University of Westminster

Westminster Business School

Individual Authentic Assessment

Module Title: Computational Methods for Finance
Module Code: 7FNCE041W

Course: MSc Fintech and Business Analytics (Core), Semester 1, 2022/2023

Word count: 2100 (excluding Table of Content, Formula, Referencing and Appendix)

Student Name: [redacted]
STUDENT ID [redacted]

Table of Contents

Python Libraries

a) **NumPy:**

Definition: It is an open-source library available in the Python programming language that is used for the storage of homogeneous as well as heterogeneous data in large multi-dimensional array object. It is used for computing mathematical/numerical and scientific functions, operation, or methods on these arrays. These operations can be basic math, or high-level functions. It supports vectorization of codes.(Yves Hilpisch, 2018)

How to call it:

To install NumPy the following codes can be used:

conda numpy

or

pip numpy

To call NumPy the following code can be used:

import numpy **as** np

After defining it as **np**, the numpy library can be called by using “np” every time an action is required like a computation is to be done during a program. The array object in NumPy is called ndarray.

Screenshot 1: Importing numpy as np and calling it to carry out an arithmetic function

```
In [1]: import numpy as np

In [2]: a = np.array([1,2,3,4,5])

In [3]: a
Out[3]: array([1, 2, 3, 4, 5])

In [4]: a1 = np.array([[1,2,3,4,5],[2,3,4,5,6],[3,4,5,6,7]])
        print(a1[2])
        [3 4 5 6 7]
```

Example:

Here, in the below code block:

Step 1 (line1): NumPy is imported as “np”.

Step 2 (line2): Then an array “a” is defined using ‘np’.

Step 3 (line3): Then ‘a’ is printed.

Step 4 (line4): Another array is defined as “a1”. Then a particular element in the position “3” is accessed and printed.

Function:

- Trigonometric – sin, cos, tan
- Round - fix, floor, ceil
- Sum, product, difference – sum, prod, diff
- Exponents and logarithms – exp, log
- Arithmetic operations – add, positive, negative, divide, multiply, power, subtract, mod
- These are the few of many functions of numpy.

(Numpy.org, n.d.)

Importance or uses:

- It is used for working with numerical data.
- It gives a quick and effective way for creating array, manipulating it, and running numerical (easy or complex) functions in it.
- It **can have both** homogeneous as well as heterogenous elements.
- It is like Python list[1]. But, in a Python list, only homogenous elements can be used for operating any mathematical function.
- It uses **less memory** to store data and is **much faster** than Python list.

(Numpy User Guide, n.d.)

Application: Linear Regression

The equation of a line is **y = mx + c**

x – independent variable

y – dependent variable

c – constant

m – slope of the line

Formula for $m = \frac{\sum(x-\bar{x})(y-\bar{y})}{\sum(x-\bar{x})^2}$

r^2 – r square is the coefficient of determination that suggests how close the data is to the fitted regression line.

Formula for $R^2 = \frac{\sum(y_{pred}-\bar{y})^2}{\sum(y-\bar{y})^2}$

Python list[1]: it is an object that is used to store data, multiple items in a single object.

In python

<u>Code Line 1</u> – The <u>libraries</u> to be imported are **numpy** for numerical computation; **yfinance** for downloading data from yahoo finance API; **matplotlib.pyplot** for plotting the graph; **LinearRgression** from **sklearn.linear_model** for computing the linear regression using least square method.

<u>Code Line 2</u> – A linear regression model is computed to predict the stock returns of **Meta Platforms Inc.** based on the return value of **NASDAQ 100 Index**. The **data for one year** is pulled out from Yahoo finance through an API.

<u>Code Line 3 and 4</u> – Then **adjusted close** is extracted and put in separate data frame.

Screenshot 2: Computation of Linear Regression using sklearn,linear_model library

In [1]:
```
# To import the required libraries

import numpy as np
import yfinance as yf
import matplotlib.pyplot as plt

# To use sklearn for linear regression
from sklearn.linear_model import LinearRegression
```

In [2]:
```
# Create a dataframe with X (Nasdaq 100) and Y (Meta)

initial_data = yf.download("NDX META", start="2022-01-02", end="2023-01-02")
```
```
[*********************100%***********************]  2 of 2 completed
```

In [3]:
```
#Extract Adjusted close data from the dataframe

Adj_close_data = initial_data['Adj Close']
Adj_close_data.head()
```

Out[3]:

	META	NDX
Date		
2022-01-03	338.540009	16501.769531
2022-01-04	336.529999	16279.730469
2022-01-05	324.170013	15771.780273
2022-01-06	332.459991	15765.360352
2022-01-07	331.790009	15592.190430

In [4]:
```
#Putting each adjusted close data in a separate table
Met_df = Adj_close_data['META']
Ndx_df = Adj_close_data['NDX']
```

Code Line 5 and 6 – A log return is computed, and null values are removed.

Code Line 7, 8 and 9 – Linear regression function is defined as **model** and is computed using these **log returns** for Meta stock and Nasdaq 100 Index. The final data is reshaped into an **array**.

Code Line 10 and 11 – Using the linear regression library, r^2, the slope and intercept of the line are computed.

r^2 value computed as 0.4834; the **slope** of the line is 1.4481 and the **intercept** is -0.0018.

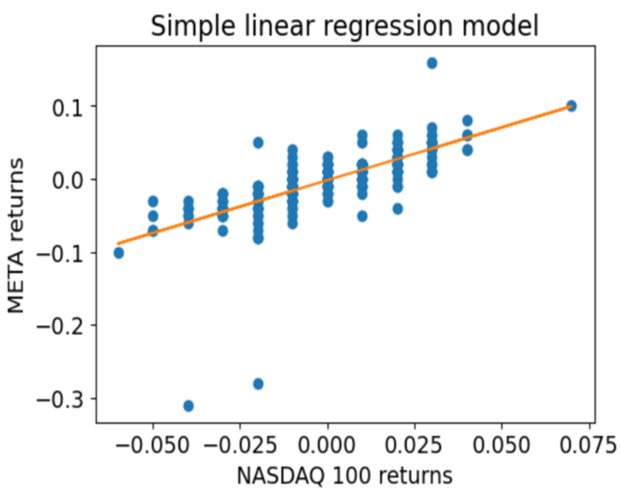
Screenshot 3: Computation of Linear Regression

```
In [5]: # Computation of log retuns for each scrip
        Met_rets = np.log(Met_df/ Met_df.shift(1)).round(2)
        Ndx_rets = np.log(Ndx_df/ Ndx_df.shift(1)).round(2)

In [6]: # Extracting valid data after removing NaN
        Met1 = Met_rets[1:-1]
        Ndx1 = Ndx_rets[1:-1]

In [7]: model = LinearRegression()

In [8]: X = Ndx1.values.reshape(-1,1)
        Y = Met1.values.reshape(-1,1)
        model.fit(X, Y)

Out[8]: LinearRegression()

In [9]: model = LinearRegression().fit(X, Y)

In [10]: r_sq = model.score(X, Y)
         print(f"coefficient of determination: {r_sq.round(4)}")
         print(f"intercept: {model.intercept_.round(4)}")
         print(f"slope: {model.coef_.round(4)}")

         coefficient of determination: 0.4834
         intercept: [-0.0018]
         slope: [[1.4481]]

In [11]: c = model.intercept_.round(4)
         m = model.coef_.round(4)
```

Code Line 12 – A plot of simple linear regression model. It has a scatter plot of the returns and the linear regression line.

Screenshot 4: Plot for Linear Regression

```
In [12]: # Linear regression plot of X (Nasdaq) and Y (Meta)

         plt.figure(figsize = (6, 4))
         plt.rcParams.update({'font.size': 14})
         plt.xlabel("NASDAQ 100 returns")
         plt.ylabel("META returns")
         plt.title("Simple linear regression model")
         plt.plot(X, Y,'o')
         plt.plot(X, m * X + c)
         plt.show()
```

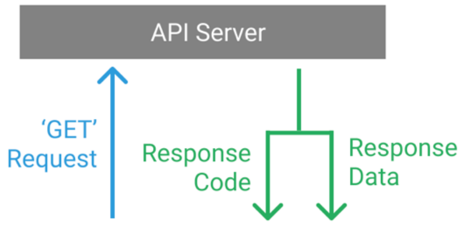


b) <u>yfinance:</u>

Definition: It is an open-source library used to access the financial data available on Yahoo Finance website through an API. (Bland, 2020). It is used to obtain historical as well as real-time data for various financial market products such as stock price, index value, option price, etc.

Application programming interface or API is an interface that allows different systems to communicate with each other. So, by using Yfinance, one can access data available on the Yahoo Finance website through python codes. To get a data set from an API a request is made, the API server receives it and responds to the request. (Devlin, 2020)

Figure 1: API Request-Response Function



It can be used to pull out time-series data for stocks.

Dataset that can be pulled out using yfinance – action, analysis, balance sheet, calendar, cashflow, historical data, information like profile of the company, news about the company, etc.

How to call it:

To call yfinance the following code can be used:

Import yfinance **as** yf

After defining it as **yf**, the yfinance library can be called by using "yf" every time a data set is to be extracted from Yahoo Finance website during a program.

Example:

Here, in the below code block:

Screenshot 5: yfinance calling code

```
In [48]: import yfinance as yf

In [49]: # Import stock data from Yahoo Finance for Google
         Google_df = yf.download("GOOG", start="2022-01-02", end="2023-01-02")
         [*********************100%***********************]  1 of 1 completed
```

Step 1: yfinance is imported as "yf". (as mentioned in line 48).

Step 2: Google or Alphabet Inc.'s financial data is extracted using the codes mentioned in the Line 49.

As mentioned in the start and end, the use can fix the time series between which the data will be extracted.

Function:

- It helps in extracting stock market data.
- It helps in prototyping which is useful for back testing using historical data.
- It helps in analysing data available on Yahoo Finance's website
- It can be used to create trading strategies and models based on live data.
- It is free of charge, actively maintained and lots of data is available at real-time.

Application:

<u>Screenshot 1 (line 57):</u> Here using the yfinance library, action data related to Google ticker on Yahoo Finance website can be extracted. The action data includes divided details and stock split details.

Screenshot 6: Calling yfinance Library

In [57]:
```
Goog = yf.Ticker('goog')
Goog.actions.head()
```

Out[57]:

Date	Dividends	Stock Splits
2014-03-27 00:00:00-04:00	0.0	2.002000
2015-04-27 00:00:00-04:00	0.0	1.002746
2022-07-18 00:00:00-04:00	0.0	20.000000

<u>Screenshot 2 (line 59):</u> Here historical data for five different stocks is extracted using yfinance library. Details like closing stock price, dividend, stock split, volume traded, etc for all the stocks can be extracted simultaneously.

Screenshot 7: Data downloaded from yfinance

In [59]:
```
companies = ['AMZN','GOOG','WMT','TSLA','META']
tickers = yf.Tickers(companies)
tickers_hist = tickers.history(start="2022-01-02", end="2023-01-02")
tickers_hist
```

```
[*********************100%***********************]  5 of 5 completed
```

Out[59]:

	Close					Dividends					...	Stock Splits					Volume	
	AMZN	GOOG	META	TSLA	WMT	AMZN	GOOG	META	TSLA	WMT	...	AMZN	GOOG	META	TSLA	WMT	AMZN	G
Date																		
2022-01-03	170.404495	145.074493	338.540009	399.926666	142.411377	0	0	0	0	0.0	...	0.0	0.0	0	0.0	0	63520000	25
2022-01-04	167.522003	144.416504	336.529999	383.196655	139.802414	0	0	0	0	0.0	...	0.0	0.0	0	0.0	0	70726000	22
2022-01-05	164.356995	137.653503	324.170013	362.706665	141.692673	0	0	0	0	0.0	...	0.0	0.0	0	0.0	0	64302000	49
2022-01-06	163.253998	137.550995	332.459991	354.899994	141.298889	0	0	0	0	0.0	...	0.0	0.0	0	0.0	0	51958000	29
2022-01-07	162.554001	137.004501	331.790009	342.320007	142.647675	0	0	0	0	0.0	...	0.0	0.0	0	0.0	0	46606000	19
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2022-12-23	85.250000	89.809998	118.040001	123.150002	143.770004	0	0	0	0	0.0	...	0.0	0.0	0	0.0	0	57433700	17
2022-12-27	83.040001	87.930000	116.879997	109.099998	143.809998	0	0	0	0	0.0	...	0.0	0.0	0	0.0	0	57284000	15
2022-12-28	81.820000	86.459999	115.620003	112.709999	141.289993	0	0	0	0	0.0	...	0.0	0.0	0	0.0	0	58228600	17
2022-12-29	84.180000	88.949997	120.260002	121.820000	142.149994	0	0	0	0	0.0	...	0.0	0.0	0	0.0	0	54995900	18
2022-12-30	84.000000	88.730003	120.339996	123.180000	141.789993	0	0	0	0	0.0	...	0.0	0.0	0	0.0	0	62330000	19

Screenshot 3 (line 60): The DataFrame can be manipulated, and the desired structure can be used for interpretation.

Screenshot 8: Data frame manipulation

```
In [60]: # TRANSFORM MULTI-LEVEL INDEX INTO A SINGLE-INDEX SET OF COLUMNS.
         tickers_hist.stack(level=1).rename_axis(['Date', 'Ticker']).reset_index(level=1)
```

Out[60]:

Date	Ticker	Close	Dividends	High	Low	Open	Stock Splits	Volume
2022-01-03	AMZN	170.404495	0.0	170.703506	166.160507	167.550003	0.0	63520000
2022-01-03	GOOG	145.074493	0.0	145.550003	143.502502	144.475494	0.0	25214000
2022-01-03	META	338.540009	0.0	341.079987	337.190002	338.299988	0.0	14537900
2022-01-03	TSLA	399.926666	0.0	400.356659	378.679993	382.583344	0.0	103931400
2022-01-03	WMT	142.411377	0.0	142.549210	140.796758	141.771442	0.0	6902200
...	...	...	...	...	...	...	...	...
2022-12-30	AMZN	84.000000	0.0	84.050003	82.470001	83.120003	0.0	62330000
2022-12-30	GOOG	88.730003	0.0	88.830002	87.029999	87.364998	0.0	19179300
2022-12-30	META	120.339996	0.0	120.419998	117.739998	118.160004	0.0	19492100
2022-12-30	TSLA	123.180000	0.0	124.480003	119.750000	119.949997	0.0	157304500
2022-12-30	WMT	141.789993	0.0	141.990005	140.809998	141.559998	0.0	3834800

1255 rows × 8 columns

Screenshot 4 (line 63, line 66): yfinance can be used to extract option data. Like strike price of call or put, at various spot prices and implied volatility.

Screenshot 9: Data extraction for Tesla's Option chain

```
In [63]: # IMPORT REQUIRED LIBRARY
         import yfinance as yf

         # CREATE A TICKER INSTANCE FOR TESLA
         tsla = yf.Ticker('TSLA')

         # FETCH OPTIONS CHAIN DATA FOR THE COMPANY
         tsla_options = tsla.option_chain()

         # ACCESS BOTH THE CALLS AND PUTS AND STORE THEM IN THEIR RESPECTIVE VARIABLES
         tsla_puts = tsla_options.puts
         tsla_calls = tsla_options.calls
```

```
In [66]: tsla_puts
```

Out[66]:

	contractSymbol	lastTradeDate	strike	lastPrice	bid	ask	change	percentChange	volume	openInterest	impliedVolatility	inTheMoney	c
0	TSLA230106P00050000	2022-12-30 20:38:55+00:00	50.0	0.01	0.00	0.01	0.000000	0.000000	243.0	5792	2.437504	False	
1	TSLA230106P00055000	2022-12-30 20:56:02+00:00	55.0	0.01	0.00	0.01	0.000000	0.000000	996.0	4578	2.187505	False	
2	TSLA230106P00060000	2022-12-30 20:59:59+00:00	60.0	0.01	0.00	0.01	-0.010000	-50.000000	2069.0	7196	1.968750	False	
3	TSLA230106P00065000	2022-12-30 20:55:47+00:00	65.0	0.01	0.01	0.02	-0.010000	-50.000000	1518.0	4396	1.937500	False	
4	TSLA230106P00070000	2022-12-30 20:59:28+00:00	70.0	0.02	0.01	0.03	-0.010000	-33.333336	1023.0	8701	1.781251	False	
...	...	...	...	...	...	...	...	...	...	...	...	...	
115	TSLA230106P00300000	2022-12-27 20:21:21+00:00	300.0	188.65	176.40	177.20	0.000000	0.000000	2.0	0	3.464845	True	
116	TSLA230106P00320000	2022-12-30 14:34:30+00:00	320.0	197.45	196.50	197.20	-2.529999	-1.265126	1.0	0	2.781253	True	
117	TSLA230106P00330000	2022-12-15 16:56:13+00:00	330.0	173.14	206.40	207.20	0.000000	0.000000	NaN	0	3.761719	True	
118	TSLA230106P00340000	2022-12-15 16:55:10+00:00	340.0	183.30	216.45	217.20	0.000000	0.000000	NaN	0	2.562504	True	

Question 2

Implied Volatility

a) Volatility Surface

Volatility is the amount of risk associated with the changing market price of an asset like stocks, bonds, derivatives, etc., over a given time-period. This volatility can be estimated using various methods, one such method is by inputting the option price back in the Black-Scholes model to get a theoretical volatility. This volatility is called **Implied Volatility**. (Malik, 2019)

A plot of implied volatility of an option with a certain time to maturity as a function of its strike price is known as **Volatility Smile**. A 3-D plot of this implied volatility as a function of **Strike Price** as well as **Time to Maturity** is called a **Volatility Surface**. (Hull, 2022)

Implied Volatility can be shown is a three-dimensional plot. This **implied volatility surface** signifies a constant value of volatility by giving all the traded options a theoretical value equal to all market value. (Wilmot, 1998)

b) Implied Volatility (σ_{IM})

Implied Volatility is the expected volatility of a stock over the life of an option. It is this volatility when fed into the option pricing model will give a theoretical value of the option which will be equal to the market price of the option. It is used to monitor the market's perception of the volatility of a stock. The implied volatility can be computed from the market price of an option by using **Newton-Raphson**'s method. It uses the derivative of the option price with respect to the volatility or **Vega**.

Newton-Raphson is a method used for swift estimation for real-valued functions $f(x)=0$ as the initial value. $f'(x)$ is the first derivative of $f(x)$.

x_1 = value at 1; x_{i+1} = final value.

$$x_1 = x_0 - \frac{f\left(x_0\right)}{f'\left(x_0\right)} \qquad\qquad x_{i+1} = x_i - \frac{f\left(x_i\right)}{f'\left(x_i\right)}$$

Formula:

Newton-Raphson Iteration

$$\sigma_{n+1} = \sigma_n - \frac{V_{mkt} - V_{BS}(\sigma_n)}{\frac{\partial V_{BS}(\sigma_n)}{\partial \sigma}}$$

Where,

Initial guess for the implied Volatility at n = 0	-	σ_n
Market Price of the Option	-	V_{mkt}
Option Price derived at initial guess	-	V_{BS}
Black–Scholes formula: $V_{BS}(S_0,\ t_0;\ \sigma\ ,\ r; K,\ T)$	=	known value
Asset price	-	S_0
Initial Time	-	t_0
Strike Price	-	K
Risk-free rate	-	r
Time to Maturity	-	T

everything is known in this equation except for σ

Vega at initial guess	-	$\frac{\partial V_{BS}}{\partial \sigma}$
Updated Implied Volatility	-	σ_{n+1}

Formula (Non-dividend European Call Option)

$$\text{Call option value } = SN(d_1) - Ke^{-r(T-t)}N(d_2)$$

Where,

$$d_1 = \frac{\log(S/K) + \left(r + \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}}$$

and

$$d_2 = \frac{\log(S/K) + \left(r - \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}} = d_1 - \sigma\sqrt{T-t}$$

S	:	Current asset price
K	:	Strike price of the option
r	:	Risk free rate
T	:	Time until option expiration (time to maturity)

t	:	Current time
σ	:	Annualized volatility of the asset's returns
N(x)	:	Cumulative distribution function for a standard normal distribution

c) Application:

Implied volatility can be computed using 'Python' by following the below mentioned steps.

Screenshot 10: Codes for downloading Google's Data from Yahoo Finance

In [1]:
```
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 [2]:
```
Google = yf.download("GOOG", start="2022-12-05", end="2023-01-05")
Google.tail()
```

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

Out[2]:

Date	Open	High	Low	Close	Adj Close	Volume
2022-12-28	87.500000	88.519997	86.370003	86.459999	86.459999	17879600
2022-12-29	87.029999	89.364998	86.989998	88.949997	88.949997	18280700
2022-12-30	87.364998	88.830002	87.029999	88.730003	88.730003	19179300
2023-01-03	89.830002	91.550003	89.019997	89.699997	89.699997	20738500
2023-01-04	91.010002	91.239998	87.800003	88.709999	88.709999	26987700

In [3]:
```
S = round(Google['Adj Close'][-1],4)
S
```

Out[3]: 88.71

Step1 (line1)**: Import yfinance** to get data for stock price and option price of the stock. Here Google is chosen as the Stock. **Import** various libraries like **numpy, pandas, scipy.stats,** and **os** for using **Black Scholes** model to compute **Implied Volatility**.

Step2 (line2): Download data for **Google** (Alphabet Inc.) Stock price from Yahoo Finance to find a spot price.

Step3 (line3): Extract the last traded price as the **Spot Price**. It is **$88.71**.

Screenshot 11: Defining the function to compute Implied Volatility using Newton-Raphson Iteration

```
In [4]: def newton_vol_call(S, K, T, C, r):

            #S: spot price
            #K: strike price
            #T: time to maturity
            #C: Call value
            #r: risk free rate
            #sigma: volatility of underlying asset

            MAX_ITERATIONS = 1000
            tolerance = 0.000001

            sigma = 0.25

            for i in range(0, MAX_ITERATIONS):
                d1 = (np.log(S / K) + (r + 0.5 * sigma ** 2) * T) / (sigma * np.sqrt(T))
                d2 = (np.log(S / K) + (r - 0.5 * sigma ** 2) * T) / (sigma * np.sqrt(T))
                price = S * si.norm.cdf(d1, 0.0, 1.0) - K * np.exp(-r * T) * si.norm.cdf(d2, 0.0, 1.0)
                vega = S * np.sqrt(T) * si.norm.pdf(d1, 0.0, 1.0)

                diff = C - price

                if (abs(diff) < tolerance):
                    return sigma
                else:
                    sigma = sigma + diff/vega

                # print(i,sigma,diff)

            return sigma
```

```
In [5]: Google_opt = yf.Ticker("GOOG")
        opt = Google_opt.option_chain('2023-01-27')
        opt.calls
```

Out[5]:

	contractSymbol	lastTradeDate	strike	lastPrice	bid	ask	change	percentChange	volume	openInterest	impliedVolatility	inTheMoney	contractS
0	GOOG230127C00050000	2022-12-23 19:13:09+00:00	50.0	39.90	0.0	0.0	0.0	0.0	37.0	37	0.000010	True	REGUL

Step4 (line4): Define a function called “**newton_vol_call**” to compute the implied volatility using Black Scholes Model for **European Call Option** as explained in the previous section. Use 1000 as max **iteration**; **tolerance** 0.000001 and **initial volatility** as 25% (sigma = 0.25). All the other formula are as per the previous section (2(b)).

Step5 (line5): Download Option Data chain of Google from Yahoo Finance. And extract the call option prices form the chain. The output shows the data present on Yahoo Finance for Option contracts

Screenshot 12: Calling the Newton-Raphson function for computation of Implied volatility

```
In [6]: impvol = newton_vol_call(S, 100, 1/12, float(opt.calls.lastPrice[opt.calls.strike == 100.00]), 0.0353)
        print('The implied volatility is', round(impvol*100,2) , '% for the one-month call with strike $100.00' )

        The implied volatility is 28.97 % for the one-month call with strike $100.00
```

Step6 (line 6): Compute the implied volatility for European call option. Here the implied volatility for one-month at **Strike price $100** is **28.97%** with **risk free rate** as **3.53%** and **time to maturity** as **1 month**.

d) Implied volatility computation of **Tesla's Option**

Strike Price - **$120**

Expire Date – **Jan 20, 2023**

Risk-free rate – **3.5%**

Screenshot 13: Fetching data from Yahoo Finance for Tesla's Stock Price

```
In [54]: 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 [64]: TSLA = yf.download("TSLA", start="2022-01-05", end="2023-01-04")
         TSLA.tail()

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

Out[64]:

Date	Open	High	Low	Close	Adj Close	Volume
2022-12-27	117.500000	119.669998	108.760002	109.099998	109.099998	208643400
2022-12-28	110.349998	116.269997	108.239998	112.709999	112.709999	221070500
2022-12-29	120.389999	123.570000	117.500000	121.820000	121.820000	221923300
2022-12-30	119.949997	124.480003	119.750000	123.180000	123.180000	157304500
2023-01-03	118.470001	118.800003	104.639999	108.099998	108.099998	231402800

```
In [56]: S = TSLA['Adj Close'][-1]
         print('The spot price is $', round(S,2), '.')

         The spot price is $ 108.1 .
```

Step1 (line54): **Import yfinance** to get data for stock price and option price of the stock. Here **Tesla** is chosen as the Stock. **Import** various libraries like **numpy, pandas, scipy.stats,** and **os** for using **Black Scholes** model to compute **Implied Volatility**.

Step2 (line 64): Download data for **Tesla's** Stock price from Yahoo Finance to find a spot price.

Step3 (line56): Extract the last traded price as the **Spot Price**. It is **$108.10**.

```
In [7]: import mibian

In [17]: c = mibian.BS([S, 120.00, 3.5, 21], callPrice = float(opt.calls.lastPrice[opt.calls.strike == 120.00]))
         c.impliedVolatility
         print('The implied volatility is', round(c.impliedVolatility,2) , '% for the three weeks call with strike $120.00' )

         The implied volatility is 86.91 % for the three weeks call with strike $120.00
```

Step4 (line7 and line17): Compute the implied volatility for European call option using Mibian. **'Mibian'** is a library in Python that helps in computing Option Price using various model formula like **Black Scholes and Merton**. Here the implied volatility for three weeks at **Strike price $120** is **86.91%** with **risk free rate** as **3.50%** and **time to maturity** as **three weeks** (contract expire date January 20, 2023).

Screenshot 14:Defining the Newton-Raphson Iteration for computing implied volatility using Black-Scholes Model

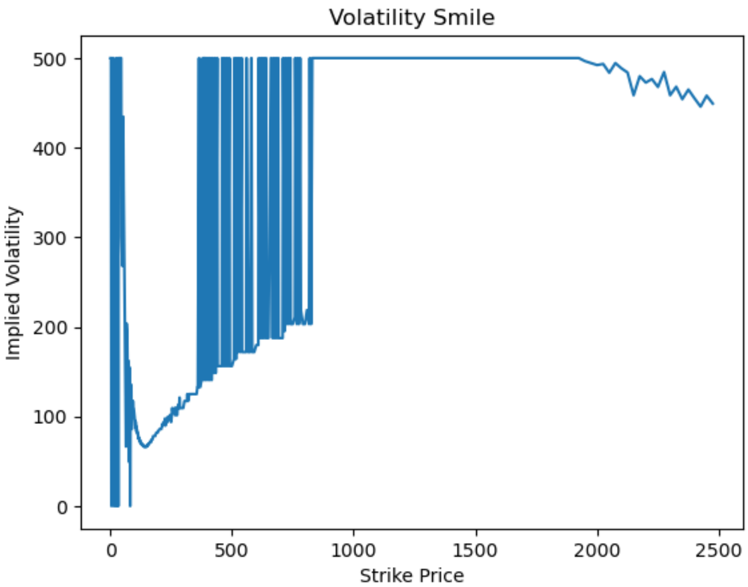
```
In [18]: def newton_vol_call(S, K, T, C, r):

             #S: spot price
             #K: strike price
             #T: time to maturity
             #C: Call value
             #r: risk free rate
             #sigma: volatility of underlying asset

             MAX_ITERATIONS = 1000
             tolerance = 0.000001

             sigma = 0.25

             for i in range(0, MAX_ITERATIONS):
                 d1 = (np.log(S / K) + (r + 0.5 * sigma ** 2) * T) / (sigma * np.sqrt(T))
                 d2 = (np.log(S / K) + (r - 0.5 * sigma ** 2) * T) / (sigma * np.sqrt(T))
                 price = S * si.norm.cdf(d1, 0.0, 1.0) - K * np.exp(-r * T) * si.norm.cdf(d2, 0.0, 1.0)
                 vega = S * np.sqrt(T) * si.norm.pdf(d1, 0.0, 1.0)

                 diff = C - price

                 if (abs(diff) < tolerance):
                     return sigma
                 else:
                     sigma = sigma + diff/vega

                 # print(i,sigma,diff)

             return sigma
```

```
In [19]: impvol = newton_vol_call(S, 120, 3/52, float(opt.calls.lastPrice[opt.calls.strike == 120.00]), 0.035)
         print('The implied volatility is', round(impvol*100,2) , '% for the three weeks call with strike $ 120.00' )

         The implied volatility is 86.35 % for the three weeks call with strike $ 120.00
```

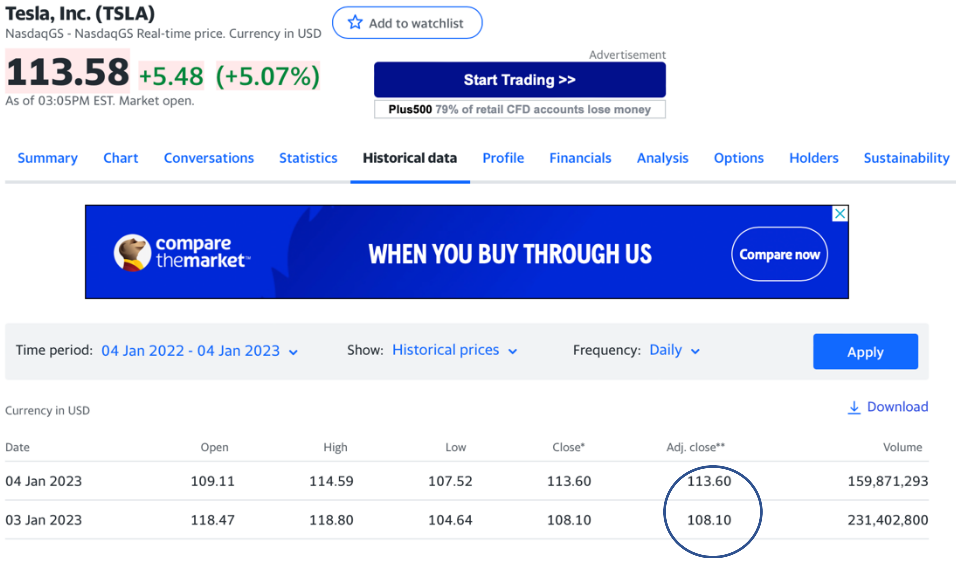
Step5 (line18 and line19): Using Black Scholes formula by applying Newton-Raphson iteration, we get the **implied volatility** as **86.35%** with **risk free rate** as **3.50%** and **time to maturity** as **three weeks** (contract expire date January 20, 2023).

Plot of Implied Volatility against the Strike Price also known as the **Volatility Smile**.

Figure 2: Volatility Smile



Spot Price from Yahoo Finance – as on January 3, 2023, is **$108.10.**



Tesla, Inc. (TSLA)
NasdaqGS - NasdaqGS Real-time price. Currency in USD
Add to watchlist

113.58 +5.48 (+5.07%)
As of 03:05PM EST. Market open.

Advertisement
Start Trading >>
Plus500 79% of retail CFD accounts lose money

Summary | Chart | Conversations | Statistics | Historical data | Profile | Financials | Analysis | Options | Holders | Sustainability

compare themarket
WHEN YOU BUY THROUGH US
Compare now

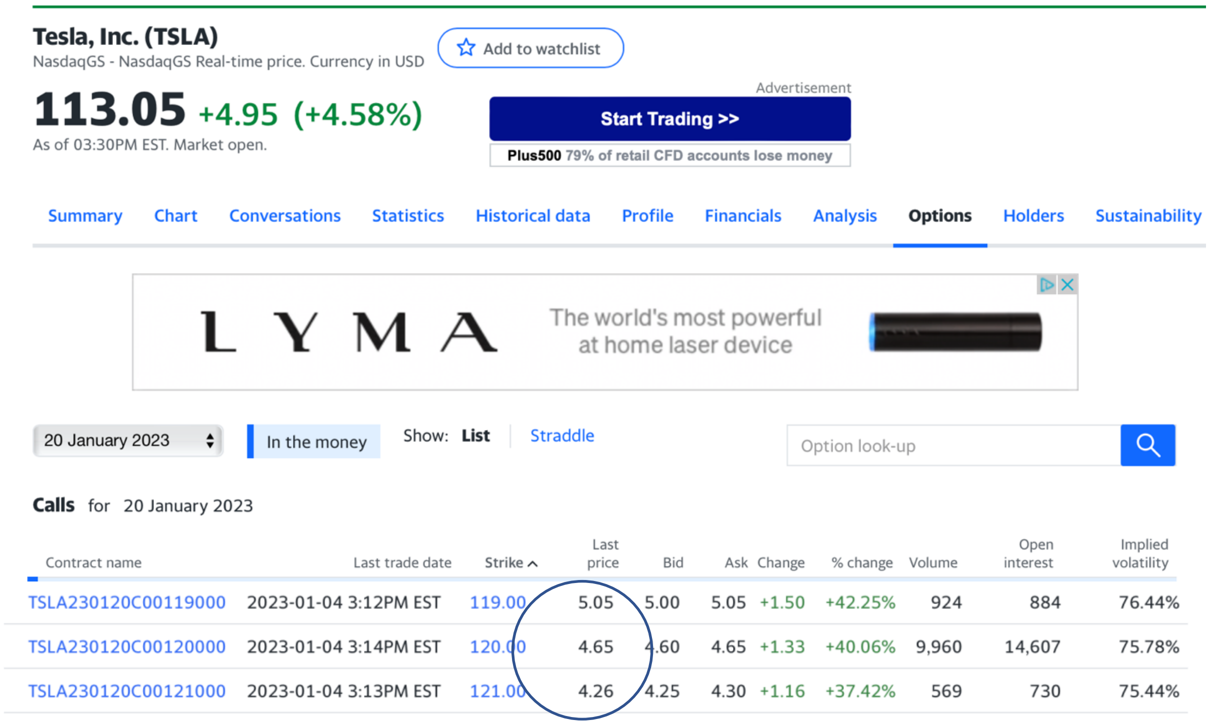
Time period: 04 Jan 2022 - 04 Jan 2023 Show: Historical prices Frequency: Daily Apply

Currency in USD Download

Date	Open	High	Low	Close*	Adj. close**	Volume
04 Jan 2023	109.11	114.59	107.52	113.60	113.60	159,871,293
03 Jan 2023	118.47	118.80	104.64	108.10	108.10	231,402,800

Source: Yahoo Finance

Call Price at Strike Price $120 from Yahoo Finance is $4.65



Tesla, Inc. (TSLA)
NasdaqGS - NasdaqGS Real-time price. Currency in USD
Add to watchlist

113.05 +4.95 (+4.58%)
As of 03:30PM EST. Market open.

Advertisement
Start Trading >>
Plus500 79% of retail CFD accounts lose money

Summary | Chart | Conversations | Statistics | Historical data | Profile | Financials | Analysis | Options | Holders | Sustainability

L Y M A The world's most powerful at home laser device

20 January 2023 | In the money | Show: List | Straddle | Option look-up

Calls for 20 January 2023

Contract name	Last trade date	Strike	Last price	Bid	Ask	Change	% change	Volume	Open interest	Implied volatility
TSLA230120C00119000	2023-01-04 3:12PM EST	119.00	5.05	5.00	5.05	+1.50	+42.25%	924	884	76.44%
TSLA230120C00120000	2023-01-04 3:14PM EST	120.00	4.65	4.60	4.65	+1.33	+40.06%	9,960	14,607	75.78%
TSLA230120C00121000	2023-01-04 3:13PM EST	121.00	4.26	4.25	4.30	+1.16	+37.42%	569	730	75.44%

Source: Yahoo Finance

e) 1-year Tesla's annualised volatility

The annualised volatility can be computed using the following formula:

$$= \sigma\sqrt{t}$$

Standard deviation of log returns multiplied by square root of time.

This computation can be carried out in the Python using the following codes.

Step1 (line57): Compute log returns by extracting the adjusted close date for Tesla stock. The formula used is log (today's return/yesterday's return)

Step2: Compute volatility using the above formula.

Square root of 252 days * standard deviation of log returns

Step3: Print the final value of annualised volatility. It is **66.96%**.

```
In [57]: log_return = np.log(TSLA['Adj Close'] / TSLA['Adj Close'].shift(1))
         vol_h = np.sqrt(252) * log_return.std()
         print('The annualised volatility is', round(vol_h*100,2), '%')

The annualised volatility is 66.96 %
```

Annualised Volatility is also called the **Historical Volatility**. It is backward looking. Whereas **Implied volatility** is forward looking. If the two volatilities are similar, it means that the option has a fair price. (HAYES, A., 2022)

Here, the annualised volatility is **66.96%** and the implied volatility is **86.35%** (using Newton-Raphson). This means that the Option Premiums are **over estimated**. High **Implied volatility** indicates higher premium. At this level Selling of the Option is a better choice as the writer can make profit from the inflated premium till the levels go back to the average range bringing the option premium back to fair price levels. Here the strategy is to **"sell high and buy low"**.

Question 3

Binomial Tree Option Pricing

Stock	–	Non-dividend paying
Spot price 'S'	-	\$100
Volatility ' σ '	–	20%
Risk free rate 'r'	–	5% p.a.
Time to Maturity 'T'	–	6 months or 0.5 years
Time Step 'n'	–	2 (3-month periods or 0.25 years each)

a) Formula for u

u = e^(σ√dT)

Computation:

Using the excel function – exp for exponential and sqrt for square root, we get the following result:

dT = T/n = 0.5/2 = 0.25

u = exp^((0.20)*(0.25))

= 1.11

Formula for d

d = 1/u

Computation:

d = 1/1.11

= 0.90

Formula for p

Risk-neutral probability for 'up'

p = (e^rdT - d)/(u - d)

Risk-neutral probability for 'up'

q = 1-p

Computation:

p = (exp^(5%*0.25) – 0.90) / (1.11-0.90)

p = 0.54

q = 1-0.54

q = 0.46

Therefore, the values of ‘u’, ‘d’, and ‘p’ are **1.11**, **0.90** and **0.54**, respectively.

b) Value of European Call Option

Strike Price - $95

p – 0.54

u – 1.11

d – 0.90

Stock Price as positions:

A = S_0 = $100

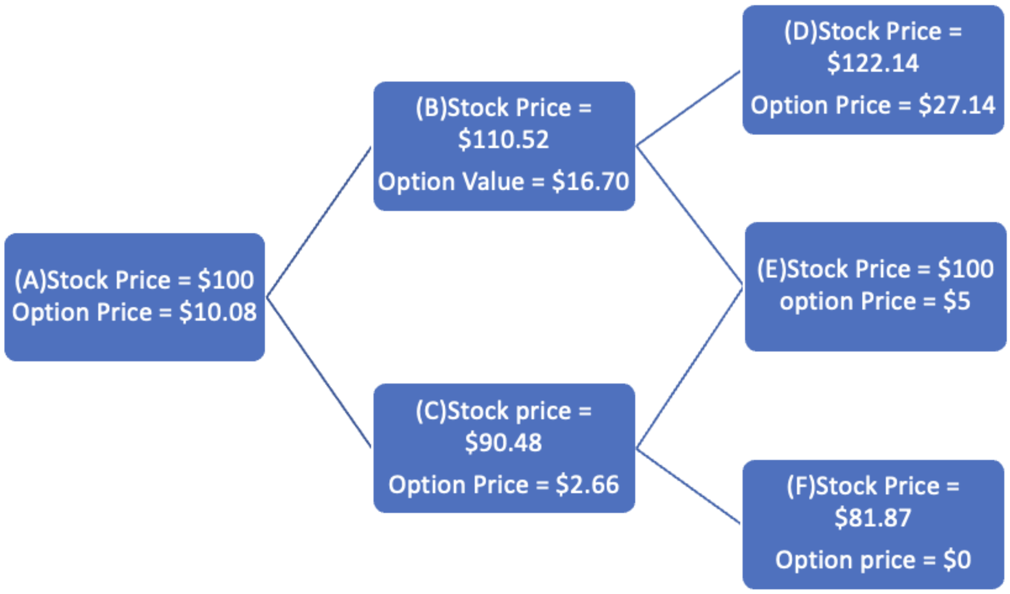
B = S_0*u = $110.52

C = S_0*d = $90.48

D = S_0*u*u = $122.14

E = S_0*u*d = $100

F = S_0*d*d = $81.87



Formula for European Call Option

$$c = e^{(-rT)} [p\, f_u + q\, f_d]$$

Here,

r is the risk-free rate

T is the time to maturity

p is the risk neutral probability

payoff is [Max (S_i - K, 0)]

Therefore, the value of the call option at node B

f_u is payoff if the stock moves up, which is (S*u*u)-95 = $27.14 in this case.

f_d is payoff if the stock moves down, which is (S_0*u*d) -95 = $5 in this case.

c = e^(-5%*3/12) [0.54*27.14 + 0.46*5]

= $16.70

The value of the call option at node C

f_u is payoff if the stock moves up, which is (S_0)-95 = $5 in this case.

f_d is payoff if the stock moves down, which is (S_0*d)-95 = $0 in this case.

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

= $2.66

The value of the call option at node A

Value of Call Option for up - $16.70

Value of Put Option for down - $2.66

c = e^(-5%*3/12) [0.54*16.70+ 0.46*2.66]

= $10.08

The value of the European Call Option is $10.08

c) Python code for Option Valuation for European Call Option

Screenshot 15:Computation of u d and p

```
In [37]: import numpy as np
         import os
```

```
In [38]: S0 = 100                    # spot stock price
         K = 95                     # strike
         T = 0.5                   # maturity
         r = 0.05                   # risk free rate
         sigma = 0.20               # diffusion coefficient or volatility
         N = 2                      # number of periods or number of time steps
         payoff = "call"            # payoff
```

```
In [39]: dT = float(T) / N                                    # Delta t
         u = np.exp(sigma * np.sqrt(dT))                      # up factor
         d = 1.0 / u                                          # down factor
```

```
In [44]: round(u,2),round(d,2)

Out[44]: (1.11, 0.9)
```

Step1 (line37): Import libraries numpy and os for conducting mathematical like addition, multiplication, square root operations and fetching data, respectively.

Step2 (line38): Define variables like Spot price, Strike Price, Time to maturity, time step, risk free rate, volatility/sigma and the payoff (call in this case).

Step3 (line39): Compute dT, u and d. dT is delta, u is the factor by which the stock price will go up and d is the factor by which the stock price will come down.

Step4 (line44): Rounding-off to two decimal places and printing the values of u and d.

Step5 (line40 line 41): Run a 'for' loop to create the binomial tree using the values of u and d for up and down movement of stock price, starting at $100. Print the array with up and down price of the stock.

Step6 (line 45, line 46): Compute the risk neutral probability for the up and down movement of the stock price.

Screenshot 16: Computation of up and down factors for stock price and drawing the binomial tree

```
In [40]: S = np.zeros((N + 1, N + 1))
         S[0, 0] = S0
         z = 1
         for t in range(1, N + 1):
             for i in range(z):
                 S[i, t] = S[i, t-1] * u
                 S[i+1, t] = S[i, t-1] * d
             z += 1
```

```
In [41]: S

Out[41]: array([[100.        , 110.51709181, 122.14027582],
                [  0.        ,  90.4837418 , 100.        ],
                [  0.        ,   0.        ,  81.87307531]])
```

```
In [45]: a = np.exp(r * dT)    # risk free compound return
         p = (a - d)/ (u - d)  # risk neutral up probability
         q = 1.0 - p           # risk neutral down probability
         round(p,2)

Out[45]: 0.54
```

```
In [46]: round(1-p,2)

Out[46]: 0.46
```

```
In [47]: S_T = S[:,-1]
         V = np.zeros((N + 1, N + 1))
         if payoff =="call":
             V[:,-1] = np.maximum(S_T-K, 0.0)
         elif payoff =="put":
             V[:,-1] = np.maximum(K-S_T, 0.0)
         V

Out[47]: array([[ 0.        ,  0.        , 27.14027582],
                [ 0.        ,  0.        ,  5.        ],
                [ 0.        ,  0.        ,  0.        ]])
```

The value of up is p – 0.54

The value of down is q – 0.46

Step7 (line47): Run an if-else if condition statement to compute the price of the Option at maturity.

Screenshot 17: Computation of Option price using binomial tree

```
In [48]: # for European Option
         for j in range(N-1, -1, -1):
             for i in range(j+1):
                 V[i,j] = np.exp(-r*dT) * (p * V[i,j + 1] + q * V[i + 1,j + 1])
         V

Out[48]: array([[10.08051081, 16.69720076, 27.14027582],
                [ 0.        ,  2.65563805,  5.        ],
                [ 0.        ,  0.        ,  0.        ]])

In [49]: print('European ' + payoff, str( V[0,0]))

         European call 10.080510812216861
```

Step8 (line48): Run another for loop to compute the value of the option at all possible nodes. The value at the beginning of the node will be the price of the option. It is **$10.08** for the European Call option with spot price at **$100,** strike price at **$95**, with the present value computed at a risk-free rate of **5%** with a maturity of total 6-months with two periods of 3-months each.

Step9 (line49): Print the final value of the European call option.

d) Compare b) and c)

The value of the European Call option using both the methods is **$10.08**.

In the manual computation the values of u, d, p and q were computed first to find out the probability of up and down stock price movement and its probability. Then the formula for option price is computed at each node.

The main difference between the two methods is in the manual computation the value is computed for all nodes, whereas in python the for loop is used to compute the value for all nodes at once.

Similarly, for finding out the option pay-off if and else if condition is used to run the loop and compute the pay-off at all nodes, whereas the pay-off is manually computed for all nodes one after the other.

References

1. Bland, G. (2020). *yfinance Library - A Complete Guide - AlgoTrading101 Blog*. [online] Quantitative Trading Ideas and Guides - AlgoTrading101 Blog. Available at: https://algotrading101.com/learn/yfinance-guide/.
2. Devlin, J. (2020). *Python API Tutorial: Getting Started with APIs*. [online] Dataquest. Available at: https://www.dataquest.io/blog/python-api-tutorial/.
3. HAYES, A. (2022). *Implied Volatility vs. Historical Volatility: The Main Differences*. [online] Investopedia. Available at: https://www.investopedia.com/articles/investing-strategy/071616/implied-vs-historical-volatility-main-differences.asp.
4. Hilpisch, Y.J. (2015). *Derivatives analytics with Python : data analysis, models, simulation, calibration and hedging*. Hoboken: Wiley.
5. Hull, J.C. (2022). *Options, Futures, and Other Derivatives*. 11th ed. Harlow Etc.: Pearson Educational Limited. Copyright.
6. Malik, F. (2019). *Volatility Surface*. [online] Medium. Available at: https://medium.com/fintechexplained/volatility-surface-499e6b029373.
7. Numpy User Guide (n.d.). *NumPy: the absolute basics for beginners — NumPy v1.20 Manual*. [online] numpy.org. Available at: https://numpy.org/doc/stable/user/absolute_beginners.html.
8. Numpy.org (n.d.). *Mathematical functions — NumPy v1.24 Manual*. [online] numpy.org. Available at: https://numpy.org/doc/stable/reference/routines.math.html [Accessed 5 Jan. 2023].
9. uk.finance.yahoo.com. (n.d.). Tesla, Inc. (TSLA) options chain – Yahoo Finance. [online] Available at: https://uk.finance.yahoo.com/quote/TSLA/options?p=TSLA [Accessed 5 Jan. 2023].
10. Wilmot, P. (1998). *Derivatives : the theory and practice of financial engineering*. Chichester, West Sussex, England ; J. Wiley.
11. Wilmott, P. (2013). *Paul Wilmott Introduces Quantitative Finance*. John Wiley & Sons.
12. Yves Hilpisch (2018). *Python for Finance*. O’Reilly Media.

Appendix

Figure 3: Computation for the Value of Option

If there is a down movement in the stock price, the value becomes

$$S_0 d\Delta - f_d$$

The two are equal when

$$S_0 u\Delta - f_u = S_0 d\Delta - f_d$$

or

$$\Delta = \frac{f_u - f_d}{S_0 u - S_0 d} \qquad \textbf{(13.1)}$$

In this case, the portfolio is riskless and, for there to be no arbitrage opportunities, it must earn the risk-free interest rate. Equation (13.1) shows that Δ is the ratio of the change in the option price to the change in the stock price as we move between the nodes at time T.

If we denote the risk-free interest rate by r, the present value of the portfolio is

$$(S_0 u\Delta - f_u)e^{-rT}$$

The cost of setting up the portfolio is

$$S_0\Delta - f$$

It follows that

$$S_0\Delta - f = (S_0 u\Delta - f_u)e^{-rT}$$

or

$$f = S_0\Delta(1 - ue^{-rT}) + f_u e^{-rT}$$

Substituting from equation (13.1) for Δ, we obtain

$$f = S_0\left(\frac{f_u - f_d}{S_0 u - S_0 d}\right)(1 - ue^{-rT}) + f_u e^{-rT}$$

or

$$f = \frac{f_u(1 - de^{-rT}) + f_d(ue^{-rT} - 1)}{u - d}$$

or

$$f = e^{-rT}[pf_u + (1 - p)f_d] \qquad \textbf{(13.2)}$$

where

$$p = \frac{e^{rT} - d}{u - d} \qquad \textbf{(13.3)}$$

Source: (John C. Hull, 2022)