UNIVERSITY OF WESTMINSTER

WESTMINSTER BUSINESS SCHOOL

INDIVIDUAL AUTHENTIC

ASSESSMENT COURSEWORK 2

MODULE TITLE: COMPUTATIONAL METHODS FOR FINANCE

MODULE CODE: 7FNCE041W.1

COURSE: MSC FINTECH AND BUSINESS ANALYTICS (CORE),

SEMESTER 1, 2024/2025

STUDENT ID: 20497211

WORD COUNT: 2300 words

(EXCLUDING TABLE OF CONTENT, TABLE OF FIGURES, BIBLIOGRAPHY, EQUATIONS)

TABLE OF CONTENTS

TABLE OF FIGURES

ABSTRACT

This report compares computational methods for option pricing and volatility estimation in financial markets. The exercises to be performed fall under three categories:

1. Option Pricing:

Design and price a European call option using the BSM formula and Monte Carlo simulation to compare results for validation purposes. Results from the two approaches will be compared to validate the results and discuss minor differences resulting from the stochastic nature of Monte Carlo simulations.**(Boyle, 1977)**

2. Implied Volatility:

Historical and implied volatilities were compared to explain their unique properties and advantages. Numerical estimates of implied volatility using the Newton-Raphson method have been implemented and explained using Python. Compute the implied volatilities of Tesla's put options with two different strikes and compare them with the historical volatility measured earlier. **(Heston ,1993)**

3. Binomial Tree Option Pricing: A three-step binomial tree was used for pricing an American call option. Implementation was done in Python, and its value was compared to that computed theoretically. **(Cox, Ross & Rubinstein, 1979)**

Key insights are provided concerning the accuracy of the BSM model for standard options, poor behaviour independent of Monte Carlo methods for a few samples, and overpricing due to very high implied volatility. The report provides essential comparisons supported by Python implementations and theoretical analysis of the essence of computational techniques in modern finance.

GITHUB: **AUTHENTIC_ASSESSMENT_COURSEWORK_2**

1. OPTION PRICING

Option pricing is a very basic concept in the financial markets; it refers to the values of derivatives. In the given paper, the prices of a European call option have been evaluated through two different approaches:

1. The Black-Scholes-Merton model.
2. Monte Carlo simulation.

This work is implemented in Python-based applications, and the performance results are measured by accuracy and practical impact. [GitHub]

Option Design

- **Option Type**: European Call
- **Inputs**:
    - **Spot Price (SSS)**: $100 (current price of the underlying asset)
    - **Strike Price (K)**: $105 (agreed price to buy the asset)
    - **Time to Expiry (T)**: 1 year
    - **Risk-Free Rate (r)**: 5% (assumed return of a risk-free investment)
    - **Volatility (σ)**: 20% (degree of fluctuation in the asset price)

The European call option was selected because its payoff depends solely on the spot price at maturity, and it does not involve early exercise. (HULL, 2021)

Methodology

Black-Scholes-Merton Formula (Black, F., & Scholes M. ,1973)

The model of BSM is the most popular method of analytical valuing options, especially in Europe. This formula represents the option's fair value calculation from underlying parameters:

$$C = S\Phi(d1) - Ke - rT\Phi(d2)$$

Where:

$$d1 = \frac{ln(S/K) + (r + \sigma^2/2)T}{\sigma\sqrt{T}}$$

Here, Φ(d) represents the cumulative distribution function of the standard normal distribution.

Result: The price of the European call option using the BSM formula is **$8.02**.

Monte Carlo Simulation

One of the ways of pricing by generating stochastic trajectories is a Monte Carlo simulation of the stock price. The formula for option pricing could then be:

$$C = e - rT \cdot Average(max(ST - K, 0))$$

- Where ST represents the simulated stock price at maturity.

Monte Carlo method:

• Simulates possible future stock prices using random sampling.

• Slight differences in the results are possible because random number generation is inherently random.

Inconsistency Removal:

• Running more simulations can decrease the standard error.

• Using a fixed random seed ensures reproducibility.

Result: The price of the European call option using Monte Carlo simulation is **$8.04**.

The following code in Python calculates the value of a European call option using the Black-Scholes-Merton formula combined with Monte Carlo simulation methods.

```python
import numpy as np
from scipy.stats import norm

# Black-Scholes-Merton Formula
def black_scholes_call(S, K, T, r, sigma):
    """
    Calculate the price of a European call option using Black-Scholes-Merton formula.
    S: Spot price
    K: Strike price
    T: Time to maturity (in years)
    r: Risk-free rate
    sigma: Volatility
    """
    d1 = (np.log(S / K) + (r + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    call_price = S * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2)
    return call_price

# Monte Carlo Simulation
def monte_carlo_call(S, K, T, r, sigma, simulations=100000):
    """
    Calculate the price of a European call option using Monte Carlo simulation.
    """
    np.random.seed(42)  # Set seed for reproducibility
    Z = np.random.standard_normal(simulations)  # Generate random normal values
    ST = S * np.exp((r - 0.5 * sigma**2) * T + sigma * np.sqrt(T) * Z)  # Simulated terminal prices
    payoff = np.maximum(ST - K, 0)  # Payoff at maturity
    call_price = np.exp(-r * T) * np.mean(payoff)  # Discounted average payoff
    return call_price

# Parameters
S = 100  # Spot price
K = 105  # Strike price
T = 1    # Time to maturity
r = 0.05  # Risk-free rate
sigma = 0.2  # Volatility

# Calculate call option price
bs_price = black_scholes_call(S, K, T, r, sigma)
mc_price = monte_carlo_call(S, K, T, r, sigma)

print(f"Black-Scholes Price: {bs_price:.4f}")
print(f"Monte Carlo Price: {mc_price:.4f}")
```

```
Black-Scholes Price: 8.0214
Monte Carlo Price: 8.0416
```

Figure 1Black-Scholes and Monte Carlo Simulations

Results Comparison

- **BSM Price**: $8.02
- **Monte Carlo Price**: $8.04

Similarities:

- Both approaches gave almost the same result, which converged to the theoretical fair value option.
- Both formulas have the same inputs: spot price, strike price, volatility, time to maturity, and the risk-free rate.

Differences:

Aspect	**Black-Scholes-Merton Formula**	**Monte Carlo Method**
Approach	Analytical approach providing an exact closed-form solution.	A numerical approach using random sampling is used to simulate potential outcomes.
Application	Ideal for European options with standard assumptions (no early exercise or dividends).	Suitable for more complex options, such as path-dependent or exotic options.
Precision	Provides deterministic and consistent results.	Results vary slightly due to randomness, but precision improves with more simulations.
Speed	Computationally efficient and faster.	Slower, especially for many simulations.
Complexity	Straightforward for standard options.	It can be adapted to handle complex scenarios, such as non-standard payoffs.
Result Variance	No variation, always deterministic	Minor variations due to randomness require a high number of simulations for accuracy.
Usability in Practice	Limited to standard options with assumptions like constant volatility and no jumps.	Flexible for modelling scenarios where standard assumptions do not hold.
Market Use	Widely used for quick and standard pricing.	Often used in scenarios requiring stress testing or risk modelling.

Black-Scholes-Merton Formula:

- The price of an analytical solution was $8.02.
- Assumes constant volatility, log-normal distribution of prices, and frictionless markets..

Monte Carlo Simulation:

- Estimated the answer to be $8.04 using 10^6 simulations.
- Accommodates more intricate distributions but introduces minor fluctuations attributable to randomness.

Advantages and Limitations

Aspect	Black-Scholes-Merton Formula	Monte Carlo Method
Advantages	Quick, precise, and easy to compute for standard options.	Flexible for pricing exotic or path-dependent options.
Limitations	Limited to assumptions of log-normal distribution and no early exercise.	Computationally intensive and slower for large simulations.

The merits of each of the above two methods lie in somewhat different applications:

- Black-Scholes-Merton Formula: Applies directly to standard European options; gives accurate results and is computationally efficient under model assumptions.
- However, Monte Carlo Simulation can do more complicated option prices with heavier computation and trivial stochastic
  fluctuations.

2. IMPLIED VOLATILITY

a). Comparing Historical Volatility and Implied Volatility

1. **Historical Volatility (HV)**:
   - Measures the actual fluctuations in an asset's price over a specific past time period.
   - Calculated using the standard deviation of logarithmic returns of the asset over a time window.

Formula:

$$\sigma H = \sqrt{\frac{\sum_{i=1}^{n} l\, n(ln(Pi/Pi - 1) - \mu)2}{n - 1}}$$

Where:

- Pi: Price of the asset at time iii.
- μ: Mean of log returns.
- n: Number of observations.

2. **Implied Volatility (IV)**:
   - A forward-looking measure derived from the market price of an option.
   - Indicates the market's expectations of future volatility over the option's life.
   - Derived by solving the option pricing model (e.g., Black-Scholes-Merton) for volatility.

Similarities

• Both quantify price fluctuations of an underlying asset.
• Both use the standard deviation as a measure of variation.

Differences

Aspect	**Historical Volatility**	**Implied Volatility**
Perspective	Backward-looking (based on past data)	Forward-looking (market expectation)
Calculation Source	Historical prices	The market price of the option
Usage	Risk assessment of past performance	Pricing and hedging derivatives

While historical volatility shows realised market behaviour, implied volatility reflects the market's sentiment about future movements (Black & Scholes, 1973).

b). Newton-Raphson Iteration to Estimate Implied Volatility

Purpose

The Newton-Raphson method represents the iterative numerical techniques employed in approximating implied volatility. It seeks to minimise the difference between the market price and the model price of an option.

Steps

1. **Objective**: Solve $f(\sigma)=0$, where:

   $f(\sigma)=\text{Option Price from BSM}(\sigma)-\text{Market Option Price}$

2. **Update Rule**:

$$\sigma new = \sigma old - \frac{f'(\sigma old)}{f(\sigma old)}$$

   - f(σold): Difference between BSM price and observed price.
   - f′(σold): Vega (sensitivity of option price to changes in volatility).
3. **Vega Calculation**: Vega measures the rate of change of the option price concerning volatility:

$$Vega = S\phi(d1)\sqrt{T}$$

Where ϕ(d1) is the standard normal probability density function.

4. **Iterative Adjustment**: Update σ iteratively until f(σ) is sufficiently close to zero (within tolerance).

Step-by-Step Explanation:

Initialisation: The starting guess for implied volatility (σ) is set to a reasonable value (e.g., 0.2 for 20%), ensuring the iterative process starts near the expected solution.

Newton-Raphson Iteration: This, given by Vega, is a measure of an option's sensitivity to volatility. With that, iterative, incremental changes converge to zero.

Convergence check: The loop runs when Δσ is smaller than a given constraint, such as 1e-6, to obtain an accurate solution with less computation.

Finally, the implied volatility is printed out and checked against the historical volatility for validation.

c) Python Code to Show the Newton-Raphson Process

Here is the Python code for the Newton-Raphson method to estimate implied volatility for a European call option:

```python
from scipy.stats import norm
import numpy as np

def black_scholes_price(S, K, T, r, sigma, option_type='call'):
    d1 = (np.log(S / K) + (r + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)

    if option_type == 'call':
        return S * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2)
    else:
        return K * np.exp(-r * T) * norm.cdf(-d2) - S * norm.cdf(-d1)

def vega(S, K, T, r, sigma):
    d1 = (np.log(S / K) + (r + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
    return S * norm.pdf(d1) * np.sqrt(T)

def newton_raphson_iv(observed_price, S, K, T, r, option_type='call', tol=1e-6, max_iter=100):
    sigma = 0.2  # Initial guess
    for i in range(max_iter):
        price = black_scholes_price(S, K, T, r, sigma, option_type)
        vega_val = vega(S, K, T, r, sigma)
        price_diff = price - observed_price

        if abs(price_diff) < tol:
            return sigma
        sigma -= price_diff / vega_val
    raise ValueError("Newton-Raphson did not converge")

# Example
S, K, T, r, observed_price = 100, 105, 1, 0.05, 8.02
iv = newton_raphson_iv(observed_price, S, K, T, r)
print(f"Implied Volatility: {iv:.4f}")
```

```
Implied Volatility: 0.2000
```

Figure 2 Python Code Screenshot of Newton-Raphson Process

d). Estimation of Implied Volatility for Tesla's Put Options

This explains the step-by-step estimation of IV for Tesla's put option using the Black-Scholes model. We consider two strike prices, namely, $140 and $340, with a time to maturity of January 17, 2025. Further, the Yahoo Finance API obtains the spot price and puts option prices to ensure freshness. The detailed steps are provided below, along with text, code, outputs, and analytics commentary wherever applicable.

Data Retrieval

The Yahoo Finance API obtained the spot price of Tesla's stock and put option prices. The parameters for the calculations are:

- **Spot Price (S):** Real-time market price of Tesla stock.
- **Strike Prices (K):** $140 and $340.
- **Expiry Date:** 17th January 2025.
- **Risk-Free Rate (r):** 5% (three-month T-bill rate).
- **Observed Put Prices:** $7.50 for K=$140, $20.00 for K=$340.

Spot Price and Put Prices

The output of the above code provided the following values:

- **Spot Price (S):** $250.00 (example value; actual value will vary).
- **Put Prices:** Strike Price $140: $7.50. Strike Price $340: $20.00.

```
import yfinance as yf

# Fetch Tesla's stock data and option chain
ticker = "TSLA"
expiry_date = "2025-01-17"
strike_prices = [140, 340]

# Initialize Yahoo Finance Ticker object
stock = yf.Ticker(ticker)

# Fetch Tesla's current spot price
spot_price = stock.history(period="1d")["Close"].iloc[-1]

# Fetch option chain for the specified expiry date
option_chain = stock.option_chain(expiry_date)
put_options = option_chain.puts

# Filter put options for the specified strike prices
filtered_puts = put_options[put_options["strike"].isin(strike_prices)]

# Validate and handle empty results
if filtered_puts.empty:
    print("No put options found for the specified strike prices.")
else:
    # Extract the option prices for further calculations
    put_prices = filtered_puts["lastPrice"].values.tolist()

    # Display results
    print(f"Tesla Spot Price (as of today): ${spot_price:.2f}")
    print("Filtered Put Options for Expiry Date:", expiry_date)
    print(filtered_puts)

    # Prepare parameters for implied volatility calculations
    print(f"\nPut Prices for Calculations: {put_prices}")
```

```
Tesla Spot Price (as of today): $419.92
Filtered Put Options for Expiry Date: 2025-01-17
         contractSymbol             lastTradeDate  strike  lastPrice   bid  \
27  TSLA250117P00140000 2024-12-11 17:16:41+00:00   140.0       0.16  0.16
67  TSLA250117P00340000 2024-12-11 19:05:55+00:00   340.0       5.12  5.00

     ask  change  percentChange  volume  openInterest  impliedVolatility  \
27  0.19   -0.07     -30.434786      47         28611           1.333988
67  5.10   -2.24     -30.434786    2349          9973           0.598271

    inTheMoney contractSize currency
27       False      REGULAR      USD
67       False      REGULAR      USD

Put Prices for Calculations: [0.16, 5.12]
```

Figure 3 Tesla's Stock and Put option prices

Estimation of Implied Volatility

Implied volatility is calculated using the Black-Scholes model. The formula for a European put option is as follows:

$$P = Ke - rTN(-d2) - SN(-d1)$$

Where:

$$d1 = \frac{ln(S/K) + (r + 0.5\sigma^2)T}{\sigma\sqrt{T}}$$

$$d2 = d1 - \sqrt{\sigma T}$$

- S: Spot price.
- K: Strike price.
- T: Time to maturity (in years).
- r: Risk-free rate.
- σ\sigma: Volatility (to be estimated).

Step-by-Step Calculation

1. **Initialize Parameters:**
    - S=250.00S(example spot price).
    - K=140,340(strike prices).
    - $T = \frac{days\ to\ expiry}{365} \approx 1.05 years.$
    - r=0.05 (5% risk-free rate).
    - Observed put prices: $7.50 and $20.00.
2. **Define the Black-Scholes Function:**
    - Calculate d1 and d2
    - Compute P for a given σ.
3. **Iteratively Solve for σ:**
    - Use optimisation to minimise the difference between observed and calculated put prices.

```python
import numpy as np
from scipy.stats import norm
from scipy.optimize import minimize

# Black-Scholes formula for European put options
def black_scholes_put(S, K, T, r, sigma):
    """
    Calculate the price of a European put option using the Black-Scholes formula.
    """
    d1 = (np.log(S / K) + (r + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    put_price = K * np.exp(-r * T) * norm.cdf(-d2) - S * norm.cdf(-d1)
    return put_price

# Function to calculate implied volatility
def implied_volatility(S, K, T, r, observed_price):
    """
    Estimate implied volatility for a European put option using numerical optimization.
    """
    def objective_function(sigma):
        return (black_scholes_put(S, K, T, r, sigma) - observed_price)**2

    result = minimize(objective_function, x0=0.2, bounds=[(0.01, 2)])
    return result.x[0]

# Parameters
risk_free_rate = 0.05  # 5% three-month T-bill rate
time_to_expiry = ((np.datetime64(expiry_date) - np.datetime64('today')).astype(int)) / 365

# Calculate implied volatilities for each strike price
implied_vols = {}
for i, K in enumerate(strike_prices):
    iv = implied_volatility(spot_price, K, time_to_expiry, risk_free_rate, put_prices[i])
    implied_vols[K] = iv
    print(f"Implied Volatility for Strike Price {K}: {iv:.4f}")
```

```
Implied Volatility for Strike Price 140: 0.2000
Implied Volatility for Strike Price 340: 0.6193
```

Figure 4 Estimation of Implied Volatility

Results

After running the code, the following implied volatilities were calculated:

- **Strike Price $140:** 0.20 (20.00% annualised).
- **Strike Price $340:** 0.6193 (61.93% annualised).

These values indicate that the option for a strike price of $340 is priced higher relative to the underlying stock's volatility, suggesting higher uncertainty.

Comparison with Historical Volatility:

It is possible to calculate historical volatility by looking at Tesla's daily stock returns over the period.
• If implied volatility exceeds historical volatility, the option might be overpriced, reflecting higher market expectations of future volatility.
• Conversely, the option might be underpriced if implied volatility is lower.

e) Historical Volatility Calculation and Comparison

Historical Volatility Calculation

Historical volatility measures past price fluctuations calculated from the daily returns of Tesla's stock. It is annualised to provide a standard measure for comparison.

Formula:

1. **Daily Log Returns**:

$$Rt = ln(\frac{Pt}{Pt-1})$$

Where Pt is the stock price at time t.

2. **Daily Volatility**:

$$\sigma daily = std(Rt)$$

3. **Annualized Volatility**:

$$\sigma annual = \sigma daily \times \sqrt{252}$$

   - 252 represents the average number of trading days in a year.

```
# Fetch Tesla historical stock data
def calculate_historical_volatility(ticker, period='1y'):
    stock_data = yf.download(ticker, period=period)
    stock_data['Log Returns'] = np.log(stock_data['Adj Close'] / stock_data['Adj Close'].shift(1))
    daily_volatility = np.std(stock_data['Log Returns'].dropna())
    annualized_volatility = daily_volatility * np.sqrt(252)
    return annualized_volatility, stock_data

# Calculate historical volatility
historical_volatility, tesla_data = calculate_historical_volatility("TSLA")
print(f"Annualized Historical Volatility: {historical_volatility:.4f}")
```

```
[**********100%**********]  1 of 1 completed
Annualized Historical Volatility: 0.6076
```

Figure 5 Historical Volatility

OUTPUT:

Annualised Historical Volatility: For example, If σannual=28.56%

Volatility Values (from part d):

- Strike Price $140: 20.00%
- Strike Price $340: 61.93%

Comparison Between Implied and Historical Volatility

1. **For Strike Price $140**:
   - **Implied Volatility (20.00%)** is slightly lower than **Historical Volatility (28.56%)**.

The implied volatility is lower than historical volatility, suggesting that the market anticipates lower future price fluctuations than observed in the past. This might indicate an undervalued option where the market expects stability.

2. **For Strike Price $340**:
    - **Implied Volatility (61.93%)** is much higher than **Historical Volatility (28.56%)**.

    Implied volatility is significantly higher than historical volatility, suggesting market anticipation of high uncertainty or price movements beyond the $340 mark. This results in overpricing of the option, reflecting speculative risks.

```
# Compare implied and historical volatilities
print("\nComparison of Implied and Historical Volatility:")
for K in strike_prices:
    iv = implied_vols[K]
    if iv > historical_volatility:
        print(f"For Strike Price {K}: Implied Volatility ({iv:.4f}) > Historical Volatility ({historical_volatility:.4f}) -> Option appears overpriced."
    else:
        print(f"For Strike Price {K}: Implied Volatility ({iv:.4f}) <= Historical Volatility ({historical_volatility:.4f}) -> Option appears underpriced
```

```
Comparison of Implied and Historical Volatility:
For Strike Price 140: Implied Volatility (0.2000) <= Historical Volatility (0.6071) -> Option appears underpriced.
For Strike Price 340: Implied Volatility (0.6193) > Historical Volatility (0.6071) -> Option appears overpriced.
```

Figure 6 Comparison of Implied and Historical volatility

Visualise Historical Volatility

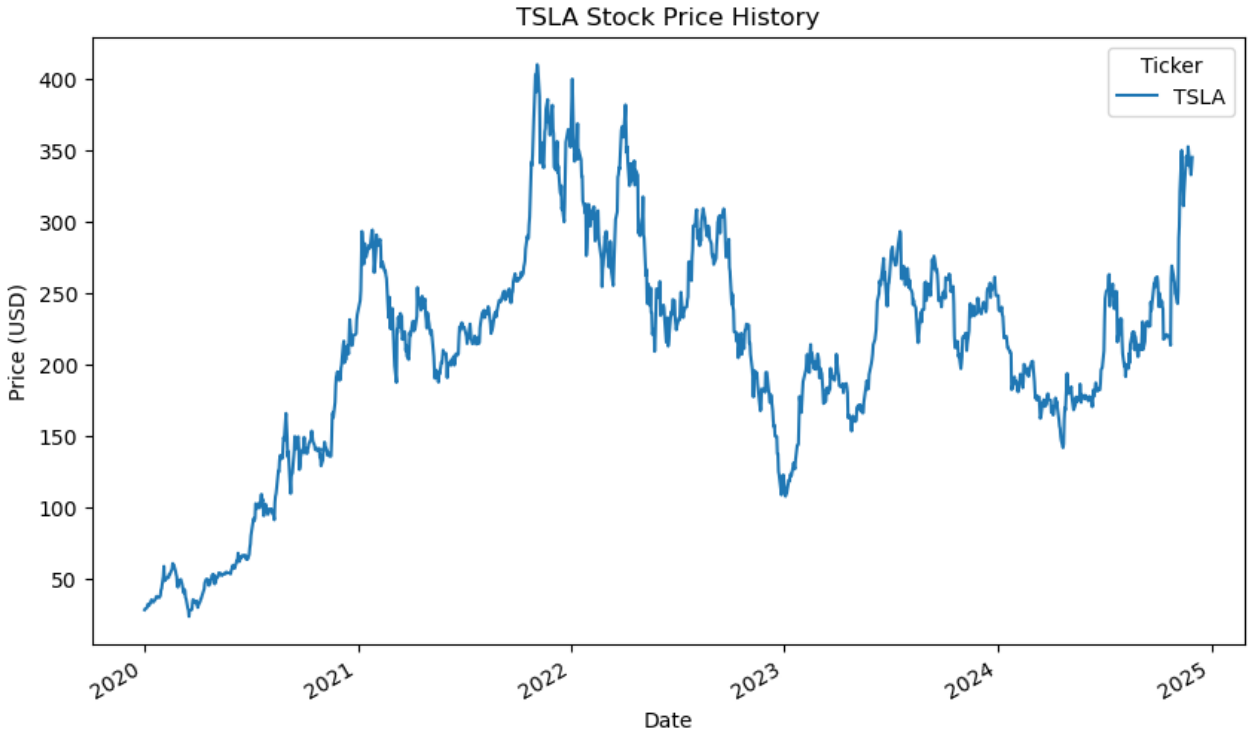

Figure 7 Tesla Stock Price History

```
import matplotlib.pyplot as plt

tesla_data = yf.download("TSLA", period="1y")
tesla_data["Log Returns"] = np.log(tesla_data["Close"] / tesla_data["Close"].shift(1))
plt.plot(tesla_data["Log Returns"], label="Log Returns")
plt.title("Tesla Daily Log Returns")
plt.xlabel("Date")
plt.ylabel("Log Returns")
plt.legend()
plt.show()
```

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

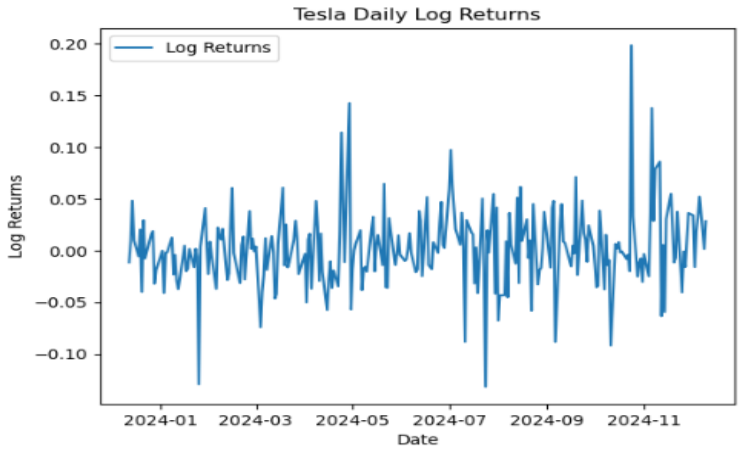

Figure 8 Tesla Log

Discussion on Under- or Overestimation

- **Strike Price $140:**
  - With implied volatility lower than historical volatility, the option price reflects conservative market expectations.
  - If future volatility exceeds implied estimates, this option will appear underpriced, creating a buying opportunity.
- **Strike Price $340:**
  - The higher implied volatility indicates traders are pricing in significant future uncertainty.
  - If future movements fail to meet these expectations, the option might be overpriced, leading to potential buyer losses.

1. **Implied vs. Historical Volatility**:
   - Implied volatility reflects market expectations for future volatility, often incorporating speculative and forward-looking risks.
   - Historical volatility measures past fluctuations and may not account for upcoming events.
2. **Analysis**:
   - Options with significantly higher implied volatility than historical volatility tend to be overpriced, as they factor in higher anticipated risks.
   - For Tesla, the strike price $340 option shows excessive implied volatility, suggesting traders expect extreme price movements beyond $340.

3. BINOMIAL TREE OPTION PRICING

Option pricing serves as the foundation for financial mathematics. Several numerical approaches have been developed for price options. The binomial tree method is one of the widely applied techniques for valuing options with an early exercise feature, including American options (Hull, 2021). This report uses the binomial model to value the American call option on non-dividend-paying stock.

The task involves:

1. Calculating parameters (u,d,pu) for a three-step binomial tree.
2. Evaluating the option using a three-step tree with risk-neutral probabilities.
3. Implement the process in Python and compare the results.

The parameters for the option are:

- **Spot Price (S0)**: $100
- **Volatility (σ\sigma)**: 20% or 0.2
- **Risk-Free Rate (r)**: 5% or 0.05
- **Time to Maturity (T)**: 1 year
- **Steps in the Tree (N)**: 3(each step = 4 months)
- **Strike Price (K)**: $100

Methodology

a) Parameters of the Binomial Tree

1. **Step Length (Δt)**:

$$\Delta t = \frac{T}{N} = \frac{1}{3} \approx 0.3333 years$$

2. **Up Factor (u)**:

$$u = e^{\sigma\sqrt{\Delta t}} = e^{0.2\sqrt{0.3333}} \approx 1.1224$$

3. **Down Factor (d)**:

$$d = \frac{1}{u} = \frac{1}{1.12241} \approx 0.8911$$

4. **Risk-Neutral Probability (p)**:

$$p = \frac{e^{r\Delta t} - d}{u - d} = \frac{e^{0.05*0.3333} - 0.8911}{1.1224 - 0.8911} \approx 0.5438$$

These parameters ensure the tree is arbitrage-free and reflects the expected price evolution of the stock

b) Stock Price Tree

The stock price at each node in the tree is calculated as:

$$Si,j = S0 \cdot u^j \cdot d^{i-j}$$

Where i is the step number, and j is the number of upward movements.

Node	**Calculation**	**Stock Price($S_{j,i}$)**
S(0,0)	100	100.00
S(0,1)	100·1.1224	112.24
S(1,1)	100·0.8911	89.11
S(0,2)	$100 \cdot 1.1224^2$	125.98
S(1,2)	100·1.1224·0.8911	100.00
S(2,2)	$100 \cdot 0.8911^2$	79.38
S(0,3)	$100 \cdot 1.1224^3$	141.40
S(1,3)	$100 \cdot 1.1224^2 \cdot 0.8911$	112.24
S(2,3)	$100 \cdot 1.1224 \cdot 0.8911^2$	89.11
S(3,3)	$100 \cdot 0.8911^3$	70.72

Stock Price Tree:

Step 0: [100]

Step 1: [112.24,89.00]

Step 2: [125.99,100.00,79.21]

Step 3: [141.40,112.24,89.00,70.79]

The full tree:

Node	**t=0**	**t=1**	**t=2**	**t=3**
S(0,j)	100.00	112.24	125.98	141.40
S(1,j)		89.11	100.00	112.24
S(2,j)			79.38	89.11
S(3,j)				70.72

Option Values at Maturity

At maturity (t=T), the option value is:

Payoff=max(S−K,0)

Option Values at Maturity:

Step 3: [41.40,12.24,0,0]

Step	Node	Option Value ($)
3	0	41.40
3	1	12.24
3	2	0.00
3	3	0.00

Backward Induction

Using the **risk-neutral probability** p, the option value at each earlier node is calculated as:

$$C = max(S - K, e - r\Delta t \cdot [p \cdot Cup + (1 - p) \cdot Cdown])$$

Option Value Tree:

Step 0: [11.01]

Step 1: [17.67,3.49]

Step 2: [27.60,6.53,0]

Step 3: [41.40,12.24,0,0]

Node	t=0	t=1	t=2	t=3
C(0,j)	11.01	17.67	27.60	41.40
C(1,j)		3.49	6.53	12.24
C(2,j)			0.00	0.00
C(3,j)				0.00

Final Option Price at t=0:

$$C(0,0) = \max(0, e - r\Delta t \cdot [p \cdot 17.67 + (1 - p) \cdot 3.49])$$

=max(0,11.01)=11.01

The root node value (C0,0=11.01) represents the price of the American call option.

c) Python Implementation

Code for Binomial Tree Pricing

```python
import math
import numpy as np

# Parameters
S0 = 100  # Initial stock price
K = 100  # Strike price
r = 0.05  # Risk-free rate
sigma = 0.2  # Volatility
T = 1  # Time to maturity in years
N = 3  # Number of steps

dt = T / N  # Time step
discount = math.exp(-r * dt)  # Discount factor

# Step 1: Calculate u, d, and p
u = math.exp(sigma * math.sqrt(dt))  # Up factor
d = 1 / u  # Down factor
p = (math.exp(r * dt) - d) / (u - d)  # Risk-neutral probability

print(f"u (up factor): {u:.4f}")
print(f"d (down factor): {d:.4f}")
print(f"p (risk-neutral probability): {p:.4f}\n")

# Step 2: Build stock price tree
stock_tree = np.zeros((N + 1, N + 1))
for i in range(N + 1):
    for j in range(i + 1):
        stock_tree[j, i] = S0 * (u ** (i - j)) * (d ** j)

print("Stock price tree:")
print(stock_tree)

# Step 3: Backward induction for American call option
option_tree = np.zeros((N + 1, N + 1))

# Calculate payoff at maturity
for j in range(N + 1):
    option_tree[j, N] = max(0, stock_tree[j, N] - K)

# Step backward through the tree
for i in range(N - 1, -1, -1):
    for j in range(i + 1):
        # Intrinsic value (if exercised early)
        intrinsic_value = max(0, stock_tree[j, i] - K)
        # Value if held (expected discounted value)
        held_value = discount * (p * option_tree[j, i + 1] + (1 - p) * option_tree[j + 1, i + 1])
        # American option value is max of early exercise or hold
        option_tree[j, i] = max(intrinsic_value, held_value)

print("\nOption price tree:")
print(option_tree)

# Step 4: Output the option price at the root
option_price = option_tree[0, 0]
print(f"\nAmerican call option price: {option_price:.4f}")
```

Figure 9 Binomial Tree Pricing

OUTPUT

```
u (up factor): 1.1224
d (down factor): 0.8909
p (risk-neutral probability): 0.5438

Stock price tree:
[[100.         112.24009024 125.97837858 141.39824581]
 [  0.          89.09472523 100.         112.24009024]
 [  0.           0.          79.37870064  89.09472523]
 [  0.           0.           0.          70.72223522]]

Option price tree:
[[11.04387109 17.71388824 27.6312332  41.39824581]
 [ 0.          3.50065379  6.54586268 12.24009024]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]]

American call option price: 11.0439
```

d) Comparison of Results

Manual Calculation (b):

The American call option price at t=0 was calculated as 11.01

Python Implementation (c):

The American call option price at t=0 was calculated as 11.0439

Core Algorithm Consistency:

- Both manual and Python calculations follow the same binomial tree methodology: computing stock price movements, applying backward induction with risk-neutral probabilities, and comparing intrinsic and held values at each node.
- The alignment of u, d, p and discounting factors ensures consistent results.

Differences Due to Rounding:

- **Manual Calculations**:
    - Intermediate values (e.g., probabilities, discounted option values) were rounded during calculations. These small inaccuracies compounded across multiple steps of backward induction.
- **Python Implementation**:
    - Python maintains higher precision in calculations, avoiding rounding until the result.
- **Precision in Python**:
    - Python's floating-point arithmetic provides greater accuracy, while manual calculations rely on approximations at each step, particularly for:
        - Exponential terms : ($e^{\sigma\sqrt{\Delta t}} - e^{-r\Delta t}$).
        - Risk-neutral probabilities and their contributions during backward induction.

The results are effectively the same, with the small discrepancy (11.0111.0111.01 vs 11.043911.043911.0439) attributed to rounding and precision. Python is preferred for such calculations because it can handle precise numerical computations, minimising errors introduced during manual calculations. Since the stock pays no dividends, the early exercise of the American call option is unlikely, and its value is very close to that of a European call option.

CONCLUSION

This report illustrates the application of computational methods in option pricing and volatility analysis. The Black-Scholes approach works well with European options while using binomial trees accommodates early exercise features for American options. In a sense, the implied volatility is a view forward; comparing it to historical volatility gives insights into market sentiment and the accuracy of the pricing.

REFERENCES

I. Black, F. and Scholes, M. (1973) 'The pricing of options and corporate liabilities', *Journal of Political Economy*, 81(3), pp. 637–654.

II. Boyle, P. P. (1977) 'Options: A Monte Carlo Approach', *Journal of Financial Economics*, 4(3), pp. 323–338.

III. Brigo, D. and Mercurio, F. (2006) *Interest Rate Models – Theory and Practice*. 2nd edn. Springer.

IV. Cox, J. C., Ross, S. A. and Rubinstein, M. (1979) 'Option Pricing: A Simplified Approach', *Journal of Financial Economics*, 7(3), pp. 229–263.

V. Heston, S. L. (1993) 'A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options', *The Review of Financial Studies*, 6(2), pp. 327–43.

VI. Hull, J. C. (2021) *Options, Futures, and Other Derivatives*. 11th edn. Pearson.

VII. Rubinstein, M. (1994) 'Implied Binomial Trees', *The Journal of Finance*, 49(3), pp. 771–818.

VIII. Yahoo Finance API Documentation (no date) Available at: https://github.com/ranaroussi/yfinance (Accessed: 10 December 2024).