillna(0)

aggreturn = aggregate[['SCHA', 'TEL', 'AFK', 'DNB', 'EQNR']]
print(f"{aggreturn.corr()} \n\n\n\n {aggreturn.cov()}")
print(f"\n")
```

Listing 4: Stock Price Code

Next, I plot a correlation matrix between the selected stocks, which are shown in the table below. I find the correlation by the following formula.

$$r_{XY} = \frac{\sum_{i=1}^{n} \left(X_i - \bar{X}\right)\left(Y_i - \bar{Y}\right)}{\sqrt{\sum_{i=1}^{n} \left(X_i - \bar{X}\right)^2} \sqrt{\sum_{i=1}^{n} \left(Y_i - \bar{Y}\right)^2}} \tag{1}$$

I of course find that stocks correlate perfectly with themselves, while not too much with other stocks. Among the biggest correlation of the selected stocks are between DNB and Schibsted A.

	SCHA	TEL	AFK	DNB	EQNR
SCHA	1.000000	0.324263	0.113642	0.547931	0.159692
TEL	0.324263	1.000000	0.144697	0.290008	0.174503
AFK	0.113642	0.144697	1.000000	0.109668	0.038249
DNB	0.547931	0.290008	0.109668	1.000000	0.003703
EQNR	0.159692	0.174503	0.038249	0.003703	1.000000

Tabell 1: Correlation Matrix

In the table below, I also show the covariance matrix between the stocks.

	SCHA	TEL	AFK	DNB	EQNR
SCHA	1.161560	0.293659	0.097964	0.648467	0.157026
TEL	0.293659	0.991835	0.121404	0.340691	0.144782
AFK	0.097964	0.121404	0.862356	0.105756	0.035151
DNB	0.648467	0.340691	0.105756	1.336277	0.004227
EQNR	0.157026	0.144782	0.035151	0.004227	1.074853

Tabell 2: Covariance Matrix

c)

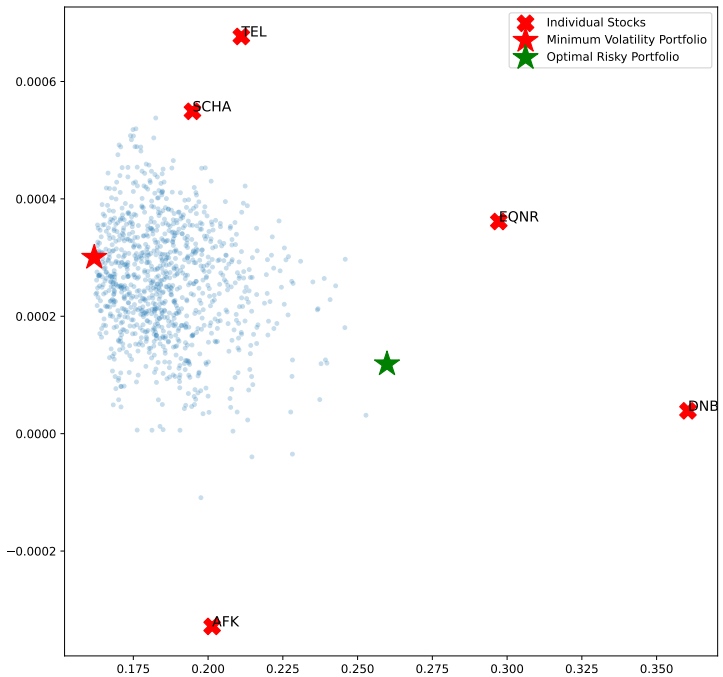
In this task I seek to find an efficient frontier for a portfolio of weights between the selected stocks. Thereafter, I plot the selected stocks in the coordinate system with (σ, r).

```
stocks = ['SCHA', 'TEL', 'AFK', 'DNB', 'EQNR']
df = yf.download(stocks, start, end)
df = df['Adj Close']

ind_er = df.resample("1D").last().pct_change().mean()
w = np.array([0.1, 0.2, 0.2, 0.2, 0.3])
port_er = np.dot(w, ind_er)
ann_sd = df.pct_change().apply(lambda x: np.log(1+x)).std().apply(lambda x: x * np.sqrt(250))
assets = pd.concat([ind_er, ann_sd], axis=1)
assets.columns = ['Returns', 'Volatility']

p_ret = []
p_vol = []
p_weights = []

num_assets = len(df.columns)
num_portfolios = 1000


# generate random portfolios
for _ in range(num_portfolios):
    weights = np.random.random(num_assets)
    weights /= np.sum(weights)
    p_weights.append(weights)
    ret = np.dot(weights, ind_er)
    p_ret.append(ret)
    cov_matrix = df.pct_change().apply(lambda x: np.log(1 + x)).cov() * 250
    vol = np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))
    p_vol.append(vol)

```

```python
# create dataframe
data = {'Return': p_ret, 'Volatility': p_vol}
for counter, symbol in enumerate(df.columns.tolist()):
    data[symbol + ' weight'] = [weight[counter] for weight in p_weights]

portfolios = pd.DataFrame(data)
min_vol_port = portfolios.iloc[portfolios['Volatility'].idxmin()]

# optimal risky portfolio
rf = 0.045
optimal_risky_port = portfolios.iloc[((portfolios['Return'] - rf) / portfolios['Volatility']).idxmax()]
plt.subplots(figsize=(10, 10))
plt.scatter(portfolios['Volatility'], portfolios['Return'], marker='o', s=10, alpha=0.25)
plt.scatter(assets['Volatility'], assets['Returns'], color='red', marker='X', s=200, label="Individual Stocks")
plt.scatter(min_vol_port['Volatility'], min_vol_port['Return'], color='r', marker='*', s=500, label='Minimum Volatility Portfolio')
plt.scatter(optimal_risky_port['Volatility'], optimal_risky_port['Return'], color='g', marker='*', s=500, label='Optimal Risky Portfolio')
plt.legend()
for i, txt in enumerate(assets.index):
    plt.annotate(txt, (assets['Volatility'][i], assets['Returns'][i]), fontsize=12)
plt.savefig(f'optimal_with_stocks.pdf')
plt.close()

print(optimal_risky_port)
```

Listing 5: Stock Price Code

I show the efficient frontier in the plot below, with the minimum volatility portfolio in green, and the optimal portfolio in the far left, tangential to the efficient frontier.



Using the optimal portfolio weight, I find a Sharpe Ratio of 0.000417 for the portfolio with 5 stocks.

Return	0.000112
Volatility	0.268528
AFK weight	0.136274
DNB weight	0.642882
EQNR weight	0.000743
SCHA weight	0.168386
TEL weight	0.051715

Tabell 3: Optimal Weight 5 stocks

$$Sharpe = \frac{0.000112}{0.268528} = 0.000417 \tag{2}$$

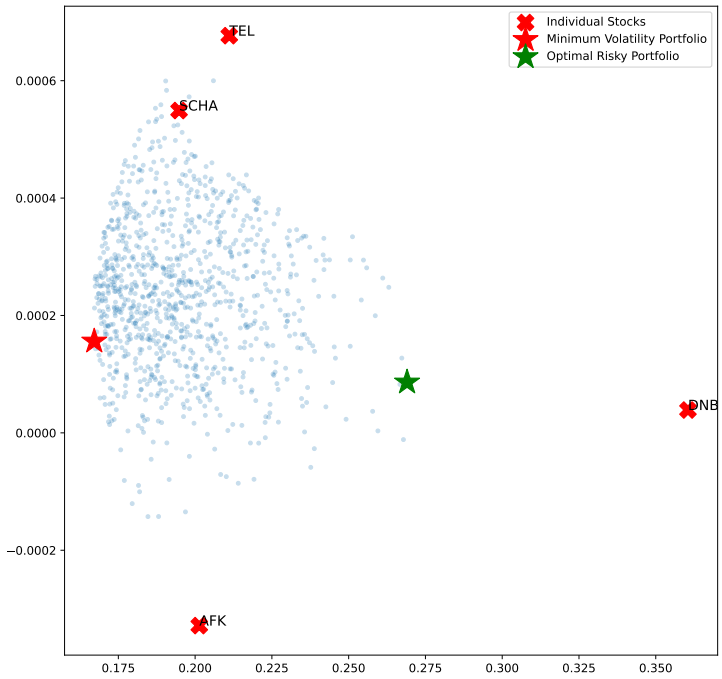
d)

In this task, I repeat the method in task c), only this time removing the Equnior stock. I find that the optimal portfolio using 4 stocks only achieve a Sharpe Ratio of 0.00000495, which is lower than the Sharpe Ratio of the optimal portfolio with 5 stocks. This is expected, as we have reduced the possible investment universe compared to in task c).

Return	0.000145
Volatility	0.292632
AFK weight	0.045948
DNB weight	0.729939
SCHA weight	0.192906
TEL weight	0.031207

Tabell 4: Optimal Weight 4 stocks

$$Sharpe = \frac{0.000145}{0.292632} = 0.00000495 \tag{3}$$



Problem 2

a)

In this task I implement the price of call options using the Black and Scholes. I selected Norwegian stocks, and have therefore decided to use the risk-free rate equal to the Norwegian central bank rate, with a weighted mean of the Norwegian central bank rate throughout the time period of `2023-01-01` to `2024-01-01`. I therefore choose to use the following.

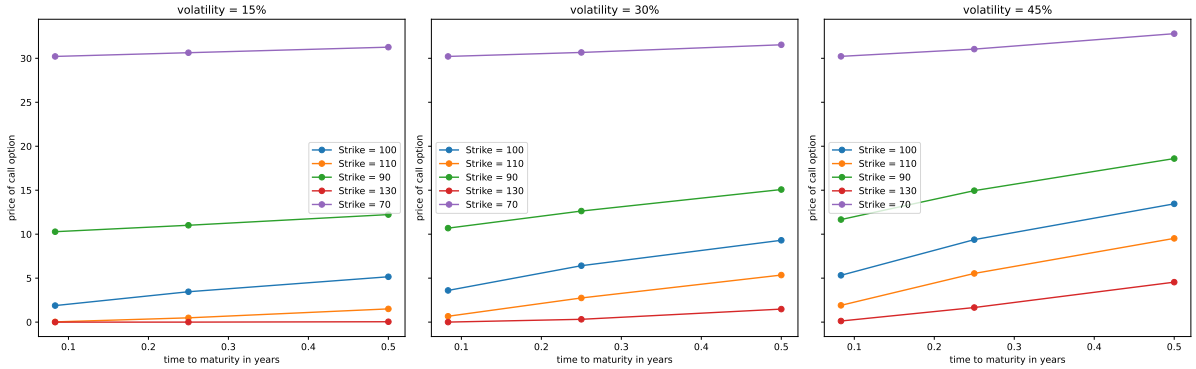
$$\frac{2.75 + 4.5}{2} = 3.625 \tag{4}$$

```
1
2  def black_scholes_call(S0, K, T, r, sigma):
3      d1 = (np.log(S0 / K) + (r + 0.5 * sigma ** 2) * T) / (sigma * np.sqrt(T))
4      d2 = d1 - sigma * np.sqrt(T)
5      call_price = S0 * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2)
6      return call_price
7
8  S0 = 100
9  r = 0.0365
10 volatilities = [0.15, 0.30, 0.45]
11 times_to_maturity = [1/12, 3/12, 6/12]
12 strike_multiples = [1, 1.1, 0.9, 1.3, 0.7]  # strike prices as percentages of S0
13
14 fig, axs = plt.subplots(1, 3, figsize=(18, 6), sharey=True)
15 strikes = [S0 * multiple for multiple in strike_multiples]
16
17 for i, sigma in enumerate(volatilities):
```

```
    prices = np.zeros((len(strikes), len(times_to_maturity)))
    for j, K in enumerate(strikes):
        for k, T in enumerate(times_to_maturity):
            prices[j, k] = black_scholes_call(S0, K, T, r, sigma)

    for j, K in enumerate(strikes):
        axs[i].plot(times_to_maturity, prices[j], marker='o', label=f'Strike = {K:.0f}')

    axs[i].set_title(f'volatility = {sigma * 100:.0f}%')
    axs[i].set_xlabel('time to maturity in years')
    axs[i].set_ylabel('price of call option')
    axs[i].legend()

plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.savefig('calloptions.pdf')
plt.close()
```

Listing 6: Black-Scholes Python Code

Here, I have plotted the call options over time of maturity, and divided into a volatility of 15%, 30%, and 45%. I find that the price of call options increases over the time of maturity, and that that lower strikes correspond to higher price of call options.



Figur 3: Call options

b)

I collect call option prices from the Standard and Poor's 500 index, from the following website: `https://www.barchart.com/stocks/quotes/&SPX/options?expiration=2024-10-21-w`.

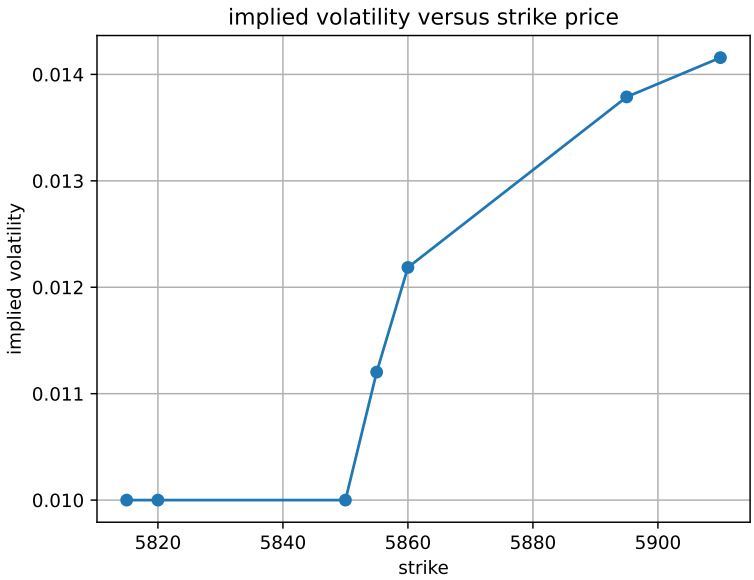
I select $S_0 = 5850$, with the corresponding strike prices in the interval $5815 - -5910$, with market prices in the interval $53.35 - -0.55$. As such, we find that there are large variations in the market prices over relatively small changes in the strike prices.

```
def black_scholes_call(S0, K, T, r, sigma):
    d1 = (np.log(S0 / K) + (r + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    call_price = S0 * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2)
    return call_price

def implied_volatility(market_price, S0, K, T, r):
    def objective(sigma):
        return (black_scholes_call(S0, K, T, r, sigma) - market_price)**2
    result = minimize(objective, 0.2, bounds=[(0.01, 5.0)])
```

```
    return result.x[0]

S0 = 5850  # current stock price
K_values =      [5815,  5820,   5850,   5855,   5860,   5895,   5910]  # strike prices
market_prices = [53.35, 48.65,  21.60,  18.55,  15.25,  1.92,   0.55]  # market prices for the call options
T = 30 / 365  # time to maturity in years
r = 0.045  # risk-free interest rate (4.5%)

implied_vols = [implied_volatility(market_price, S0, K, T, r) for K, market_price in zip(K_values,
    market_prices)]

plt.plot(K_values, implied_vols, marker='o')
plt.xlabel('strike')
plt.ylabel('implied volatility')
plt.title('implied volatility versus strike price')
plt.grid(True)
# plt.show()
plt.savefig('impliedvolatility.pdf')
plt.close()
```

Listing 7: Implied Volatility

In the following figure, I have plotted the implied volatility over strike price, where we find a stable implied volatility when the strike price S is below the initial strike price $S_0 = 5850$, and thereafter increases.



Figur 4: Implied Volatility

c)

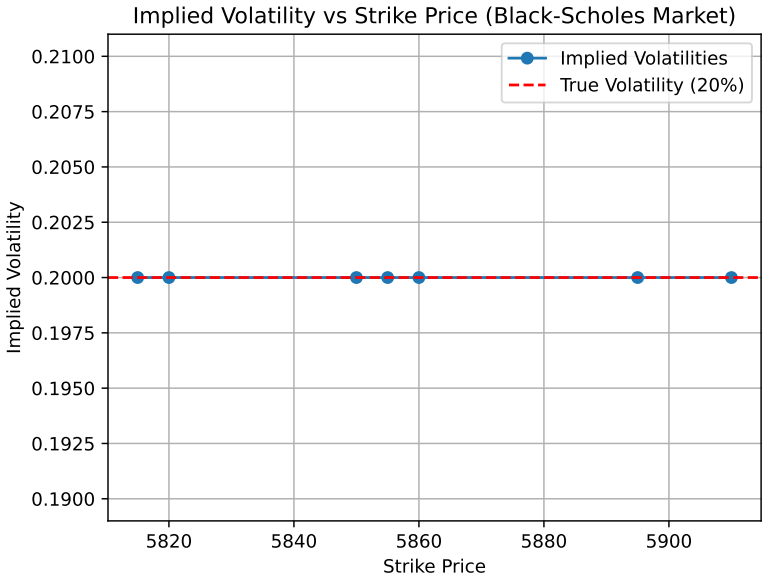
Next, I implement the implied volatility using the Black Scholes pricing model from task a).

```
    def black_scholes_call(S0, K, T, r, sigma):
    d1 = (np.log(S0 / K) + (r + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    call_price = S0 * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2)
    return call_price
```

```
6
7  def implied_volatility(market_price, S0, K, T, r):
8      def objective(sigma):
9          return (black_scholes_call(S0, K, T, r, sigma) - market_price)**2
10     result = minimize(objective, 0.2, bounds=[(0.01, 5.0)])  # Initial guess of 20% volatility
11     return result.x[0]
12
13 S0 = 5850  # Current stock price of S&P 500
14 K_values = [5815, 5820, 5850, 5855, 5860, 5895, 5910]  # Strike prices
15 T = 30 / 365  # Time to maturity (30 days)
16 r = 0.045  # Risk-free interest rate (4.5%)
17 true_volatility = 0.2  # The volatility assumed by the market (20%)
18
19 market_prices_bs = [black_scholes_call(S0, K, T, r, true_volatility) for K in K_values]
20
21 implied_vols_bs = [implied_volatility(price, S0, K, T, r) for price, K in zip(market_prices_bs, K_values)]
22
23 plt.plot(K_values, implied_vols_bs, marker='o', label="Implied Volatilities")
24 plt.axhline(y=true_volatility, color='r', linestyle='--', label="True Volatility (20%)")
25 plt.xlabel('Strike Price')
26 plt.ylabel('Implied Volatility')
27 plt.title('Implied Volatility vs Strike Price (Black-Scholes Market)')
28 plt.grid(True)
29 plt.legend()
30 plt.savefig('implied_volatility_bs_market.pdf')
31 plt.close()
```

Listing 8: Implied Volatility using Black-Scholes to price the options

In the world where we use Black Scholes to price the call option, we find that the true volatility (here set to 20%) is equal to the implied volatility found in Black Scholes. The market is efficient, and the investors have priced in all available information, and therefore reducing any arbitrage opportunities.



Figur 5: Implied Volatility using Black Scholes