Module Title: Computational Methods for Finance

Module Code: 7FNCE041W

MSc Fintech and Business Analytics (Core), Semester 1, 2023/2024

FROM THEORY TO PRACTICE: PYTHON TOOLS FOR OPTIONS AND MARKET ANALYSIS

Student ID: 20196958

Student Name: Sarika Patel

Words count: 2458.

TABLE OF CONTENTS

CONTENTS

1.EXECUTIVE SUMMARY

This research paper provides a thorough examination of quantitative finance by diving into the areas of binomial tree option pricing, volatility analysis, and Python libraries. The three separate portions of the course of study each add to an in-depth understanding of computer modelling and financial analytics.

In the First section we examine the fundamental Python libraries—numpy, pandas, matplotlib, and yfinance. The study begins by explaining what a Python "library" is, why it's important, and what happens if you don't use it. The salient characteristics of each library are outlined, similarities are noted, and applications are differentiated in a comparative study that follows. The practical aspect is demonstrated via Jupyter notebook, which highlight real-world uses with pertinent formulae.

The second section inquiries into every aspect of volatility comparing implied and historical volatility. It presents a method for evaluating implied volatility in non-dividend European call options called the Newton-Raphson iteration. The tool goes so far as to use Yahoo Finance data to estimate the implied volatility of an Amazon option.

In the last part, binomial tree option pricing is examined in the context of a stock that does not pay dividends. It starts by figuring out the parameters for a four-step tree: u, d, and risk-neutral probability. Next, using the obtained risk-neutral probability, the European call option is priced.

This research study gives thorough understanding of quantitative finance is made possible by the in-depth analysis and practical examples, making this an invaluable tool for academics and business executives alike.

Note: Share price Values are in $.

GitHub Link for Python Jupyter Notebook Codes: https://github.com/xxsarikapatel/CMF-Coursework/blob/main/okay%20Final%20coursework%20.ipynb

GitHub Link updated here of python jupyter notebook for more clarity and visibility of codes as screenshots may not be more clearly visible at some point of time.

2.PYTHON LIBRARIES

Python is a very popular programming language because of its many libraries that improve functionality and its adaptability. The objective of this study paper is to objectively evaluate matplotlib, yfinance, pandas, and numpy—four crucial Python libraries. In scientific computing, data analysis, visualisation, and financial data retrieval, these libraries are essential.

What is a Library in python?

In Python, a library is an assemblage of pre-written code that makes jobs easier and enables programmers to reuse code for shared capabilities. Libraries include functions and modules that may be added to Python scripts, saving developers from having to start from scratch when performing complicated processes.

How to use python Library?

A Python library must first be imported into the script using the import statement to be used. Example as below.

```
# Importing necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import yfinance as yf
```

Functions and modules in the library are available once they are imported, which streamlines code development and lessens the need for human coding.

What will happen without using python library?

To provide the required functionality, developers might have to create a significant amount of custom code if they don't use a library while running the needed command. This increases the time and effort required, increases the risk of mistakes, and reduces the maintainability of the code. Libraries improve the efficiency and dependability of code by encapsulating best practices and optimisations.

Critique of python libraries

(1). NumPy: Numerical computing Library

Supporting massive, multi-dimensional arrays and matrices, numpy is a core Python module for numerical calculations. Numerical operations are considerably improved by its optimised functions. For newcomers, though, the learning curve might be rather high.

Features:

- effective management of matrices and multidimensional arrays.
- array operations using mathematical functions (e.g., mean, sum, dot product).
- Broadcasting functionalities to manipulate arrays.
- Integration with additional languages and libraries (like C/C++).

Application:

```
# Example using numpy
arr = np.array([1, 2, 3, 4, 5])
mean = np.mean(arr)
print("Mean:", mean)
```

```
Mean: 3.0
```

(2). Pandas: Data Analysis Library

Pandas is an effective library for working with and analysing data. It presents data structures like Series and Data Frame, making operations like data cleansing, transformation, and exploration easier. Data scientists find it invaluable due to its vast capabilities. However, managing big databases might provide performance issues.

Features:

- For structured data, use DataFrame; for one-dimensional labelled arrays, use Series.
- Tools for cleaning, restructuring, and combining datasets during data manipulation.
- strong indexing and choosing ability.
- Excel and database integration.

Application:

```python
# Example using pandas
data = {'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [25, 30, 22]}
df = pd.DataFrame(data)
print(df)
```

```
      Name  Age
0    Alice   25
1      Bob   30
2  Charlie   22
```

(3). Matplotlib: Data Visualization Library

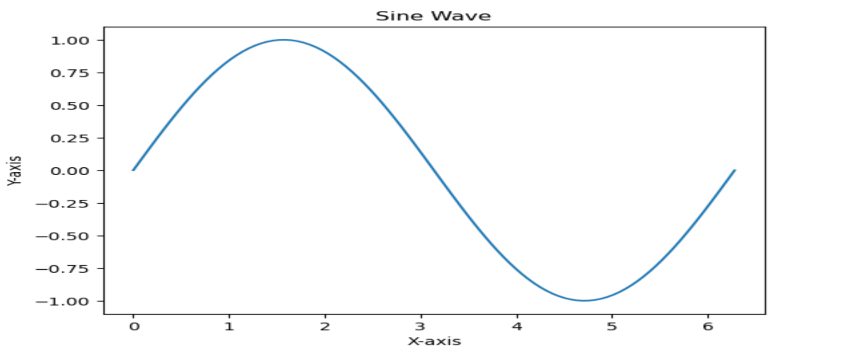
Python visualisations may be made static, animated, or interactive with the help of the flexible matplotlib module. Although it allows for freedom in customising the storyline, its complicated syntax may draw criticism. Data visualisation capabilities are improved by its interaction with Pandas.

Features:

- Entire plotting library for interactive, animated, and static visualisations.
- support for several styles, customisation choices, and narrative kinds.
- seamless data structure integration with pandas and numpy.
- Plots and visualisations fit for publication.

Application:

```python
# Example using matplotlib
x = np.linspace(0, 2 * np.pi, 100)
y = np.sin(x)
plt.plot(x, y)
plt.title('Sine Wave')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()
```



(4). yfinance: Financial Data Retrieval Library

yfinance is designed to retrieve financial data from Yahoo Finance, namely stock market data. It simplifies the process of obtaining current and historical data. Nevertheless, disparities in data accuracy might result from its reliance on other data sources.

Features:

- A dedicated library designed to retrieve financial information from Yahoo Finance.

- obtaining current and historical stock market data.
- support for a variety of financial products, including currencies, equities, and ETFs.
- Data analysis made easier with panda's integration.

Application:

```
# Importing yfinance with an alias (commonly used alias is yf)
import yfinance as yf

# Example using yfinance
ticker = 'AAPL'
data = yf.download(ticker, start='2022-01-01', end='2023-01-01')
print(data.head())
```

```
[*********************100%%**********************]  1 of 1 completed
                  Open        High         Low       Close   Adj Close  \
Date
2022-01-03  177.830002  182.880005  177.710007  182.009995  179.953888
2022-01-04  182.630005  182.940002  179.119995  179.699997  177.669998
2022-01-05  179.610001  180.169998  174.639999  174.919998  172.943985
2022-01-06  172.699997  175.300003  171.639999  172.000000  170.056961
2022-01-07  172.889999  174.139999  171.029999  172.169998  170.225037

               Volume
Date
2022-01-03  104487900
2022-01-04   99310400
2022-01-05   94537600
2022-01-06   96904000
2022-01-07   86709100
```

Comparative Analysis of Python Libraries

Similarities	Differences
Integration with pandas and numpy: Matplotlib easily integrates with both libraries for data visualisation. Numpy and pandas are frequently used together for effective numerical calculations.	**Specialisation:** numpy and pandas are more general-purpose libraries for numerical computation and data manipulation, whereas yfinance is specifically designed for financial data retrieval
Capabilities for processing and modifying data: Although they have distinct goals, pandas and yfinance both offer tools for this.	**Use Cases:** The main applications for numpy are mathematical operations; pandas is used for data analysis; matplotlib is used for data visualisation; and yfinance is used to get financial data.
Effective Data Management: One similarity between pandas and numpy is their focus on effective data management.	**Data Structures:** Pandas provides higher-level data structures like DataFrames and Series,

	whereas numpy works with arrays. Contrarily, yfinance works with financial data structures.

3. VOLATILITY

There are differences between historical volatility and implied volatility, each with different applications and methods. Implied volatility, which is particularly pertinent in the context of options trading, measures market expectations on future price swings, whereas historical volatility represents actual price movements and offers a historical perspective on an asset's risk.

1.HISTORICAL VOLATILITY

Based on past price data, historical volatility is a statistical measure of the dispersion of returns over a certain time period. It provides information about a financial instrument's price variability by quantifying its historical price changes.

Measurement: Historical σ is commonly computed by taking the standard deviation of logarithmic returns during a certain time interval N.

$$\sigma = \sqrt{\frac{T}{n-1}\sum_{i=1}^{n}(R_i - \bar{R})^2}$$

Where,

n = Historical volatility period

T = No. of periods

2.IMPLIED VOLATILITY

The implied volatility of a financial instrument is the market's prediction of its future volatility. It is calculated using option pricing and represents what market players believe will likely be the future range of prices.

Measurement: It is possible to solve for implied volatility by rearranging the Black-Scholes model, which is a popular model for pricing options.

Formula: $\mathbf{C = SN\,(d_1) - N\,(d_2)\,Ke^{-rt}}$

Where,

- C is the Option Premium,
- S is the price of the stock
- K is the Strike price
- r is the risk-free rate
- t is the time to maturity
- e is the exponential term

Comparative Analysis of Python Libraries

Similarities	Differences
Measurement of Volatility: The price variability of an asset is expressed in terms of both implied and historical volatility.	**Information source:** Historical volatility, which represents real market fluctuations, is computed using historical price data. Implied volatility, on the other hand, is derived from option pricing and represents market expectations.
Statistical in Nature: They both come from statistical analysis that shed light on the possible risk and unpredictability connected to a certain asset.	**Method of Calculation:** Implied volatility is frequently generated using intricate mathematical models, such as the Black-Scholes model, whereas historical volatility is computed directly from historical price data.

Newton-Raphson iteration

An iterative technique for estimating the implied volatility of a financial option is the Newton-Raphson iteration. The implied volatility, or, in the context of a non-dividend European call option is the volatility parameter that, when applied to an option pricing model (such the Black-Scholes model), yields a theoretical option price that is equal to the market price.

The following is the Newton-Raphson iteration formula for calculating implied volatility:

Formula: $\sigma n + 1 = \sigma n - f(\sigma n)/f'(\sigma n)$

- σn+1 is the updated estimate of implied volatility.
- σn is the current estimate of implied volatility.
- $f(\sigma)$ is the difference between the market option price and the theoretical option price.
- $f'(\sigma)$ is the derivative of $f(\sigma)$ with respect to σ.

Implementation in python

The Black-Scholes European call option pricing is determined by the black Scholes call function.

The Newton-Raphson iteration is used by the implied volatility newton Raphson function to estimate implied volatility.

The calculated implied volatility is the outcome mentioned below.

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

def black_scholes_call(S, X, T, r, sigma, q=0):
    """
    Calculate the Black-Scholes European call option price.
    """
    d1 = (np.log(S / X) + (r - q + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    call_price = S * np.exp(-q * T) * norm.cdf(d1) - X * np.exp(-r * T) * norm.cdf(d2)
    return call_price

def implied_volatility_newton_raphson(option_price, S, X, T, r, initial_guess=0.2, tol=1e-6, max_iter=100):
    """
    Estimate implied volatility using Newton-Raphson iteration.
    """
    sigma = initial_guess

    for _ in range(max_iter):
        d1 = (np.log(S / X) + (r + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
        d2 = d1 - sigma * np.sqrt(T)

        # Calculate Black-Scholes option price and its derivative
        f = black_scholes_call(S, X, T, r, sigma) - option_price
        f_prime = (S * np.exp(-r * T) * norm.pdf(d1) * np.sqrt(T))

        # Update sigma using Newton-Raphson iteration formula
        sigma -= f / f_prime

        # Check for convergence
        if abs(f) < tol:
            break

    return sigma

# Example usage:
market_price = 10.0
stock_price = 100.0
strike_price = 100.0
time_to_maturity = 1.0
interest_rate = 0.05

estimated_volatility = implied_volatility_newton_raphson(market_price, stock_price, strike_price, time_to_maturity, interest_
print("Estimated Implied Volatility:", estimated_volatility)
```

```
Estimated Implied Volatility: 0.18797164966818192
```

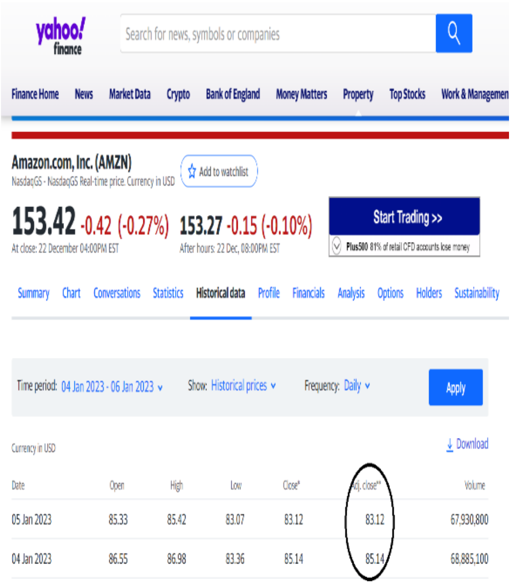
Steps in python begins with function definition where in black Scholes call calculates European call option price and implied volatility newton Raphson calculates implied volatility.

Then in second step defined parameters such as (S,K,T,r,and q) which computes call option price while using d1 and d2 values.

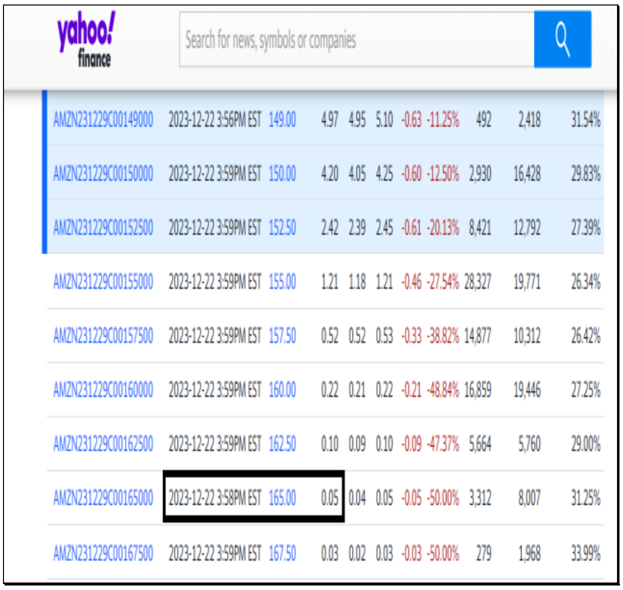
Then further calculated implied volatility using newton iteration with parameter initial guess and max.iteration.and then used example where stock and strike price given and then given print command to get estimated implied volatility.

Downloading Amazon’s Data from Yahoo Finance

Spot and call price via Yahoo Finance



Call Price at Strike Price $165 from Yahoo Finance is $0.05



AMZN231229C00149000	2023-12-22 3:56PM EST	149.00	4.97	4.95	5.10	-0.63	-11.25%	492	2,418	31.54%
AMZN231229C00150000	2023-12-22 3:59PM EST	150.00	4.20	4.05	4.25	-0.60	-12.50%	2,930	16,428	29.83%
AMZN231229C00152500	2023-12-22 3:59PM EST	152.50	2.42	2.39	2.45	-0.61	-20.13%	8,421	12,792	27.39%
AMZN231229C00155000	2023-12-22 3:59PM EST	155.00	1.21	1.18	1.21	-0.46	-27.54%	28,327	19,771	26.34%
AMZN231229C00157500	2023-12-22 3:59PM EST	157.50	0.52	0.52	0.53	-0.33	-38.82%	14,877	10,312	26.42%
AMZN231229C00160000	2023-12-22 3:59PM EST	160.00	0.22	0.21	0.22	-0.21	-48.84%	16,859	19,446	27.25%
AMZN231229C00162500	2023-12-22 3:59PM EST	162.50	0.10	0.09	0.10	-0.09	-47.37%	5,664	5,760	29.00%
AMZN231229C00165000	2023-12-22 3:58PM EST	165.00	0.05	0.04	0.05	-0.05	-50.00%	3,312	8,007	31.25%
AMZN231229C00167500	2023-12-22 3:59PM EST	167.50	0.03	0.02	0.03	-0.03	-50.00%	279	1,968	33.99%

Importing Amazon's Data in python via Yfinance

In [19]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as si
import yfinance as yf
import os
```

In [20]:
```python
Amazon = yf.download("AMZN", start="2022-12-05", end="2023-01-05")
Amazon.tail()
```

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

Out[20]:

Date	Open	High	Low	Close	Adj Close	Volume
2022-12-28	82.800003	83.480003	81.690002	81.820000	81.820000	58228600
2022-12-29	82.870003	84.550003	82.550003	84.180000	84.180000	54995900
2022-12-30	83.120003	84.050003	82.470001	84.000000	84.000000	62401200
2023-01-03	85.459999	86.959999	84.209999	85.820000	85.820000	76706000
2023-01-04	86.550003	86.980003	83.360001	85.139999	85.139999	68885100

Obtain Spot price& call price in python.

```python
# Step 2: Obtain spot price and call price
spot_price = Amazon['Close'][-1]
print("\nSpot Price:", spot_price)
```

```
Spot Price:  85.13999938964844
```

```python
import numpy as np
import scipy.stats as si

# Function to calculate European call option price using Black-Scholes
def black_scholes_call(S, X, T, r, sigma):
    d1 = (np.log(S / X) + (r + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    call_price = S * si.norm.cdf(d1) - X * np.exp(-r * T) * si.norm.cdf(d2)
    return call_price

# Given parameters
spot_price = 85.14
strike_price = 165
time_to_maturity = 1
risk_free_rate = 0.05
volatility = 0.7

# Calculate the European call option price
call_option_price = black_scholes_call(spot_price, strike_price, time_to_maturity, risk_free_rate, volatility)

print("European Call Option Price:", call_option_price)
```

```
European Call Option Price: 8.2143164577924
```

Estimated implied volatility using the Black-Scholes model.

```
In [43]: initial_guess = 0.3  # Adjust the initial guess
```

```
In [44]: tol = 1e-8  # Increase the tolerance
         max_iter = 200  # Increase the maximum number of iterations
```

```
In [45]: import numpy as np
         import scipy.stats as si

         def black_scholes_call(S, X, T, r, sigma, q=0):
             """
             Calculate the Black-Scholes European call option price.
             """
             d1 = (np.log(S / X) + (r - q + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
             d2 = d1 - sigma * np.sqrt(T)
             call_price = S * np.exp(-q * T) * si.norm.cdf(d1) - X * np.exp(-r * T) * si.norm.cdf(d2)
             return call_price

         def implied_volatility_newton_raphson(option_price, S, X, T, r, q=0, initial_guess=0.3, tol=1e-8, max_iter=200):
             """
             Estimate implied volatility using Newton-Raphson iteration.
             """
             sigma = initial_guess

             for _ in range(max_iter):
                 d1 = (np.log(S / X) + (r - q + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
                 d2 = d1 - sigma * np.sqrt(T)

                 f = black_scholes_call(S, X, T, r, sigma, q) - option_price
                 f_prime = (S * np.exp(-q * T) * si.norm.pdf(d1) * np.sqrt(T))

                 # Check for very small denominator to avoid division by zero
                 if abs(f_prime) < 1e-10:
                     sigma = np.nan
                     break

                 sigma -= f / f_prime

                 if abs(f) < tol:
                     break

                 print("Iteration:", _, "f:", f, "f_prime:", f_prime)

             return sigma

         # Given parameters
         spot_price = 85.14
         strike_price = 165
         time_to_maturity = 1.0
         risk_free_rate = 0.05

         # Example call option price
         option_price = 10.0

         # Estimate implied volatility
         estimated_volatility = implied_volatility_newton_raphson(option_price, spot_price, strike_price, time_to_maturity, risk_free

         print("Estimated Implied Volatility:", estimated_volatility)

         Iteration: 0 f: -9.737152669843473 f_prime: 5.706021985270308
         Iteration: 1 f: 39.51944964644595 f_prime: 26.615121425112232
         Iteration: 2 f: -6.502866656411248 f_prime: 22.413947731534975
         Iteration: 3 f: 1.6670081758640936 f_prime: 31.974446147080396
         Iteration: 4 f: 0.023618460842235527 f_prime: 31.027449018614025
         Iteration: 5 f: 5.973043784024412e-06 f_prime: 31.011745962492597
         Estimated Implied Volatility: 0.7588488882991238
```

Estimated Implied Volatility for amazon’s stock is 78%.

Python codes for comparison implied and historical volatility are as below.

```
In [60]:
import numpy as np
import pandas as pd
import yfinance as yf
from scipy.stats import norm
from scipy.optimize import root_scalar

# Download historical stock price data for Amazon
ticker = "AMZN"
start_date = "2022-01-01"
end_date = "2023-01-01"
stock_data = yf.download(ticker, start=start_date, end=end_date)

# Calculate daily returns
stock_data['Daily_Return'] = stock_data['Adj Close'].pct_change()

# Calculate annualized historical volatility
annual_volatility = np.sqrt(252) * stock_data['Daily_Return'].std()

print("Annual Historical Volatility:", annual_volatility)

# Function to calculate Black-Scholes call option price
def black_scholes_call(S, X, T, r, sigma):
    d1 = (np.log(S / X) + (r + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    call_price = S * norm.cdf(d1) - X * np.exp(-r * T) * norm.cdf(d2)
    return call_price

# Function to calculate implied volatility using bisect method
def calculate_implied_volatility(observed_price, S, X, T, r):
    # Define the implied volatility function
    def implied_volatility(sigma, observed_price, S, X, T, r):
        return black_scholes_call(S, X, T, r, sigma) - observed_price

    # Use bisect method to find implied volatility
    bracket = [0.001, 1.0]
    result = root_scalar(implied_volatility, bracket=bracket, args=(observed_price, S, X, T, r), method="bisect")

    if result.converged:
        implied_volatility = result.root
        return implied_volatility
    else:
        raise RuntimeError("Failed to converge to a solution.")

# Assuming you have the observed market price of the call option
observed_option_price = 10.25

# Calculate implied volatility
implied_volatility = calculate_implied_volatility(observed_option_price, stock_data['Adj Close'].iloc[-1], 165, 1, 0.05)

print("Implied Volatility:", implied_volatility)

# Compare implied and historical volatilities
if implied_volatility > annual_volatility:
    print("Implied volatility is higher than historical volatility.")
elif implied_volatility < annual_volatility:
    print("Implied volatility is lower than historical volatility.")
else:
    print("Implied volatility is equal to historical volatility.")

[*********************100%***********************]  1 of 1 completed
Annual Historical Volatility: 0.5011771055942209
Implied Volatility: 0.7793909383940099
Implied volatility is higher than historical volatility.
```

The implied volatility (78%) is more than the historical volatility (50%), as the comparison demonstrates. Indicated volatility indicates that the market anticipates more price movements in the future than what has historically been seen. It is higher when compared to historical volatility.

If implied volatility exceeds historical volatility, option prices may rise because of the market's perception of increased risk or uncertainty in the future. Given that the market is pricing in a greater degree of volatility than has traditionally happened, the option price in this instance may be viewed as possibly overvalued.

4.BIONOMIAL TREE OPTION PRICING

(A)CALCULATE PARAMETERS

Parameters	**Notes**
Stock price	Non dividend paying
Strike price (S)	$100
Risk free Rate of Interest'[r]	5% per annum
Volatility (σ)	20%
Time to Maturity(T)	1 year
No.of time steps (n)	4

The parameters u, d, and the risk-neutral probability p are utilised in a binomial option pricing model to simulate the movement of the stock price across discrete time steps. The following are the correlations between these parameters:

The likelihood of an upward movement is represented by p, which is the risk-neutral probability.The anticipated return in the risk-neutral environment is assumed to be equal to the risk-free rate, which leads to the formulae for u, d, and p.

The formulae are as follows:

Calculate Δt

Δt =T/n, =1/4 =0.25

Calculate *u* and *d*:

$$u = e^{(r-\sigma^2/2)\Delta t+\sigma\sqrt{\Delta t}}$$
$$d = e^{(r-\sigma^2/2)\Delta t-\sigma\sqrt{\Delta t}}$$

While substituting the given values:

$$u = e^{(0.05-0.2^2/2)\cdot\frac{1}{4}+0.2\cdot\sqrt{\frac{1}{4}}}$$
$$d = e^{(0.05-0.2^2/2)\cdot\frac{1}{4}-0.2\cdot\sqrt{\frac{1}{4}}}$$

Simplifying these expressions has yielded below values:

$$u \approx 1.1135$$
$$d \approx 0.8981$$

Calculate *p* value:

$$p = \frac{e^{r\Delta t} - d}{u - d}$$

Substituting the values:

$$p = \frac{e^{0.05 \cdot \frac{1}{4}} - 0.8981}{1.1135 - 0.8981}$$

Simplifying these expressions has yielded below values:

p=0.54

By manual calculation found Up factor (u): 1.11, Down factor (d): 0.90 and

Risk-neutral probability (p): 0.54.

(B) MANUAL CALCULATION OF EUROPEAN CALL OPTION

Parameters	**Notes**
Strike price (S)	\$100
p	0.54
u	1.11
d	0.9
Stock price as positions	
A=So	\$100
B=So*u	\$110.52
C=So*d	\$90.48
D=So*u*u	\$122.14
E=So*u*d	\$100
F=So*d*d	\$81.87

Formula for European call option

C=e^(-rT) [p fu +q fd]

Payoff is [max (Si-K,0)]

Where,

T is Time to maturity
r is Risk free rate
p is the risk neutral probability
q=0.46 (q=1-p)
fu is payoff if the stock goes up.
fd is payoff if the stock goes down.

Let's now calculate the value of call option at node B.

fu =max (B - K,0) =max (110.52-100,0) =$10.52.

fd =max(C- K,0) =max (90.48-100,0) =$0.

C =e^(-rT) [p fu +q fd]

= e^(-5%*3/12) [0.54*10.52 + 0.46*0]

= $5.41

value of call option at node C.

fu =max (C - K,0) =max (90.48-100,0) =$0

fd =max (D - K,0) =max (122.14-100,0) =$22.14.

C =e^(-rT) [p fu +q fd]

= e^(-5%*3/12) [0.54*0 + 0.46*22.14]

= $11.26

value of call option at node D.

fu =max (D - K,0) =max (122.14-100,0) =$22.14

fd =max (E - K,0) =max (100-100,0) =$0

C =e^(-rT) [p fu +q fd]

= e^(-5%*3/12) [0.54*22.14+ 0.46*0]

= $11.26

value of call option at node E.

fu =max (E - K,0) =max (100-100,0) =$0

fd =max (F - K,0) =max (100-100,0) =$0

C =e^(-rT) [p fu +q fd]

= e^(-5%*3/12) [0.54*0+ 0.46*0]

= $0

Hence the value of call option at node B=5.41, C=11.26, D=11.26, E=0 respectively.

Now value of the call option @ node A

Call option value for up = $5.41

Call option value for down = $11.26

C = e^(-5%*3/12) [0.54*5.41+ 0.46*11.26]

= $10.55

The value of the European Call Option is $ 10.55

(C) PYTHON CODE FOR VALUATION OF EUROPEAN CALL OPTION

1.Parameters: Imported math library and **defined parameters** (volatility,S,K,T,N,u,d,p,q,r etc .) in jupyter notebook and screenshot attached herewith.

```
import math

# Given parameters
spot_price = 100  # Spot price of the stock
strike_price = 100  # Strike price of the European call option
num_periods = 4  # Number of periods in the binomial tree
u = 1.11  # Up factor
d = 0.9  # Down factor
p = 0.54  # Risk-neutral probability
q = 1 - p  # Probability of a down movement
risk_free_rate = 0.05  # Risk-free interest rate
time_to_maturity = 1  # Time to maturity in years
```

2.Function definition: In next step defined a function i.e., European call option value which c alculates the option value at each node using the binomial tree approach.

```python
# Function to calculate the option value at each node
def european_call_option_value(spot, u, d, p, q, strike, num_periods):
    # Calculate the time step
    delta_t = time_to_maturity / num_periods

    # Calculate terminal stock prices
    terminal_stock_prices = [spot * (u ** up_moves) * (d ** (num_periods - up_moves))
                             for up_moves in range(num_periods + 1)]

    # Calculate terminal option values
    terminal_option_values = [max(0, price - strike) for price in terminal_stock_prices]

    # Backward induction to calculate the option value at each node
    for step in range(num_periods - 1, -1, -1):
        terminal_option_values = [
            math.exp(-risk_free_rate * delta_t) *
            (p * terminal_option_values[up_moves + 1] +
             q * terminal_option_values[up_moves])
            for up_moves in range(step + 1)
        ]

    # Return the option value at the initial node
    return terminal_option_values[0]
```

3.Calculated Loop: In this I have used backward induction loop to calculate option values at each node, it has been initiated from terminal node and moved backward to the initial node.

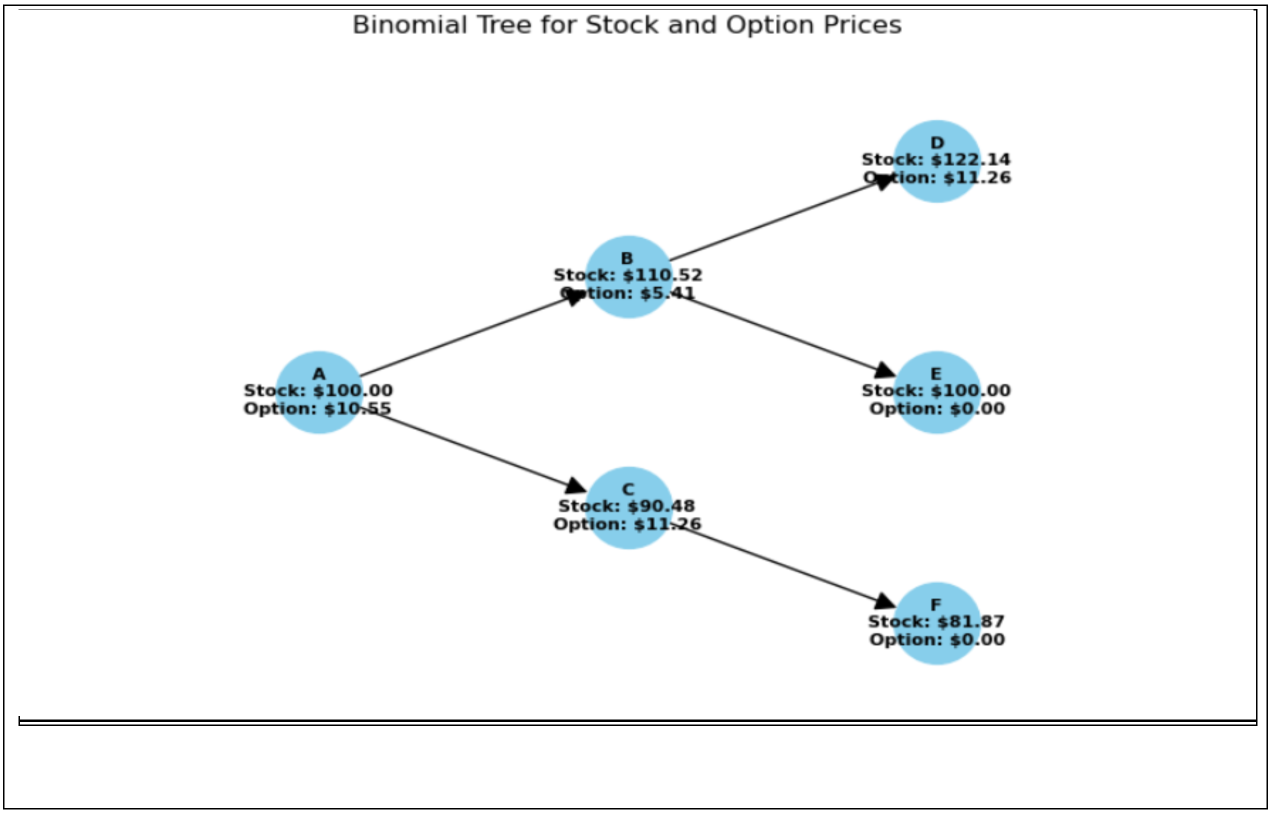
```python
# Calculate the value of the European call option
european_call_value = european_call_option_value(
    spot_price, u, d, p, q, strike_price, num_periods
)
```

4.Print Result: Print the final European call option value

```python
# Print the results
print(f"European call option value: ${european_call_value:.2f}")
```

```
European call option value: $10.55
```

European call option value is $**10.55** at spot price is $100, and its volatility is 20%, the risk-free rate is 5% per annum, the time to maturity is one year with four 3-month periods.



This is the four-step binomial tree chart for the specified stock prices. The lines show the evolution from one time step to the next, and each node represents a potential stock price at that moment.

(D) COMPARISON OF B AND C

Using both approaches, the European Call option has a value of $10.55.

To determine the likelihood of an upward or downward shift in the stock price, the values of u, d, p, and q were first calculated manually. Next, each node computes the option price formula.

The primary distinction between the two approaches is that while the value is determined manually for each node, the value is generated simultaneously for each node in Python using a for loop.

Similarly, to determine the option pay-off, calculated terminal option values. The payoff is determined by taking the maximum of zero and the difference between the stock price and the strike price (price - strike) for each terminal stock price (price) in the terminal stock prices list. Where the max expression always ensures that the option payoff is always nonnegative, if the stock price is greater than strike price, the payoff is the difference, otherwise payoff will be zero. where in the line of code calculates option payoff at each node of binomial tree.

5.REFERENCES

C. J. Foot, Physics Dept, University of Oxford. (2022). Financial Physics. *The binomial tree model: a simple example of pricing financial derivatives*, 8.

department of financial economics-university at albany. (2023). financial economics. *black scholes option pricing* , 21.

Hull ,J.C. (2022). Options, Futures, and Other Derivatives. . In J. Hull, *Options, Futures, and Other DerivativeS* (p. 179). N.Y.: 11th ed. Harlow .

Libraries in python. (2023, 12 28). *Libraries in python.* Retrieved from Greeks for geeks: https://www.geeksforgeeks.org/libraries-in-python/

Steffen Lauritzen, University of Oxford. (2022). Newton–Raphson Iteration. *Newton–Raphson Iteration*, 19.

Yahoo fiannce. (2023, dec 28). *Amazon option chain data* . Retrieved from Yahoo fiannce: https://uk.finance.yahoo.com/quote/AMZN/options?p=AMZN

Yahoo fiannce. (2023, dec 28). *AMZN Stocck historical data Yahoo fiannce*. Retrieved from Yahoo fiannce: https://uk.finance.yahoo.com/quote/AMZN?p=AMZN&.tsrc=fin-srch