DECEMBER 4, 2023

UNIVERSITY OF WESTMINSTER

ALIBABA GROUP HOLDING LIMITED - A REPORT ON FINANCIAL DERIVATIVE

Computational Methods for Finance - 7FNCE041W
Fintech With Business Analytics MSc

VERSHA SANDESH
STUDENT ID: 20395827

Word Count: 2147, excluding Cover Page, Abstract, Table of Contents and References.

Table of Contents

Abstract

This report analyses the Alibaba Group Holding Limited stock's performance, followed by designing a non-dividend paying European call and put option through Python application by using parameters taken from the same data.

It is revealed that the equity movement for Alibaba stock has faced downward trend in the timeframe of last two years from 01 November 2021 to 31 October 2023. It is proven with the help of annualized average return and annualized standard deviation that BABA stock has negative returns with high volatility risk over the specified period.

The three well known option pricing models are used to design the option on underlying stock of Alibaba. Binomial Tree, Monte Carlo Simulation and Black-Scholes option pricing models derived similar call and put option pricing results, with Monte Carlo and Black-Scholes giving more equivalent results.

Finally, the Greeks over designed option is analysed to measure option's sensitivity with respect to changes in price of underlying asset (delta), the rate of change of delta (gamma), changes in time (theta), changes in volatility (vega), and the change in interest rate (rho).

All the relevant calculations can be referred on below link:
https://github.com/vershasandesh/CMF/blob/main/Coursework-%20Versha%20Sandesh%20(20395827).ipynb

1. Introduction

This report evaluates the equity performance of Alibaba Group Holding Limited, one of the FTSE100's most valuable corporations.

The data spanning the preceding two years has been sourced from Yahoo Finance, covering the period from November 1, 2021, to October 31, 2023. The analysis includes an examination of Alibaba's stock, the formulation of a financial derivative, and the computation of Greeks for risk assessment and management. The analysis is conducted using Python.

2. Company Overview

Alibaba Group Holding Limited (BABA), founded in 1999 by Jack Ma and 17 co-founders in Hangzhou, China, is a prominent Chinese multinational technology company specializing in e-commerce, retail, internet, and technology. Initially envisioned as an online marketplace connecting Chinese manufacturers with global buyers, Alibaba has evolved into one of the world's largest retailers and e-commerce entities.

Expanding beyond China, Alibaba has established a global presence with offices in China, India, Singapore, Japan, Australia, Korea, Germany, the UK, and the US. The company provides a diverse range of services, including consumer-to-consumer (C2C), business-to-consumer (B2C), and business-to-business (B2B) sales through Chinese and global marketplaces. Additionally, Alibaba offers local consumer services, digital media, entertainment, logistics, and cloud computing services (MarketLine, 2023).

Notably, Alibaba gained significant attention with its historic initial public offering (IPO) on the New York Stock Exchange (NYSE) in September 2014, raising billions of dollars and achieving a substantial valuation.

Alibaba's Mission:

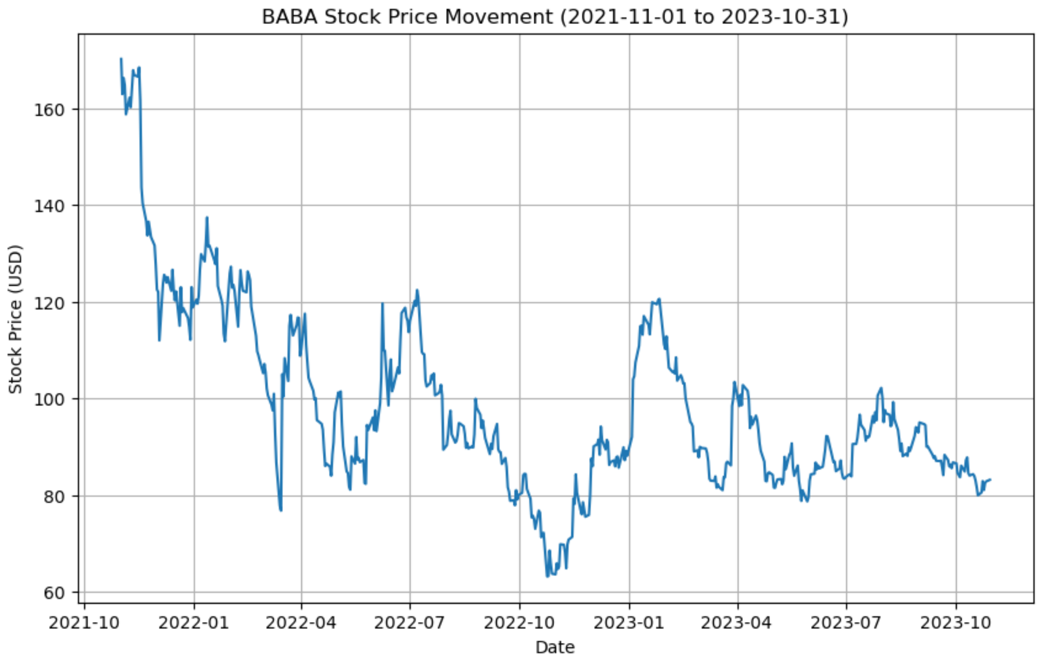
"To make it easy to do business anywhere". (Alibaba, 2023)

Key Facts:

Head Office:	Hangzhou, Zhejiang, China
Web Address:	www.alibabagroup.com
NYSE Ticker:	BABA
Financial Year End:	March

3. Stock Price Movement

The below graph demonstrates the stock price movement of Alibaba Group over the analysis period of two years. It shows that the stock price has continuously fluctuated and decreased over time, however, with frequent upward pushes in price.



4. Annualized Return and Standard Deviation

Annualized average return and annualized standard deviation metrics are used in Finance to describe the performance and risk of an investment. Annualized average return represents the average rate of return per year over a certain period, and annualized standard deviation is a measure of volatility or risk associated with an investment. It quantifies the degree of variation of returns from the average return.

	Formula	**Normal Return**	**Logarithm Return**
Annualized Average Return	$\bar{X} = \bar{x} * T$	-34.7397%	-71.6272%
Annualized Average Standard Deviation	$Vol = \sigma\sqrt{T}$	86.6646%	85.0797%

Logarithmic return and standard deviation are employed for a symmetric interpretation of Alibaba stock results. The -71.6272% logarithmic return indicates a negative average return, implying a loss in Alibaba's stock value over the specified period. Additionally, the 85.0797% annual standard deviation reflects high volatility, indicating a heightened level of risk.

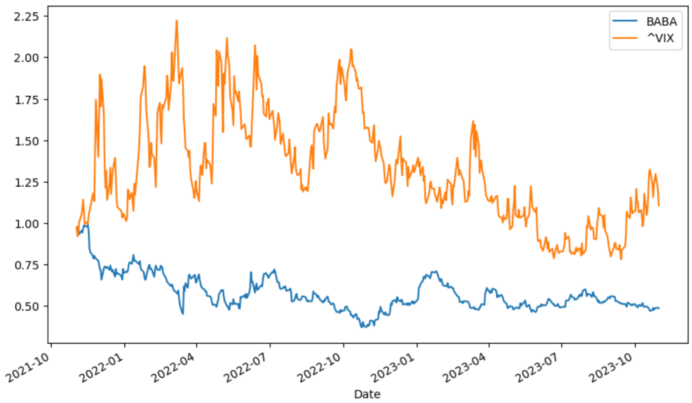
Summary Statistics

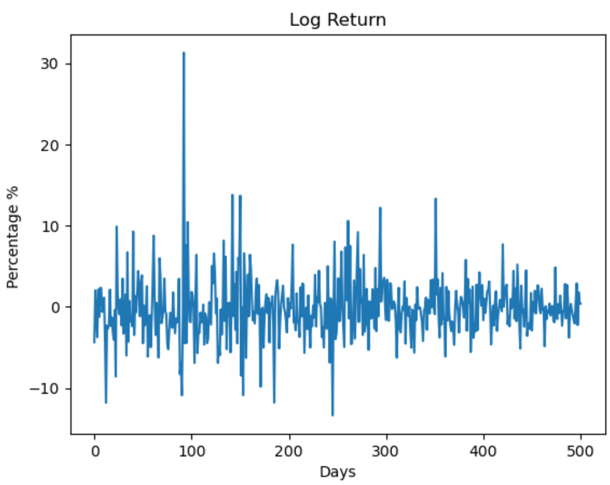
Descriptive statistics for BABA stock reveal a mean price of USD 98.23 and an average volatility (VIX) of USD 21.78. Within the specified timeframe, the stock experienced a minimum price of USD 63.15 and a maximum of USD 170.17.

As of December 1, 2023, the current stock price stands at USD 73.99, placing it in the lower 25th percentile and suggesting potential upward movement (Yahoo Finance, 2023).

	BABA	^VIX
count	503.00	503.00
mean	98.23	21.78
std	18.90	5.39
min	63.15	12.82
25%	85.85	17.64
50%	92.90	21.09
75%	108.63	25.71
max	170.17	36.45

Alibaba's stock price movement with respect to its volatility and the log return are illustrated in below plots.





5. Designing a Non-Dividend Option on Alibaba Stock

A call and put option are designed based on Alibaba's stock performance by using multiple option pricing models like Binomial Trees, Monte Carlo Simulation and Black-Scholes Model. The analysis is performed assuming a non-dividend paying European Option, having the following characteristics:

Details	Symbol	Value
Spot Price (USD) – As of 31/10/2023	S	82.50
Strike Price (USD) – Assumed	K	83.50
Time to Maturity – 3 Months	T	3/12 or 0.25
Risk Free Rate – 13 Weeks or 3 months US Treasury Bill Rate as of 31/10/2023 (U.S. Department of the Treasury, 2023)	r	5.33%
Volatility – Calculated as 3-month rolling standard deviation of Alibaba Stock	sigma	0.02

5.1. Binomial Tree

A binomial tree is like a decision tree that helps in visualizing and computing the possible future values of a financial asset. It considers two outcomes at each stage, where the stock price can either go up or down, creating branching possibilities for evolving stock prices.

Each branching point in the tree is a node, with the initial node representing the current stock price and the final set of nodes indicating possible prices at the option's maturity. Increasing the number of nodes, achieved by adding more steps in the tree, enhances accuracy.

This report uses 3 steps (N=3) for the application of Binomial Tree. The geometric Brownian Motion is approximated to examine mean and variance of the stock price under risk-neutral valuation. To construct a binomial tree, it is required to calculate the risk-neutral probability, up factor, and down factor with help of following formulas:

Risk-neutral probability: $p = \frac{e^{r*\Delta t} - d}{u-d}$ = 0.8840696345329057

Up factor: $u = e^{\sigma\sqrt{\Delta t}}$ = 1.0057902014799276

Down factor: $d = \frac{1}{u}$ = 0.9942431319459984

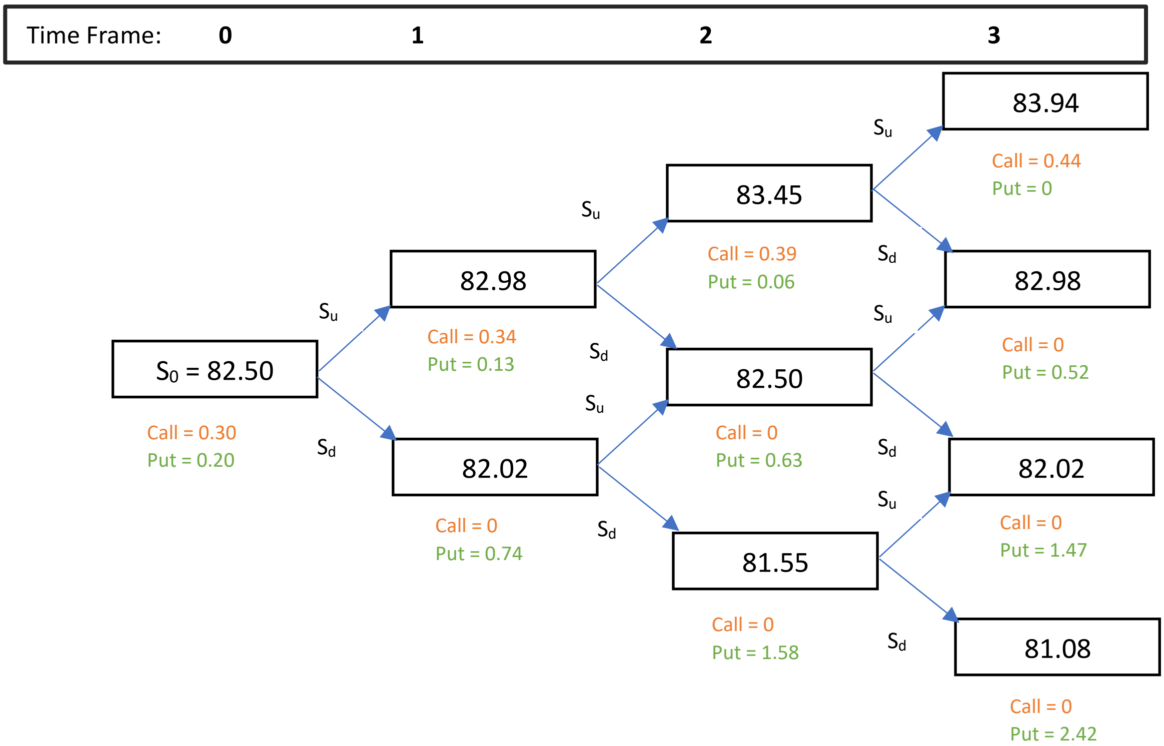
Where:

r = risk-free rate = 5.33% or 0.0533

σ = volatility = 0.02

Δt = T/N = (3/12)/3 = 1/36 for each node

The resulting tree depicts the option price of Alibaba for each month during the 3-month tenure with potential upward or downward values.



According to above results, if the maturity price exceeds the strike price ($S_T > 83.5$), the call option will be exercised while the put option becomes worthless. Conversely, if the stock price falls below the strike price ($S_T < 83.5$), the put option will be exercised, rendering the call option worthless. For instance, at a stock price of 83.94 (node 3,1), the call option has a payoff value of 0.44, whereas the put option has no value. Conversely, at a stock price of 82.94 (node 3,2), the call option has no payoff, but the put option has a value of 0.52.

Subsequently, the expected price for 3-month non-dividend-paying European option from the Binomial Tree calculation is:

Call option price = 0.3009495045480285 ≅ 0.3

Put option price = 0.19569213544319675 ≅ 0.2

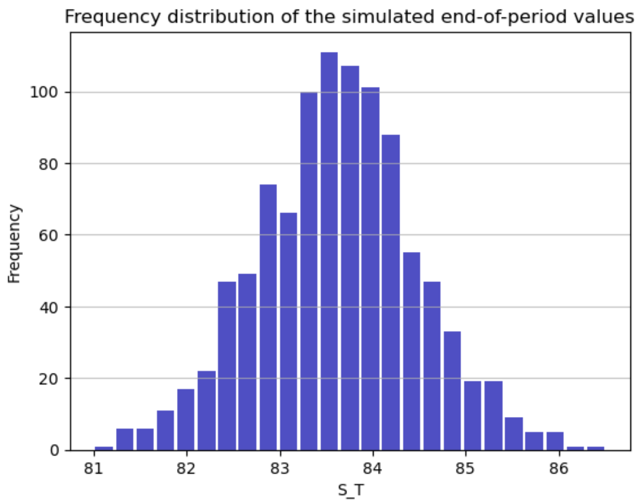
5.2. Monte Carlo Simulation

Monte Carlo Simulation is widely used for predicting stock option prices. It implicitly incorporates the risk-neutral probability distribution of future stock prices. It allows analysts to consider a wide range of potential future scenarios and provides a more nuanced understanding of the option's fair value.

The simulation involves projecting the future prices over discrete time intervals, typically matching the time to expiration of the option. It generates multiple random paths of future stock prices using the Geometric Brownian Motion (GBM) model. Each path represents a potential scenario of how the stock price might evolve over time.

For each simulated path, option payoff at the expiration date is calculated. The payoff depends on the option type (call or put) and the difference between the stock price and the option's strike price. Then the risk-free rate is applied to discount each simulated future payoff to its present value. The simulation is iteratively repeated for numerous paths to yield a distribution of possible option prices. The present values of option payoffs across all simulations are aggregated and averaged at maturity. Statistical measures such as mean and standard deviation are then computed to know central tendency and variability of option's value.

The frequency distribution of simulated option values over BABA stock is illustrated below.



The following European option prices has been derived from Monte Carlo Simulation:

Call Option Price = 0.39755421222590964 ≅ 0.4

Put Option Price = 0.2806965462662029 ≅ 0.28

5.3. Black- Scholes Model

The Black-Scholes Model (BSM) offers an analytical expression to calculate the fair market value of an option under certain assumptions. It assumes that stock price follows Geometric Brownian Motion, capturing randomness and volatility over time. BSM also assumes a constant and known risk-free rate, along with an efficient financial market with no implied transaction costs or taxes. BSM is used for pricing non-dividend paying European-style options.

The Black-Scholes formula is given as follows:

For Call option price: $C = S_0 * N(d_1) - K e^{-rT} * N(d_2)$

For Put option price: $P = K e^{-rT} * N(-d_2) - S_0 * N(-d_1)$

Where:

S_0: current stock price

K: strike price

r: risk-free rate

T: time to expiration

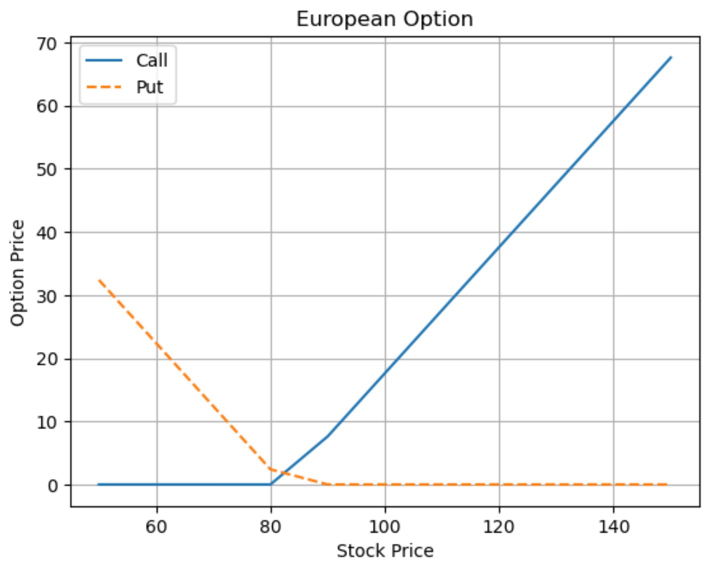
N (d_1) and N (d_2) are cumulative distribution functions of the standard normal distribution.

$$d_1 = \frac{\ln\left(\frac{So}{K}\right) + (r + \frac{\sigma^2}{2})T}{\sigma\sqrt{T}} \qquad d_2 = d_1 - \sigma\sqrt{T}$$

The calculated non-dividend European option prices using the Black-Scholes Model in Python are:

Call option price = 0.38

Put option price = 0.28



The graph above shows that call option has value when stock price is greater than strike price, and put option has value when stock price is lower than strike price of USD 83.50.

5.4. Option Pricing Models – Results Summary

Binomial trees are relatively simple, providing a discrete and step-by-step representation. Monte Carlo is more flexible and can handle complex scenarios. Black-Scholes is a closed-form solution, offering simplicity for specific cases.

Additionally, Binomial trees can be adapted for American-style options. Monte Carlo is versatile and can handle a wide range of scenarios. Black-Scholes is specific to European-style options.

	Binomial Tree	**Monte Carlo Simulation**	**Black-Scholes Model**
Call Option Price	0.3	0.4	0.38
Put Option Price	0.2	0.28	0.28

The above table reflects that the European no-dividend paying option derivative price of Alibaba stock is approximately similar in value with utilized pricing models, however, the results of Monte Carlo Simulation and Black-Scholes Model have lesser differences than the results from Binomial Tree test.

6. Greeks

"Greeks” are used to help traders and investors understand the sensitivity of option prices to various factors, where each Greek provides insights into how changes in underlying variables can affect the value and risk of an options position.

6.1. Delta (Δ)

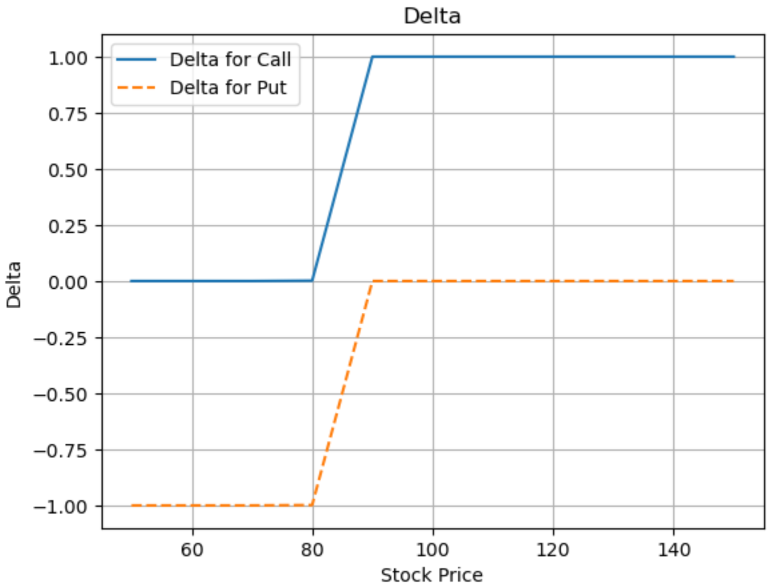
Delta measures the sensitivity of an option's price to changes in the price of the underlying asset.

$$\Delta = \frac{\partial c}{\partial S}$$

Where:

∂: the first derivative
c: the (call or put) option’s price
S: the underlying asset’s price.

The call option delta ranges between 0 and 1, while the put option delta ranges between 0 and -1. For Alibaba option parameters, the call option has a computed delta of 0.55, suggesting that for every $1 change in the underlying stock, the option price is expected to change by $0.55. Similarly, the put option has a delta value of -0.44, indicating that for every $1 change in stock price, the put option price will change by $0.44.



6.2. Gamma (Γ)

Gamma measures the rate of change of an option's delta in response to fluctuations in the underlying asset's price, assessing the curvature or convexity of the option's price curve. A higher gamma indicates that delta is more sensitive to small changes in the underlying price.

$$\Gamma = \left(\frac{\partial^2 \Pi}{\partial S^2}\right)$$

Where:

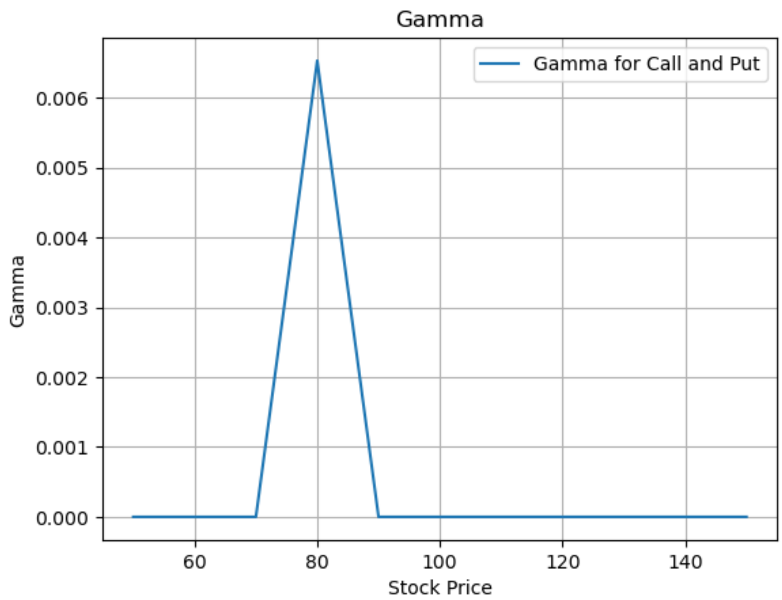
Π is the portfolio of options.

For European call or put option on a non-dividend paying stock, gamma is given as:

$$\Gamma = \frac{N'(d1)}{S\sigma\sqrt{T}}$$

The gamma value remains consistent for both call and put option prices.

The gamma for Alibaba stock options is 0.4793, indicating that for every one-point change in the underlying asset's price, the delta of the option will change by approximately 0.4793.



6.3. Theta (Θ)

Theta, or time decay, quantifies the expected decrease in an option's premium over time, assuming all other factors remain constant. It signifies the rate at which the option loses value as it approaches the expiration date. Theta is negative for long calls and positive for long puts.

For a European option on a non-dividend-paying stock:

$$\Theta(\text{call}) = -\frac{SN\prime(d1)\sigma}{2\sqrt{T}} - rKe^{-rT}\,N(d_2)$$

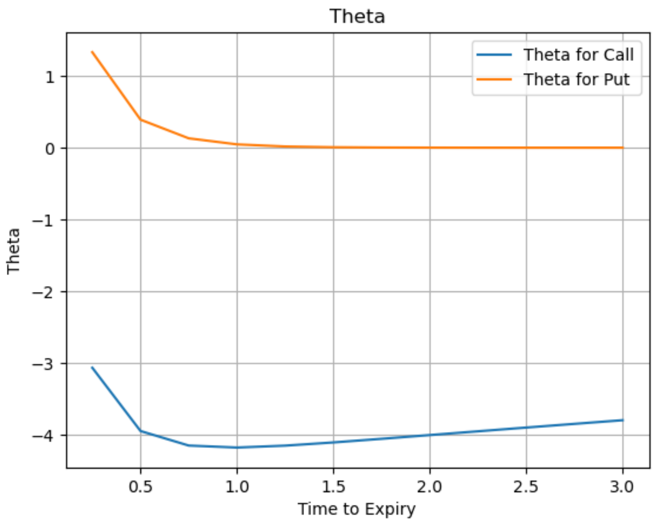
$$\Theta(\text{put}) = -\frac{SN\prime(d1)\sigma}{2\sqrt{T}} + rKe^{-rT}\,N(-d_2)$$

Where:

$N'(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$ is the probability density function for a standard normal distribution.

In our Alibaba example, the theta for the call option is -3.06268, implying that for each day that passes, the option's premium is expected to decrease by approximately $3.06.

Whereas theta for put option is derived as 1.32896, indicating for each day that passes, the option's premium is expected to decrease by approximately $1.33.



6.4. Vega (ν)

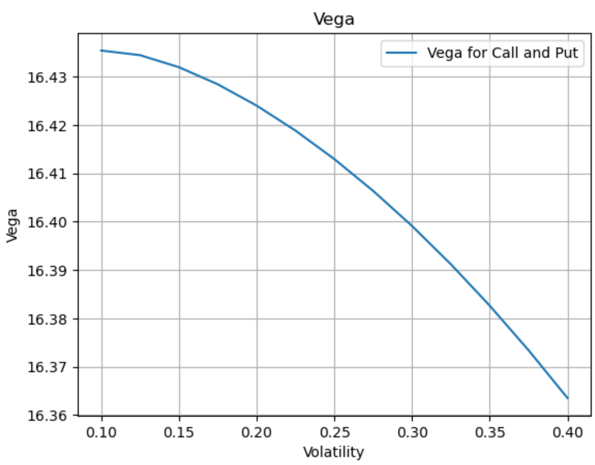
Vega measures how much an option's price is anticipated to change for a one-percentage-point shift in implied volatility, with higher vega indicating increased sensitivity to volatility changes.

For a European call or put option on a non-dividend-paying stock, vega is given by:

$$\nu = S\sqrt{T}\, N'(d_1)$$

The vega value for Alibaba stock options is 16.312, implying that a one-percentage-point rise in implied volatility is expected to increase the option's price by approximately $16.31. Conversely, a one-percentage-point decrease in implied volatility is anticipated to decrease the option's price by the same amount.

In a long position, higher volatility is advantageous, potentially leading to increased profits. Conversely, in a short position, a negative vega suggests that the position benefits from a decrease in implied volatility, potentially resulting in profits for the option seller as a reduction in volatility can lead to a decline in the option's premium.



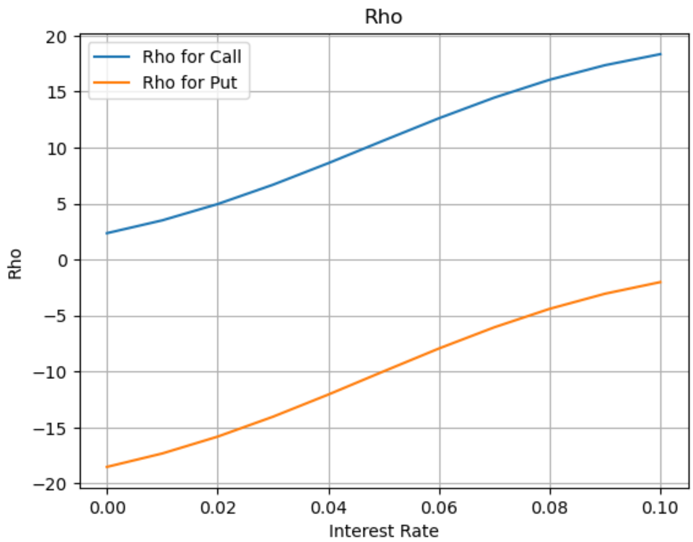
6.5. Rho (ρ)

Rho signifies the expected change in an option's price for a one-percentage-point shift in interest rates, with greater relevance for longer-term options.

$$\text{rho(call)} = KTe^{-rT}\, N(d_2)$$

$$\text{rho(put)} = -\,KTe^{-rT}\, N(-d_2)$$

A positive rho value of 11.3048 for the call option implies that the option's price is expected to increase with a one-percentage-point rise in the risk-free interest rate. Conversely, a rho value of -9.2938 for the put option suggests that the option's price is expected to decrease with a one-percentage-point increase in the risk-free interest rate.



6.6. Summary of Greeks Analysis

<table>
<tr><th>Greeks</th><th>Purpose</th><th>Call Option Value</th><th>Put Option Value</th><th>Long Call</th><th>Long Put</th><th>Short Call</th><th>Short Put</th></tr>
<tr><td>Delta (Δ)</td><td>Indicates directional risk</td><td>0.55277</td><td>-0.44722</td><td>A positive delta means position benefits from an increase in the price of the underlying asset.</td><td>A negative delta indicates that position benefits from an increase in the price of the underlying asset.</td><td>A negative delta suggests potential benefits from decreasing underlying asset prices.</td><td>A positive delta suggests potential benefits from increasing underlying asset prices.</td></tr>
<tr><td>Gamma (Γ)</td><td>Shows how delta changes with underlying price movements</td><td>0.47932</td><td>0.47932</td><td colspan="2">Positive gamma indicates positive relation between option's delta and price of the underlying stock</td><td colspan="2">Negative gamma indicates negative relation between option's delta and price of underlying stock</td></tr>
<tr><td>Theta (Θ)</td><td>Measures time decay</td><td>-3.06268</td><td>1.32895</td><td>A negative theta indicates option holder is facing time decay.</td><td>A positive theta means that the option holder is facing time decay.</td><td colspan="2">Traders benefit from positive theta in short positions, as they collect premium due to time decay.</td></tr>
<tr><td>Vega (ν)</td><td>Assesses sensitivity to changes in volatility</td><td>16.31218</td><td>16.31218</td><td colspan="2">A positive vega means option's premium is expected to rise with higher volatility, potentially leading to increased profits.</td><td colspan="2">A negative vega suggests that your position benefits from a decrease in implied volatility.</td></tr>
<tr><td>Rho (ρ)</td><td>Measure the impact of interest rate changes</td><td>11.30485</td><td>-9.29383</td><td>A positive rho means position may benefit from an increase in interest rates.</td><td>A negative rho means position may benefit from an increase in interest rates.</td><td>A negative rho suggests potential benefits from decreasing interest rates</td><td>A positive rho suggests potential benefits from increasing interest rates</td></tr>
</table>

References

Alibaba, 2023. *Culture and Values.* [Online]
Available at: https://www.alibabagroup.com/en-US/about-alibaba

MarketLine, 2023. *Alibaba Group Holding Limited MarketLine Company Profile.* [Online]
Available at: https://web.p.ebscohost.com/ehost/pdfviewer/pdfviewer?vid=2&sid=23229156-a9e9-4070-9159-3a608319d2ef%40redis
[Accessed 02 December 2023].

U.S. Department of the Treasury, 2023. *Daily Treasury Bill Rates.* [Online]
Available at: https://home.treasury.gov/resource-center/data-chart-center/interest-rates/TextView?type=daily_treasury_bill_rates&field_tdr_date_value=2023

Yahoo Finance, 2023. *Alibaba Group Holding Limited (BABA).* [Online]
Available at: https://uk.finance.yahoo.com/quote/BABA?p=BABA&.tsrc=fin-srch
[Accessed 03 December 2023].

Appendix

In [14]:
```
pip install yfinance==0.2.28
```

```
Requirement already satisfied: yfinance==0.2.28 in c:\users\sande\anaconda3\lib\site-packages (0.2.28)Note: you may need to res
tart the kernel to use updated packages.

Requirement already satisfied: pandas>=1.3.0 in c:\users\sande\anaconda3\lib\site-packages (from yfinance==0.2.28) (2.0.3)
Requirement already satisfied: numpy>=1.16.5 in c:\users\sande\anaconda3\lib\site-packages (from yfinance==0.2.28) (1.24.3)
Requirement already satisfied: requests>=2.31 in c:\users\sande\anaconda3\lib\site-packages (from yfinance==0.2.28) (2.31.0)
Requirement already satisfied: multitasking>=0.0.7 in c:\users\sande\anaconda3\lib\site-packages (from yfinance==0.2.28) (0.0.1
1)
Requirement already satisfied: lxml>=4.9.1 in c:\users\sande\anaconda3\lib\site-packages (from yfinance==0.2.28) (4.9.3)
Requirement already satisfied: appdirs>=1.4.4 in c:\users\sande\anaconda3\lib\site-packages (from yfinance==0.2.28) (1.4.4)
Requirement already satisfied: pytz>=2022.5 in c:\users\sande\anaconda3\lib\site-packages (from yfinance==0.2.28) (2023.3.post
1)
Requirement already satisfied: frozendict>=2.3.4 in c:\users\sande\anaconda3\lib\site-packages (from yfinance==0.2.28) (2.3.8)
Requirement already satisfied: beautifulsoup4>=4.11.1 in c:\users\sande\anaconda3\lib\site-packages (from yfinance==0.2.28) (4.
12.2)
Requirement already satisfied: html5lib>=1.1 in c:\users\sande\anaconda3\lib\site-packages (from yfinance==0.2.28) (1.1)
Requirement already satisfied: soupsieve>1.2 in c:\users\sande\anaconda3\lib\site-packages (from beautifulsoup4>=4.11.1->yfinan
ce==0.2.28) (2.4)
Requirement already satisfied: six>=1.9 in c:\users\sande\anaconda3\lib\site-packages (from html5lib>=1.1->yfinance==0.2.28)
(1.16.0)
Requirement already satisfied: webencodings in c:\users\sande\anaconda3\lib\site-packages (from html5lib>=1.1->yfinance==0.2.2
8) (0.5.1)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\sande\anaconda3\lib\site-packages (from pandas>=1.3.0->yfinan
ce==0.2.28) (2.8.2)
Requirement already satisfied: tzdata>=2022.1 in c:\users\sande\anaconda3\lib\site-packages (from pandas>=1.3.0->yfinance==0.2.
28) (2023.3)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\sande\anaconda3\lib\site-packages (from requests>=2.31->yfi
nance==0.2.28) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\sande\anaconda3\lib\site-packages (from requests>=2.31->yfinance==0.2.2
8) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\sande\anaconda3\lib\site-packages (from requests>=2.31->yfinance=
=0.2.28) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\sande\anaconda3\lib\site-packages (from requests>=2.31->yfinance=
=0.2.28) (2023.7.22)
```

In [15]:
```
import yfinance as yf
import numpy as np
import pandas as pd
import matplotlib as mpl
```

In [16]:
```
initial_data = yf.download("BABA ^VIX", start="2021-11-01", end="2023-11-01")
```

```
[*********************100%%**********************]  2 of 2 completed
```

In [17]:
```
initial_data.head()
```

Out[17]:

	Adj Close		Close		High		Low		Open		Volume	
	BABA	^VIX	BABA	^VIX	BABA	^VIX	BABA	^VIX	BABA	^VIX	BABA	^VIX
Date												
2021-11-01	170.169998	16.410000	170.169998	16.410000	171.895004	17.700001	165.800003	16.32	165.839996	16.850000	17609500	0
2021-11-02	162.899994	16.030001	162.899994	16.030001	167.100006	16.650000	162.759995	15.89	167.100006	16.540001	19072900	0
2021-11-03	166.240005	15.100000	166.240005	15.100000	166.649994	16.389999	163.110001	14.90	165.360001	16.110001	13569700	0
2021-11-04	164.789993	15.440000	164.789993	15.440000	169.940002	16.139999	164.500000	14.73	169.279999	15.060000	16669200	0
2021-11-05	158.729996	16.480000	158.729996	16.480000	164.794998	17.020000	158.300003	14.95	164.794998	15.590000	22098200	0

In [18]:
```
initial_data['Adj Close'].head()
```

Out[18]:

	BABA	^VIX
Date		
2021-11-01	170.169998	16.410000
2021-11-02	162.899994	16.030001
2021-11-03	166.240005	15.100000
2021-11-04	164.789993	15.440000
2021-11-05	158.729996	16.480000

Stock Price Movement Plot

In [2]:
```
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Define the stock symbol and time period
stock_symbol = "BABA"
start_date = "2021-11-01"
end_date = "2023-10-31"

# Download historical stock data
stock_data = yf.download(stock_symbol, start=start_date, end=end_date)

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

# Plot the stock price movement
plt.figure(figsize=(10, 6))
plt.plot(stock_data['Adj Close'])
plt.title(f'{stock_symbol} Stock Price Movement ({start_date} to {end_date})')
plt.xlabel('Date')
plt.ylabel('Stock Price (USD)')
plt.grid(True)
plt.show()
```

Annualized Average Return and Standard Deviation

In [20]:
```
df = pd.read_csv('BABA.csv')
df.head()
```

Out[20]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2021-11-01	165.839996	171.895004	165.800003	170.169998	170.169998	17609500
1	2021-11-02	167.100006	167.100006	162.759995	162.899994	162.899994	19072900
2	2021-11-03	165.360001	166.649994	163.110001	166.240005	166.240005	13569700
3	2021-11-04	169.279999	169.940002	164.500000	164.789993	164.789993	16669200
4	2021-11-05	164.794998	164.794998	158.300003	158.729996	158.729996	22098200

In [21]:
```
df.tail()
```

Out[21]:

	Date	Open	High	Low	Close	Adj Close	Volume
498	2023-10-25	81.300003	82.000000	80.779999	81.029999	81.029999	11390200
499	2023-10-26	81.264999	83.089996	81.199997	82.510002	82.510002	13010400
500	2023-10-27	83.870003	84.120003	82.480003	82.820000	82.820000	10795600
501	2023-10-30	83.629997	84.239998	83.011002	83.139999	83.139999	8980500
502	2023-10-31	81.940002	82.540001	80.879997	82.540001	82.540001	12094300

In [22]:
```
# Calculate normal return
import numpy as np
normal_return = []
for i in range(0,len(df)-1):
    adjclose_yesterday = df.iloc[i]['Adj Close']
    adjclose_today = df.iloc[i+1]['Adj Close']
    x = (adjclose_today - adjclose_yesterday) / adjclose_yesterday
    normal_return.append(x)
normal_return[:5]
```

Out[22]:
```
[-0.04272200790647009,
 0.020503444585762257,
 -0.008722401085105759,
 -0.03677405945396217,
 0.021609072553621097]
```

In [23]:
```
# Calculate log return
log_return = []
for i in range(0,len(df)-1):
    adjclose_yesterday = df.iloc[i]['Adj Close']
    adjclose_today = df.iloc[i+1]['Adj Close']
    y = np.log(adjclose_today / adjclose_yesterday)
    log_return.append(y)
log_return[:5]
```

Out[23]:
```
[-0.04366144685480603,
 0.02029607865232781,
 -0.008760663883581784,
 -0.037467273160207634,
 0.021378906426615243]
```

In [24]:
```
#Changing the list variable to numpy array:
normal_return=np.array(normal_return)
#calculating the mean and standard deviation on normal returns using numpy:
mean_nr = normal_return.mean() * len(normal_return)
sd_nr = normal_return.std() * (len(normal_return) ** 0.5)
print(f'Annualized Average Return: {mean_nr:.4%}')
print(f'Annualized Standard Deviation: {sd_nr:.4%}')
```

```
Annualized Average Return: -35.4614%
Annualized Standard Deviation: 87.6670%
```

In [25]:
```
#Changing the list variable to numpy array:
log_return=np.array(log_return)
#calculating the mean and standard deviation on log returns using numpy:
mean_logr = log_return.mean() * len(log_return)
sd_logr = log_return.std() * (len(log_return) ** 0.5)
print(f'Annualized Average Return: {mean_logr:.4%}')
print(f'Annualized Standard Deviation: {sd_logr:.4%}')
```

```
Annualized Average Return: -72.3515%
Annualized Standard Deviation: 85.0817%
```

Summary Statistics

In [26]:
```
data.describe().round(2)
```

Out[26]:

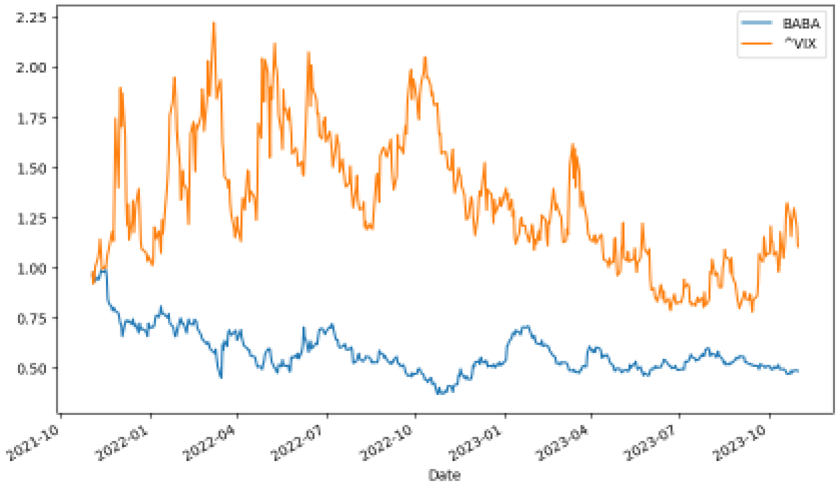
	BABA	^VIX
count	503.00	503.00
mean	98.23	21.78
std	18.90	5.39
min	63.15	12.82
25%	85.85	17.64
50%	92.90	21.09
75%	108.63	25.71
max	170.17	36.45

In [27]:
```
rets = np.log(data / data.shift(1))
rets.head().round(4)
```

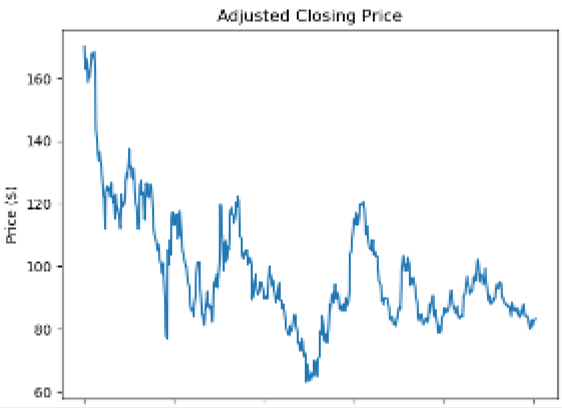
Out[27]:

Date	BABA	^VIX
2021-11-01	NaN	NaN
2021-11-02	-0.0437	-0.0234
2021-11-03	0.0203	-0.0598
2021-11-04	-0.0088	0.0223
2021-11-05	-0.0375	0.0652

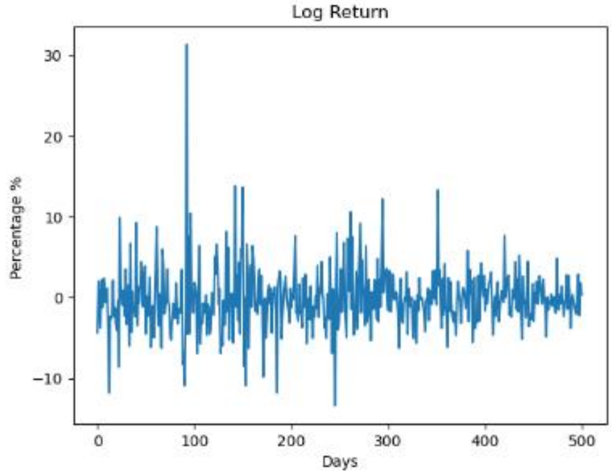
In [28]:
```
rets.cumsum().apply(np.exp).plot(figsize=(10, 6))
```
Out[28]: <Axes: xlabel='Date'>



In [25]:
```
import matplotlib.pyplot as plt
fig = plt.figure()
plt.plot(df['Adj Close']) #choosing the series you want to plot
plt.xlabel('Days')
plt.ylabel('Price ($)')
plt.title('Adjusted Closing Price')
```
Out[25]: Text(0.5, 1.0, 'Adjusted Closing Price')



Out[26]: Text(0.5, 1.0, 'Log Return')



In [42]:
```
# Calculate 3-month rolling standard deviation
rolling_std_3months = stock_data['Daily_Return'].rolling(window=3*21).std()  # Assuming 21 trading days in a month
print("3-Month Rolling Standard Deviation:")
print(rolling_std_3months.dropna())
```

```
3-Month Rolling Standard Deviation:
Date
2022-02-01    0.038571
2022-02-02    0.038446
2022-02-03    0.038337
2022-02-04    0.038340
2022-02-07    0.038782
                ...
2023-10-25    0.021272
2023-10-26    0.020214
2023-10-27    0.020088
2023-10-30    0.019987
2023-10-31    0.019051
Name: Daily_Return, Length: 440, dtype: float64
```

Binomial Tree

In [30]:
```
S0 = 82.5               # spot stock price
K = 83.5                # strike
T = 3/12                  # maturity
r = 0.0533                  # risk free rate
sigma = 0.02                # standard deviation (volatility)
N = 3                    # number of periods or number of time steps
payoff = "call"           # payoff
```

In [31]:
```
dT = float(T) / N                             # Delta t
u = np.exp(sigma * np.sqrt(dT))               # up factor
d = 1.0 / u                                   # down factor
print(f"Up factor (u): {u}")
print(f"Down factor (d): {d}")
```

```
Up factor (u): 1.0057902014799276
Down factor (d): 0.9942431319459984
```

In [140]:
```
S = np.zeros((N + 1, N + 1))
S[0, 0] = S0
z = 1
for t in range(1, N + 1): #looping forwards, from 1 to N
    for i in range(z):  #looping forwards, from 0 to z-1
        S[i, t] = S[i, t-1] * u
        S[i+1, t] = S[i, t-1] * d
    z += 1  # same as z=z+1
```

In [141]:
```
S
```

Out[141]:
```
array([[82.5       , 82.97769162, 83.45814917, 83.94138867],
       [ 0.        , 82.02505839, 82.5       , 82.97769162],
       [ 0.        ,  0.        , 81.55285095, 82.02505839],
       [ 0.        ,  0.        ,  0.        , 81.08336194]])
```

In [132]:
```
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
p
```

Out[132]: 0.8840696345329057

```
In [142]: 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[142]: array([[0.        , 0.        , 0.        , 0.44138867],
                 [0.        , 0.        , 0.        , 0.        ],
                 [0.        , 0.        , 0.        , 0.        ],
                 [0.        , 0.        , 0.        , 0.        ]])

In [143]: # 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[143]: array([[0.3009495 , 0.34192917, 0.38848895, 0.44138867],
                 [0.        , 0.        , 0.        , 0.        ],
                 [0.        , 0.        , 0.        , 0.        ],
                 [0.        , 0.        , 0.        , 0.        ]])

In [144]: print('European ' + payoff, str( V[0,0]))

          European call 0.3009495045480285

In [145]: payoff = "put"

In [146]: 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[146]: array([[0.        , 0.        , 0.        , 0.        ],
                 [0.        , 0.        , 0.        , 0.52230838],
                 [0.        , 0.        , 0.        , 1.47494161],
                 [0.        , 0.        , 0.        , 2.41663806]])

In [147]: # 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[147]: array([[0.19569214, 0.12576412, 0.06028305, 0.        ],
                 [0.        , 0.73646819, 0.62994328, 0.52230838],
                 [0.        , 0.        , 1.57709233, 1.47494161],
                 [0.        , 0.        , 0.        , 2.41663806]])

In [148]: print('European ' + payoff, str( V[0,0]))

          European put 0.19569213544319675
```

Monte Carlo Simulation

In [149]:
```python
def mcs_simulation_np(p):
    M = p
    I = p
    dt = T / M
    S = np.zeros((M + 1, I))
    S[0] = S0
    rn = np.random.standard_normal(S.shape)
    for t in range(1, M + 1):
        S[t] = S[t-1] * np.exp((r - sigma ** 2 / 2) * dt + sigma * np.sqrt(dt) * rn[t])
    return S
```

In [150]:
```python
T = 3/12
r = 0.0533
sigma = 0.02
S0 = 82.5
K = 83.5
```

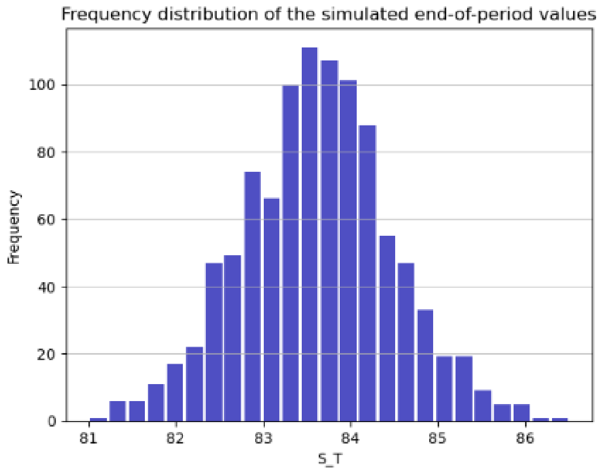
In [151]:
```python
S = mcs_simulation_np(1000)
```

In [152]:
```python
S = np.transpose(S)
S
```

Out[152]:
```
array([[82.5       , 82.50265945, 82.48599024, ..., 81.78674597,
        81.77232531, 81.7293812 ],
       [82.5       , 82.451711  , 82.42524998, ..., 84.46526148,
        84.4812303 , 84.45391686],
       [82.5       , 82.52877512, 82.53467674, ..., 83.60181368,
        83.58398848, 83.57728377],
       ...,
       [82.5       , 82.46170769, 82.46349961, ..., 84.42314594,
        84.46674284, 84.50049666],
       [82.5       , 82.4903098 , 82.4651873 , ..., 82.19475589,
        82.2061349 , 82.18363534],
       [82.5       , 82.50236212, 82.5128208 , ..., 84.87776189,
        84.87596385, 84.90882861]])
```

In [156]:
```python
import matplotlib.pyplot as plt
n, bins, patches = plt.hist(x=S[:,-1], bins='auto', color='#0504aa',alpha=0.7, rwidth=0.85)
plt.xlabel('S_T')
plt.ylabel('Frequency')
plt.title('Frequency distribution of the simulated end-of-period values')
```

Out[156]: Text(0.5, 1.0, 'Frequency distribution of the simulated end-of-period values')



In [154]:
```python
c = np.exp(-r*T)*np.mean(np.maximum(S[:,-1] - K,0))
print('European call', str(c))
```

```
European call 0.39755421222590964
```

In [155]:
```python
p = np.exp(-r*T)*np.mean(np.maximum(K - S[:,-1],0))
print('European put', str(p))
```

```
European put 0.2806965462662029
```

Black-Scholes Model

In [159]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as si
import yfinance as yf
```

In [160]:
```
def euro_option_bs(S, K, T, r, vol, payoff):

    #S: spot price
    #K: strike price
    #T: time to maturity
    #r: risk free rate
    #vol: volatility of underlying asset
    #payoff: call or put

    d1 = (np.log(S / K) + (r + 0.5 * vol ** 2) * T) / (vol * np.sqrt(T))
    d2 = (np.log(S / K) + (r - 0.5 * vol ** 2) * T) / (vol * np.sqrt(T))
    if payoff == "call":
        option_value = S * si.norm.cdf(d1, 0.0, 1.0) - K * np.exp(-r * T) * si.norm.cdf(d2, 0.0, 1.0)
    elif payoff == "put":
        option_value = - S * si.norm.cdf(-d1, 0.0, 1.0) + K * np.exp(-r * T) * si.norm.cdf(-d2, 0.0, 1.0)

    return option_value
```

In [169]:
```
call = euro_option_bs(82.50, 83.50, 0.25, 0.0533, 0.02, 'call')
print('The Black-Scholes call price is', round(call, 2))
```

```
The Black-Scholes call price is 0.38
```

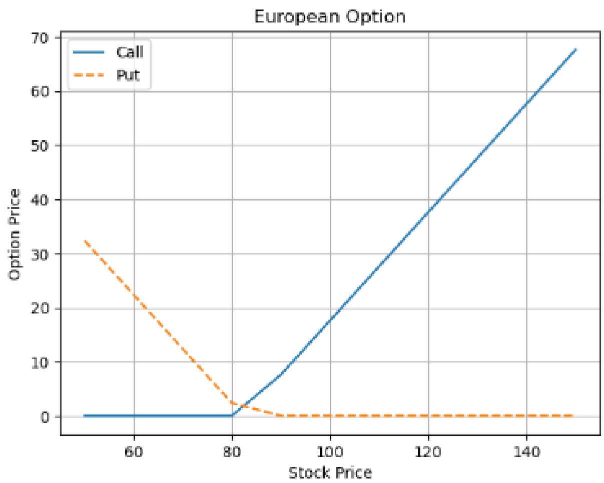
In [171]:
```
put = euro_option_bs(82.50, 83.50, 0.25, 0.0533, 0.02, 'put')
print('The Black-Scholes put price is', round(put, 2))
```

```
The Black-Scholes put price is 0.28
```

In [172]:
```
S = np.linspace(50,150,11)
Call = np.zeros((len(S),1))
Put = np.zeros((len(S),1))
for i in range(len(S)):
    Call [i] = euro_option_bs(S[i], 83.50, 0.25, 0.0533, 0.02, 'call')
    Put [i] = euro_option_bs(S[i], 83.50, 0.25, 0.0533, 0.02, 'put')
```

In [173]:
```
fig = plt.figure()
plt.plot(S, Call, '-')
plt.plot(S, Put, '--')
plt.grid()
plt.xlabel('Stock Price')
plt.ylabel('Option Price')
plt.title('European Option')
plt.legend(['Call','Put'])
```

Out[173]: `<matplotlib.legend.Legend at 0x263b6d9bed0>`



Delta

```
In [174]: def delta(S, K, T, r, vol, payoff):

              d1 = (np.log(S / K) + (r + 0.5 * vol ** 2) * T) / (vol * np.sqrt(T))
              if payoff == "call":
                  delta = si.norm.cdf(d1, 0.0, 1.0)
              elif payoff == "put":
                  delta =  si.norm.cdf(d1, 0.0, 1.0)-1

              return delta
```

```
In [176]: delta(82.50, 83.50, 0.25, 0.0533, 0.02, 'call')

Out[176]: 0.5527712920791097
```

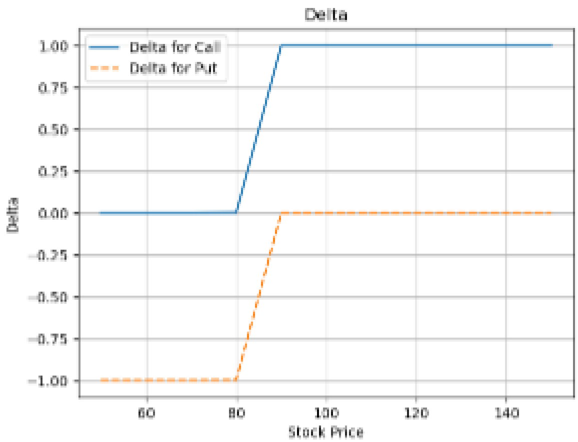
```
In [177]: delta(82.50, 83.50, 0.25, 0.0533, 0.02, 'put')

Out[177]: -0.4472287079208903
```

```
In [178]: S = np.linspace(50,150,11)
          Delta_Call = np.zeros((len(S),1))
          Delta_Put = np.zeros((len(S),1))
          for i in range(len(S)):
              Delta_Call [i] = delta(S[i], 83.50, 0.25, 0.0533, 0.02, 'call')
              Delta_Put [i] = delta(S[i], 83.50, 0.25, 0.0533, 0.02, 'put')
```

```
In [179]: fig = plt.figure()
          plt.plot(S, Delta_Call, '-')
          plt.plot(S, Delta_Put, '--')
          plt.grid()
          plt.xlabel('Stock Price')
          plt.ylabel('Delta')
          plt.title('Delta')
          plt.legend(['Delta for Call','Delta for Put'])

Out[179]: <matplotlib.legend.Legend at 0x263b7175390>
```



Gamma

```
In [180]: def gamma(S, K, T, r,  vol, payoff):

              d1 = (np.log(S / K) + (r  + 0.5 * vol ** 2) * T) / (vol * np.sqrt(T))

              gamma = si.norm.pdf(d1, 0.0, 1.0) / (vol *  np.sqrt(T) * S)


              return gamma
```

```
In [181]: gamma(82.50, 83.50, 0.25, 0.0533, 0.02, 'call')
```

```
Out[181]: 0.4793296114915965
```

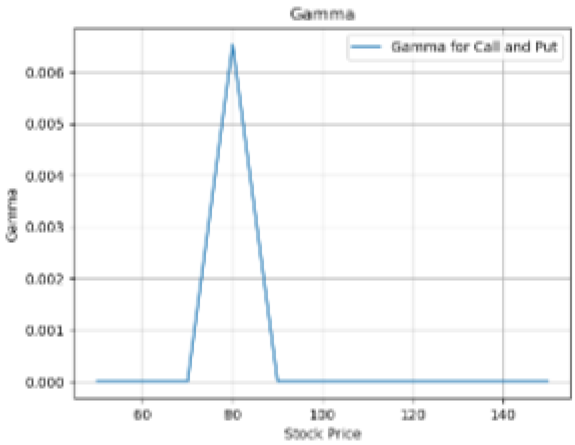
```
In [182]: gamma(82.50, 83.50, 0.25, 0.0533, 0.02, 'put')
```

```
Out[182]: 0.4793296114915965
```

```
In [183]: S = np.linspace(50,150,11)
          Gamma = np.zeros((len(S),1))
          for i in range(len(S)):
              Gamma [i] = gamma(S[i], 83.50, 0.25, 0.0533, 0.02, 'call')
```

```
In [184]: fig = plt.figure()
          plt.plot(S, Gamma, '-')
          plt.grid()
          plt.xlabel('Stock Price')
          plt.ylabel('Gamma')
          plt.title('Gamma')
          plt.legend(['Gamma for Call and Put'])
```

```
Out[184]: <matplotlib.legend.Legend at 0x263b7330e90>
```



Theta

In [185]:
```
def theta(S, K, T, r, vol, payoff):

    d1 = (np.log(S / K) + (r + 0.5 * vol ** 2) * T) / (vol * np.sqrt(T))
    d2 = (np.log(S / K) + (r - 0.5 * vol ** 2) * T) / (vol * np.sqrt(T))
    N_d1_prime=1/np.sqrt(2 * np.pi) * np.exp(-d1**2/2)

    if payoff == "call":
        theta = - S * N_d1_prime * vol / (2 * np.sqrt(T)) - r * K * np.exp(-r * T) * si.norm.cdf(d2, 0.0, 1.0)
    elif payoff == "put":
        theta = - S * N_d1_prime * vol / (2 * np.sqrt(T)) + r * K * np.exp(-r * T) * si.norm.cdf(-d2, 0.0, 1.0)

    return theta
```

In [186]:
```
theta(82.50, 83.50, 0.25, 0.0533, 0.02, 'call')
```

Out[186]: -3.062681990566221

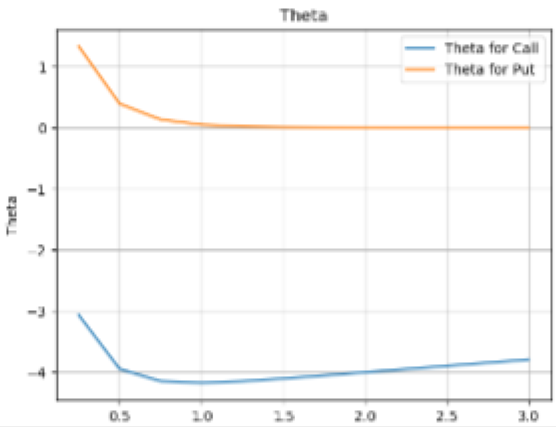
In [187]:
```
theta(82.50, 83.50, 0.25, 0.0533, 0.02, 'put')
```

Out[187]: 1.3289577916604922

In [188]:
```
T = np.linspace(0.25,3,12)
Theta_Call = np.zeros((len(T),1))
Theta_Put = np.zeros((len(T),1))
for i in range(len(T)):
    Theta_Call [i] = theta(82.50, 83.50, T[i], 0.0533, 0.02, 'call')
    Theta_Put [i] = theta(82.50, 83.50, T[i], 0.0533, 0.02, 'put')
```

In [189]:
```
fig = plt.figure()
plt.plot(T, Theta_Call, '-')
plt.plot(T, Theta_Put, '-')
plt.grid()
plt.xlabel('Time to Expiry')
plt.ylabel('Theta')
plt.title('Theta')
plt.legend(['Theta for Call', 'Theta for Put'])
```

Out[189]: <matplotlib.legend.Legend at 0x263b7332010>



Rho

```
In [190]: def rho(S, K, T, r, vol, payoff):

              d1 = (np.log(S / K) + (r + 0.5 * vol ** 2) * T) / (vol * np.sqrt(T))
              d2 = (np.log(S / K) + (r - 0.5 * vol ** 2) * T) / (vol * np.sqrt(T))
              if payoff == "call":
                  rho =  K * T * np.exp(-r * T) * si.norm.cdf(d2, 0.0, 1.0)
              elif payoff == "put":
                  rho = - K * T * np.exp(-r * T) * si.norm.cdf(-d2, 0.0, 1.0)

              return rho
```

```
In [191]: rho(82.50, 83.50, 0.25, 0.0533, 0.02, 'call')
```

```
Out[191]: 11.3048525184019
```

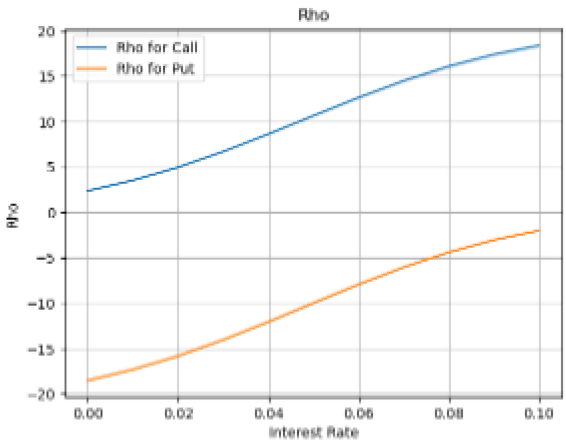
```
In [192]: rho(82.50, 83.50, 0.25, 0.0533, 0.02, 'put')
```

```
Out[192]: -9.293833139321894
```

```
In [194]: r = np.linspace(0,0.1,11)
          Rho_Call = np.zeros((len(r),1))
          Rho_Put = np.zeros((len(r),1))
          for i in range(len(r)):
              Rho_Call [i] = rho(82.50, 83.50, 0.25, r[i], 0.02, 'call')
              Rho_Put [i] = rho(82.50, 83.50, 0.25, r[i], 0.02, 'put')
```

```
In [195]: fig = plt.figure()
          plt.plot(r, Rho_Call, '-')
          plt.plot(r, Rho_Put, '-')
          plt.grid()
          plt.xlabel('Interest Rate')
          plt.ylabel('Rho')
          plt.title('Rho')
          plt.legend(['Rho for Call', 'Rho for Put'])
```

```
Out[195]: <matplotlib.legend.Legend at 0x263b7320358>
```



Vega

```
In [196]: def vega(S, K, T, r, vol, payoff):

              d1 = (np.log(S / K) + (r + 0.5 * vol ** 2) * T) / (vol * np.sqrt(T))
              N_d1_prime=1/np.sqrt(2 * np.pi) * np.exp(-d1**2/2)
              vega = S * np.sqrt(T) * N_d1_prime

              return vega
```

```
In [197]: vega(82.50, 83.50, 0.25, 0.0533, 0.02, 'call')
```

```
Out[197]: 16.312185841073394
```

```
In [198]: vega(82.50, 83.50, 0.25, 0.0533, 0.02, 'put')
```

```
Out[198]: 16.312185841073394
```

```
In [199]: vol = np.linspace(0.1,0.4,13)
          Vega = np.zeros((len(vol),1))
          for i in range(len(vol)):
              Vega [i] = vega(82.50, 83.50, 0.25, 0.0533, vol[i], 'call')
```

```
In [200]: fig = plt.figure()
          plt.plot(vol, Vega, '-')
          plt.grid()
          plt.xlabel('Volatility')
          plt.ylabel('Vega')
          plt.title('Vega')
          plt.legend(['Vega for Call and Put'])
```

```
Out[200]: <matplotlib.legend.Legend at 0x263b86d7190>
```