option-pricing-model

September 20, 2023

[213]:
```
import numpy as np
import pandas as pd
import yfinance as yf
from scipy.optimize import minimize
import matplotlib.pyplot as plt
from scipy.stats import norm
```

[214]:
```
data = yf.download('EURUSD=X', start='2023-09-15', end='2023-09-20')
stock_price = data["Close"].iloc[-1]
```

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

[215]:
```
# Calculate daily returns
data['Returns'] = data['Adj Close'].pct_change().dropna()
```

[216]:
```
# Calculate daily volatility (standard deviation of returns)
daily_volatility = data['Returns'].std()
```

[217]:
```
# Annualize the volatility (assuming 252 trading days in a year)
annual_volatility = daily_volatility * np.sqrt(252)
```

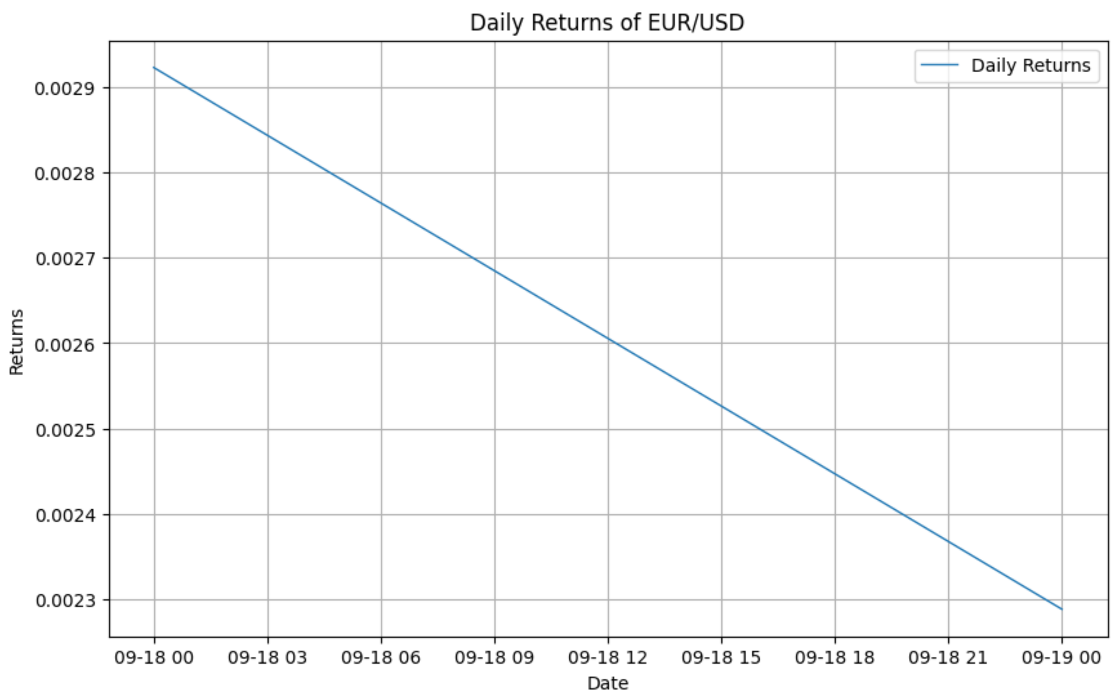
[218]:
```
data.dropna()
```

[218]:
```
                Open      High       Low     Close  Adj Close  Volume  \
Date
2023-09-18  1.066826  1.069816  1.065632  1.066826   1.066826       0
2023-09-19  1.069267  1.071926  1.067635  1.069267   1.069267       0

             Returns
Date
2023-09-18  0.002923
2023-09-19  0.002288
```

[219]:
```
# Print the results
print("Daily Volatility: {:.4f}".format(daily_volatility))
print("Annual Volatility: {:.4f}".format(annual_volatility))
```

```
Daily Volatility: 0.0004
Annual Volatility: 0.0071
```

[220]:
```python
# Visualize the daily returns
plt.figure(figsize=(10, 6))
plt.plot(data.index, data['Returns'], label='Daily Returns', linewidth=1)
plt.title('Daily Returns of EUR/USD')
plt.xlabel('Date')
plt.ylabel('Returns')
plt.legend()
plt.grid(True)

# Show the plot
plt.show()
```



[221]:
```python
# Define the option parameters
S0 = 1.0701   # Current stock price
K = 1.0704   # Strike price
T = 1       # Time to expiration (in years)
r = 0.0048   # Risk-free interest rate
N = 3     # Number of time steps
sigma = 0.0004 # Volatility
```

Binomial

The Binomial pricing model is a method for evaluating an option by using the varying price as a function of time for the financial instrument

[222]:
```
returns = data["Adj Close"].pct_change().dropna()
```

[223]:
```
# Calculate the daily volatility
daily_volatility = returns.std()
```

[224]:
```
# Calculate parameters for the binomial model
dt = T / N
u = np.exp(sigma * np.sqrt(dt))
d = 1 / u
q = (np.exp(r * dt) - d) / (u - d)
```

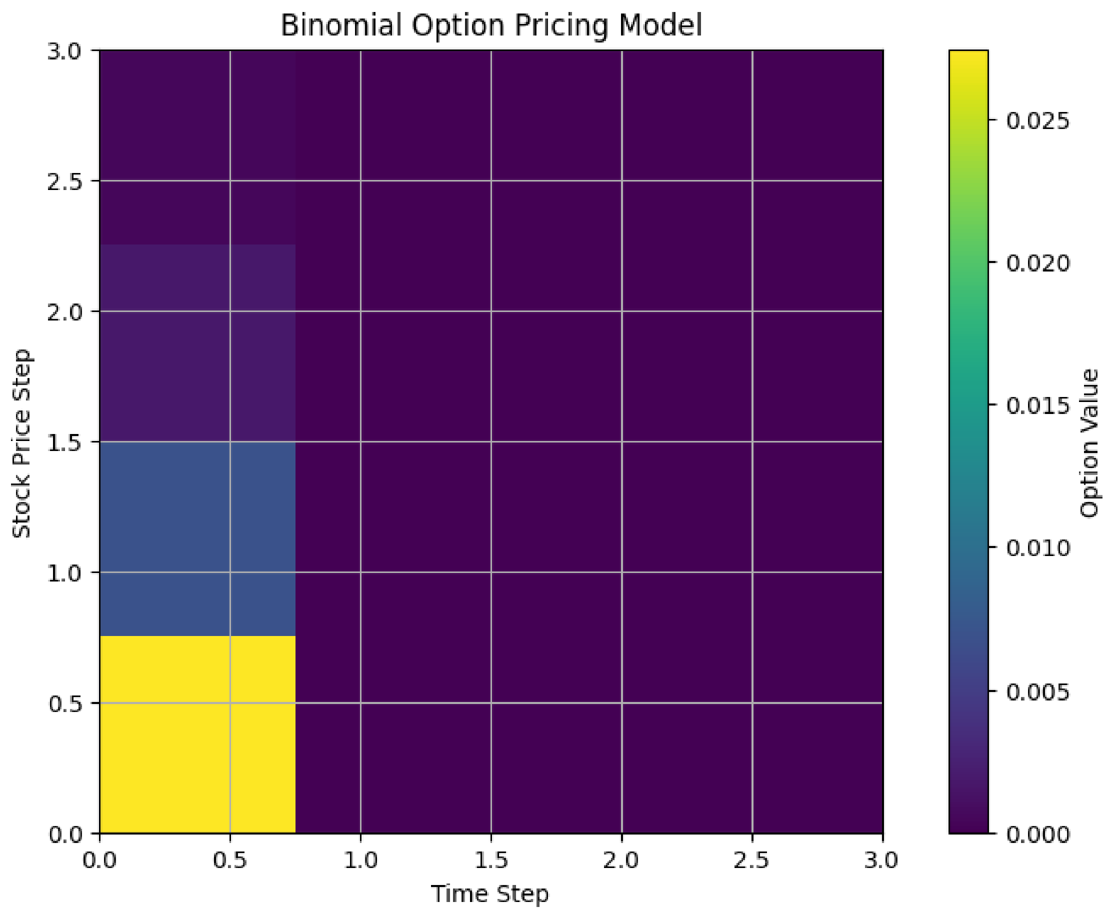
[225]:
```
# Initialize arrays for stock prices and option values
stock_prices = np.zeros((N + 1, N + 1))
option_values = np.zeros((N + 1, N + 1))
```

[226]:
```
# Calculate stock prices at each node
for i in range(N + 1):
    for j in range(i + 1):
        stock_prices[i, j] = S0 * (u ** (i - j)) * (d ** j)
```

[227]:
```
# Calculate option values at expiration (at the last time step)
option_values[N, :] = np.maximum(0, stock_prices[N, :] - K)
```

[228]:
```
# Calculate option values at earlier time steps using backward induction
for i in range(N - 1, -1, -1):
    for j in range(i + 1):
        option_values[i, j] = np.exp(-r * dt) * (q * option_values[i + 1, j] +
 ↪(1 - q) * option_values[i + 1, j + 1])
```

[229]:
```
# Create a plot
plt.figure(figsize=(10, 6))
plt.imshow(option_values, cmap='viridis', extent=[0, N, 0, N], origin='lower')
plt.colorbar(label='Option Value')
plt.title('Binomial Option Pricing Model')
plt.xlabel('Time Step')
plt.ylabel('Stock Price Step')
plt.grid(True)

# Show the plot
plt.show()
```




```
[230]: option_price = option_values[0, 0]
       print(f"The fair option price is: {option_price:.2f}")
```

```
The fair option price is: 0.03
```

Monte Carlo

```
[231]: # Calculate daily returns
       daily_returns = data['Adj Close'].pct_change().dropna()
```

```
[232]: # Calculate mean and standard deviation of daily returns
       avg_daily_return = daily_returns.mean()
       std_daily_return = daily_returns.std()
```

```
[233]: num_simulations = 100        # Number of Monte Carlo simulations
       num_days = 252               # Number of trading days in a year
```

```
[234]: # Initialize arrays to store results
       simulated_prices = np.zeros((num_simulations, num_days))
```

[235]:

```
# Perform Monte Carlo simulation
for i in range(num_simulations):
    daily_volatility = std_daily_return / np.sqrt(num_days)
    daily_returns_sim = np.random.normal(avg_daily_return, daily_volatility,␣
 ↪num_days)
    price_sim = np.zeros(num_days)
    price_sim[0] = data['Adj Close'][-1]  # Initial stock price

    for j in range(1, num_days):
        price_sim[j] = price_sim[j - 1] * (1 + daily_returns_sim[j])

    simulated_prices[i, :] = price_sim
```

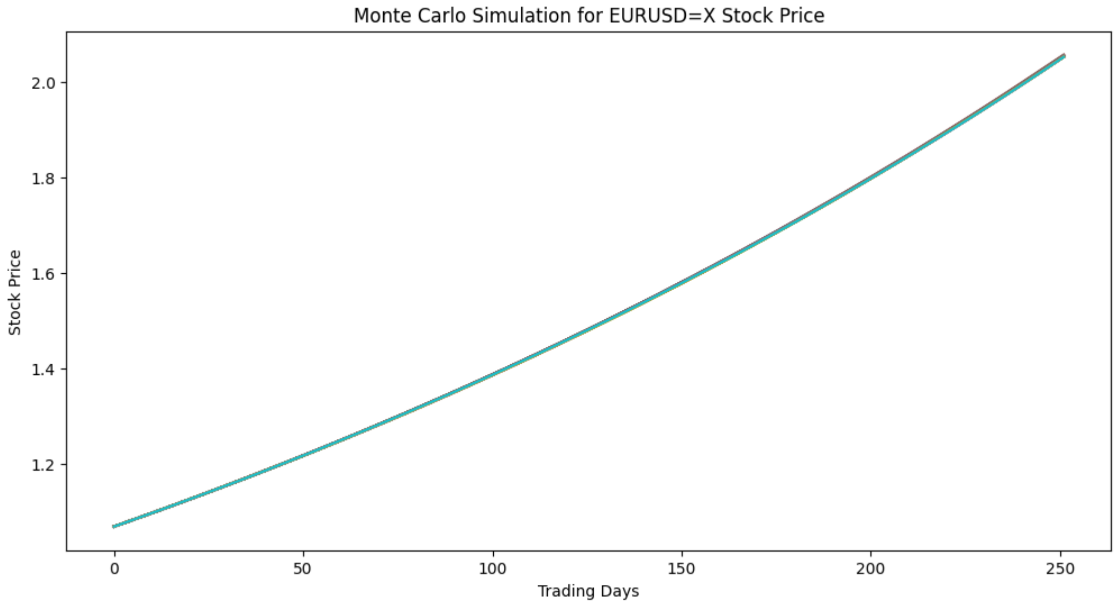
[236]:

```
print(f"Monte Carlo Simulation for {simulated_prices} Stock Price")
```

```
Monte Carlo Simulation for [[1.06926715 1.07205468 1.07480161 … 2.04460959
2.04983681 2.05521339]
 [1.06926715 1.07201468 1.07486222 … 2.04440116 2.04975613 2.05513123]
 [1.06926715 1.07211594 1.07486977 … 2.04392804 2.049248   2.05453409]
 …
 [1.06926715 1.07206827 1.07488631 … 2.04359807 2.04910507 2.05440082]
 [1.06926715 1.07205662 1.07486264 … 2.04260456 2.04802107 2.05338566]
 [1.06926715 1.0719993  1.07480598 … 2.04222165 2.04753727 2.05283722]] Stock
Price
```

[237]:

```
# Plot the Monte Carlo simulation results
plt.figure(figsize=(12, 6))
for i in range(num_simulations):
    plt.plot(range(num_days), simulated_prices[i, :])

plt.title(f"Monte Carlo Simulation for {symbol} Stock Price")
plt.xlabel("Trading Days")
plt.ylabel("Stock Price")
plt.show()
```



Black Scholes

[238]:
```
# Define the parameters
symbol = "EURUSD=X"  # Stock symbol
rf = 0.0048  # Risk-free rate (e.g., 1%)
T = 1  # Time to expiration (in years)
K = 1.0692  # Strike price
sigma = 0.0004 # Volatility (e.g., 20%
```

[239]:
```
d1 = (np.log(data / K) + (rf + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
d2 = d1 - sigma * np.sqrt(T)
```

```
c:\Users\akram\AppData\Local\Programs\Python\Python310\lib\site-
packages\pandas\core\internals\blocks.py:351: RuntimeWarning: divide by zero
encountered in log
  result = func(self.values, **kwargs)
```

[240]:
```
# Calculate the put option price using the BSM formula
put_option_price = K * np.exp(-rf * T) * norm.cdf(-d2) - data * norm.cdf(-d1)
call_option_price = data * norm.cdf(d1) - K * np.exp(-rf * T) * norm.cdf(d2)
```

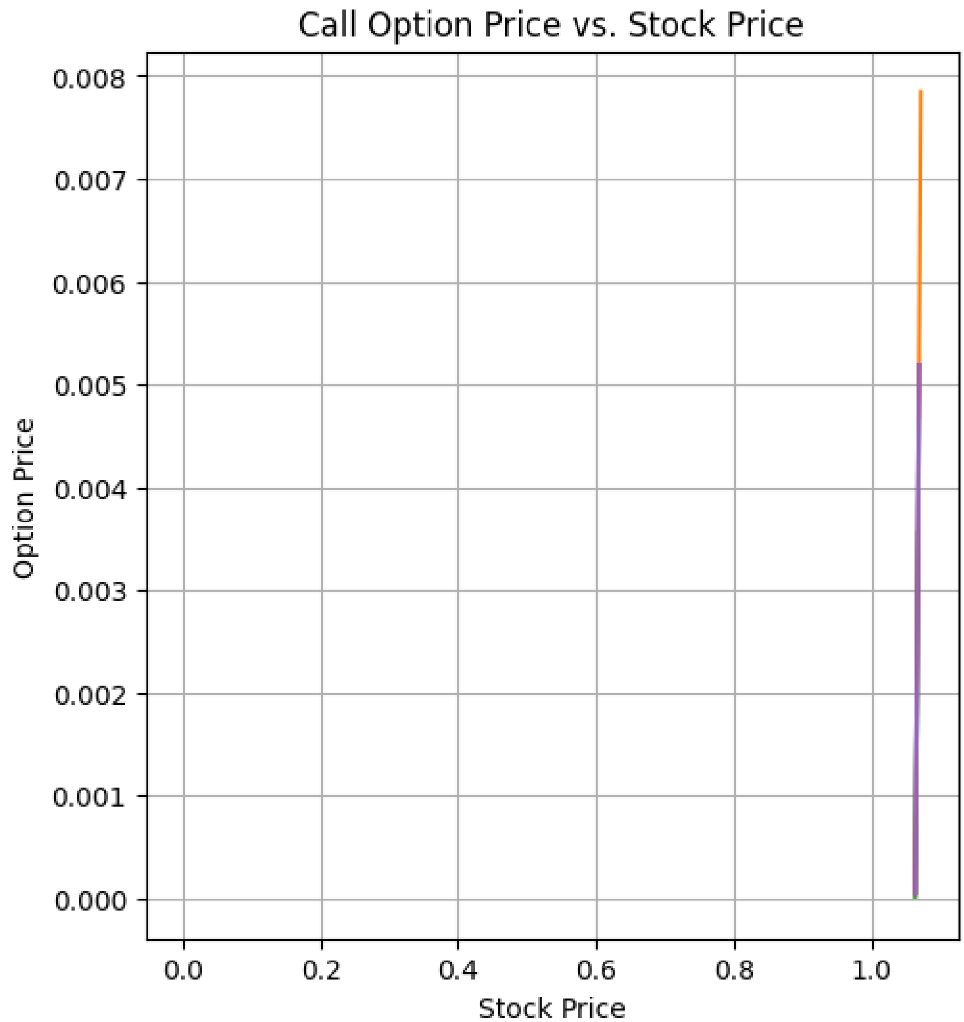
[241]:
```
print("Black-Scholes-Merton Put Call Price:", put_option_price)
print("Black-Scholes-Merton Call Option Price:", call_option_price)
```

```
Black-Scholes-Merton Put Call Price:                    Open          High
Low         Close  \
Date
2023-09-15  4.099397e-04  9.004465e-33  6.266999e-04  4.099397e-04
```

```
2023-09-18  3.708662e-15  5.501193e-46  1.390695e-08  3.708662e-15
2023-09-19  9.117851e-39  2.823864e-80  1.890385e-21  9.117851e-39

               Adj Close   Volume   Returns
Date
2023-09-15  4.099397e-04  1.06408       NaN
2023-09-18  3.708662e-15  1.06408  1.061157
2023-09-19  9.117851e-39  1.06408  1.061792
Black-Scholes-Merton Call Option Price:                Open      High       Low
Close  Adj Close  Volume  Returns
Date
2023-09-15  0.000046  0.004684  0.000014  0.000046   0.000046     0.0      NaN
2023-09-18  0.002746  0.005736  0.001552  0.002746   0.002746     0.0      0.0
2023-09-19  0.005187  0.007846  0.003555  0.005187   0.005187     0.0      0.0
```

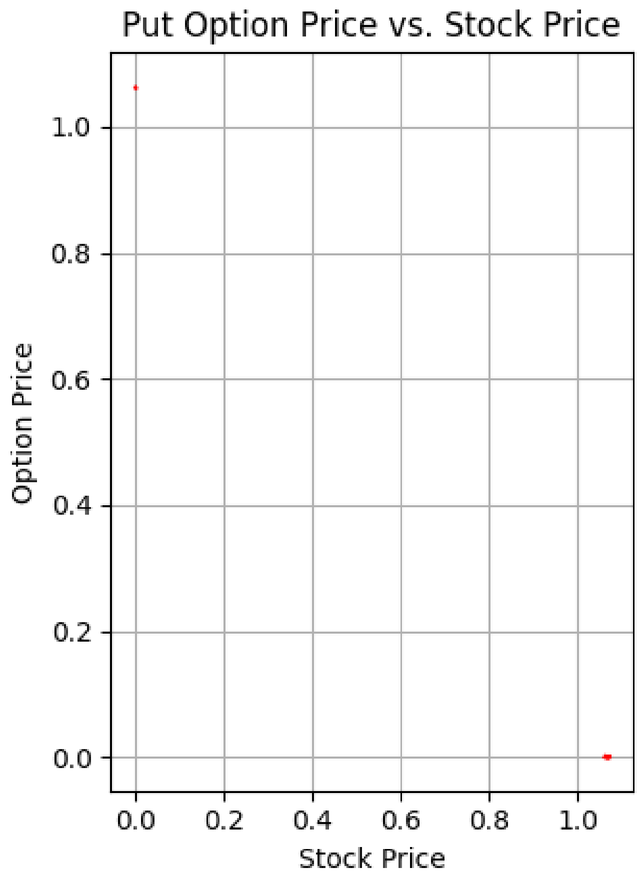
[242]:

```
# Create plots
plt.figure(figsize=(12, 6))

# Call Option Price vs. Stock Price
plt.subplot(1, 2, 1)
plt.plot(data, call_option_price, label='Call Option Price')
plt.xlabel('Stock Price')
plt.ylabel('Option Price')
plt.title('Call Option Price vs. Stock Price')
plt.grid()
```



[243]:

```
# Put Option Price vs. Stock Price
plt.subplot(1, 2, 2)
plt.plot(data, put_option_price, label='Put Option Price', color='red')
plt.xlabel('Stock Price')
plt.ylabel('Option Price')
plt.title('Put Option Price vs. Stock Price')
plt.grid()

plt.tight_layout()
plt.show()
```


Put Option Price vs. Stock Price
Option Price
Stock Price
0.0
0.2
0.4
0.6
0.8
1.0