DERIVATIVE PRICING AND RISK MANAGEMENT

ASSIGNMENT 2: OPTION PRICING ALGORITHMS

Adharsha Sam Edwin Sam Devahi	1004361
James Huang Yong Heng	1005461
Low Haoron	1005018
Ryan Seah Yu Wei	1005123

EXECUTIVE SUMMARY

In the field of Quantitative Finance, call and put option premiums are computed using option pricing methodologies such as Put-Call Parity, Monte Carlo Simulation, Binomial Option Pricing Model and Black-Scholes Method. The objective of this assignment is to implement the aforementioned pricing methodologies to find the prices of European call and put options for a stock that pays annualized dividends continuously throughout the year. Our team chose to code in Python as the programming language contains powerful packages such as NumPy, Math and SciPy that support quantitative modelling for finance. Since we are dealing with European options, Put-Call Parity can be applied to find respective put and call option prices for call and put option prices derived from Monte Carlo Simulation, Binomial Option Pricing Model and Black-Scholes Method (and vice versa). The call and put option prices derived using Monte Carlo Simulation, Binomial Option Pricing Model and Black Scholes Method and those calculated using the Put-Call Parity do not have significant variations. The variations in the same call and put option prices calculated using Binomial Option Pricing Model and the Put-Call Parity are a lot lower than the variations in option prices calculated using Monte Carlo Simulation and Put-Call Parity. However, as the number of simulations in the Monte Carlo Simulation was increased, the variations in the option prices calculated using these two methods decreased as well. Given the assumption of constant implied volatility, Black-Scholes Method is used to accurately calculate the prices of call and put options and these prices are then compared with those computed from Monte Carlo Simulation and Binomial Option Pricing Model. It is also interesting to note that the call and put option prices calculated using Binomial Option Pricing Model are the most similar to the call and put option prices calculated using Black-Scholes Method.

MONTE CARLO SIMULATION

We created two Python functions – `ParityPutPrice()` and `ParityCallPrice()` – that are used to calculate the put and call option prices from known call and put option prices respectively.

Put-Call Parity to calculate put option price using known call option price for a stock that pays annualized dividends:

$$P = C - S_0 e^{-q\,T} + X e^{-r_f\,T}$$

Put-Call Parity to calculate call option price using known put option price for a stock that pays annualized dividends:

$$C = P + S_0 e^{-q\,T} - X e^{-r_f\,T}$$

where P = put option price, C = call option price, S_0 = current stock price, X = strike price, r_f = annualized risk-free rate (continuously compounded), T = time to maturity and q = annualized dividends

(Refer Section 1. Put-Call Parity for pricing European options with dividends in the Appendix for relevant Python code)

We created two Python functions – `MCSforCallOption()` and `MCSforPutOption()` – to perform option pricing using the Monte Carlo Simulation which generates '`simcount`' random paths that the call and put option prices could take.

The stock price at maturity (S_T) is calculated for the random paths of call and put option prices using the following formula:

$$S_{T,i} = S_t \, Exp(r_{T,i})$$

$$r_{T,i} \sim N\left(\left(r_f - q - \frac{1}{2}\sigma^2\right) * T, \sigma^2 T\right)$$

where sample returns ($r_{T,i}$) is generated from a normal distribution for each iteration i, r_f = annualized risk-free rate (continuously compounded), q = annualized dividends, σ = annualized volatility of log returns and T = time to maturity

Call and put option payouts are calculated as follows:

$$Call\ Option\ Payout_i = Max(S_{T,i} - X, 0)$$

$$Put\ Option\ Payout_i = Max(X - S_{T,i}, 0)$$

Expected call and put option payouts are calculated by taking the arithmetic average of the sampled call and put option payouts as follows:

$$E(Call\ Option\ Payout) = \frac{1}{N}\sum_{i=1}^{N} Call\ Option\ Payout_i$$

$$E(Put\ Option\ Payout) = \frac{1}{N}\sum_{i=1}^{N} Put\ Option\ Payout_i$$

where $E(Call\ Option\ Payout)$ = `ExpectedCallPayout`,
$E(Put\ Option\ Payout)$ = `ExpectedPutPayout` and
N = number of simulations generated (`simcount`)

The expected call and put option payouts are then discounted back to t = 0 by multiplying $Exp(-r_f\ T)$ to calculate the call and put option prices as follows:

$$Call\ Option\ Price = E(Put\ Option\ Payout) * Exp(-r_f\ T)$$

$$Put\ Option\ Price = E(Put\ Option\ Payout) * Exp(-r_f\ T)$$

where $Call\ Option\ Price$ = `CallOptionPrice` and $Put\ Option\ Price$ = `PutOptionPrice`

(Refer Sections 2.1. Monte Carlo Simulation Approach 1 and 2.2. Monte Carlo Simulation Approach 2 in the Appendix for relevant Python code)

Use Monte Carlo Simulation to value the put and call. What are the option prices when the number of iterations, N, equals 1,000? 10,000? 100,000?

For each simulation:

1. Report the option price.

```
Call option price from 1000 Simulations of Monte Carlo Simulation: $19.94
Call option price from 10000 Simulations of Monte Carlo Simulation: $21.37
Call option price from 100000 Simulations of Monte Carlo Simulation: $21.16

Put option price from 1000 Simulations of Monte Carlo Simulation: $17.53
Put option price from 10000 Simulations of Monte Carlo Simulation: $16.63
Put option price from 100000 Simulations of Monte Carlo Simulation: $16.8
```

(Refer Sections 2.1. Monte Carlo Simulation Approach 1 and 2.2. Monte Carlo Simulation Approach 2 in the Appendix for relevant Python code)

2. If the simulation is for the call option, price the put option using Put-Call Parity and the call option value from the Monte Carlo Simulation.

```
Put option price for the simulated call price of $19.94 is $15.64
Put option price for the simulated call price of $21.37 is $17.07
Put option price for the simulated call price of $21.16 is $16.86
```

(Refer Sections 1. Put-Call Parity for pricing European options with dividends and 2.1. Monte Carlo Simulation Approach 1 in the Appendix for relevant Python code)

3. If the simulation is for the put option, price the call option using Put-Call Parity and the call option value from the Monte Carlo Simulation.

```
Call option price for the simulated put price of $17.53 is $21.83
Call option price for the simulated put price of $16.63 is $20.93
Call option price for the simulated put price of $16.8 is $21.1
```

(Refer Sections 1. Put-Call Parity for pricing European options with dividends and 2.2. Monte Carlo Simulation Approach 2 in the Appendix for relevant Python code)

4. Report the simulated option prices and option prices using Put-Call Parity. What happens to the difference between the put-call parity price and the simulated option price?

Simulated option prices:

```
Call option price from 1000 Simulations of Monte Carlo Simulation: $19.94
Call option price from 10000 Simulations of Monte Carlo Simulation: $21.37
Call option price from 100000 Simulations of Monte Carlo Simulation: $21.16

Put option price from 1000 Simulations of Monte Carlo Simulation: $17.53
Put option price from 10000 Simulations of Monte Carlo Simulation: $16.63
Put option price from 100000 Simulations of Monte Carlo Simulation: $16.8
```

Option prices using Put-Call Parity:

```
Put option price for the simulated call price of $19.94 is $15.64
Put option price for the simulated call price of $21.37 is $17.07
Put option price for the simulated call price of $21.16 is $16.86
```

```
Call option price for the simulated put price of $17.53 is $21.83
Call option price for the simulated put price of $16.63 is $20.93
Call option price for the simulated put price of $16.8 is $21.1
```

Difference between the put-call parity price and the simulated option price:

```
Variation in call option prices calculated using Monte Carlo Simulation (1000
Simulations) & Put-Call Parity: $1.89
Variation in call option prices calculated using Monte Carlo Simulation (10000
Simulations) & Put-Call Parity: $0.44
Variation in call option prices calculated using Monte Carlo Simulation
(100000 Simulations) & Put-Call Parity: $0.06

Variation in put option prices calculated using Monte Carlo Simulation (1000
Simulations) & Put-Call Parity: $1.89
Variation in put option prices calculated using Monte Carlo Simulation (10000
Simulations) & Put-Call Parity: $0.44
Variation in put option prices calculated using Monte Carlo Simulation (100000
Simulations) & Put-Call Parity: $0.06
```

The variations in option prices calculated using these two methods decrease as we increase the number of simulations in the Monte Carlo Simulation.

(Refer Section 2.3. Comparing Call & Put Option Prices calculated using methods: Monte Carlo Simulation & Put-Call Parity in the Appendix for relevant Python code)

BINOMIAL OPTION PRICING MODEL

We created two Python functions – `BOPMforCallOption()` and `BOPMforPutOption()` – to perform option pricing using the iterative Binomial Option Pricing Model which generates two possible call and put option price payouts for upward and downward movement factors at each node of '`binomial_time_step`' binomial tree time steps.

We need to calculate expected option price payouts using risk-neutral probabilities for upward and downward movement factors derived as follows.

Upward and downward movement factors:

$$u = e^{-\sigma * \sqrt{dt}}$$

$$d = \frac{1}{u}$$

Risk-neutral probabilities for upward and downward movement factors:

$$Pu = \frac{e^{r_f * dt} - d}{u - d}$$

$$Pd = 1 - Pu$$

where `dt` = `T` (time to maturity) / `binomial_time_step` (number of time steps)

Probability at each node:

$$P = \binom{N}{i} (P_u)^i (P_d)^{N-i}$$

where N = `binomial_time_step` and i = number of upward movements in stock price

Call Option Price:

$$Call_i = \sum_i^N \binom{N}{i} P^i (1-P)^{N-i} Max(S_0 u^i d^{N-i} - X, 0)$$

where $Call_i$ = `CallOption`

$$Call\ Option\ Price = Call_i * Exp(-r_f\ T)$$

where $Call\ Option\ Price$ = `DiscCallOption`

Put Option Price:

$$Put_i = \sum_i^N \binom{N}{i} P^i (1-P)^{N-i} Max(X - S_0 u^i d^{N-i}, 0)$$

where Put_i = `PutOption`

$$Put\ Option\ Price = Put_i * Exp(-r_f\ T)$$

where $Put\ Option\ Price$ = `DiscPutOption`

(Refer Sections 3.1. Binomial Option Pricing Model Approach 1 and 3.2. Binomial Option Pricing Model Approach 2 in the Appendix for relevant Python code)

Find the price of both put and call option when the stock follows a binomial process.

1. What are the option prices when the number of time steps in the binomial tree equals 100? 500? 1,000?

```
Call option price from 100 binomial tree time steps of Binomial Option Pricing
Model: $21.15
Call option price from 500 binomial tree time steps of Binomial Option Pricing
Model: $21.1
Call option price from 1000 binomial tree time steps of Binomial Option
Pricing Model: $21.11

Put option price from 100 binomial tree time steps of Binomial Option Pricing
Model: $16.85
Put option price from 500 binomial tree time steps of Binomial Option Pricing
Model: $16.81
Put option price from 1000 binomial tree time steps of Binomial Option Pricing
Model: $16.81
```

(Refer Sections 3.1. Binomial Option Pricing Model Approach 1 and 3.2. Binomial Option Pricing Model Approach 2 in the Appendix for relevant Python code)

2. If the binomial tree is for the call option, price the put option using put-call parity and the call option value from the binomial model.

```
Put option price for the call price of $21.15 derived from Binomial Option
Pricing Model is $16.85
Put option price for the call price of $21.1 derived from Binomial Option
Pricing Model is $16.8
Put option price for the call price of $21.11 derived from Binomial Option
Pricing Model is $16.81
```

(Refer Sections 1. Put-Call Parity for pricing European options with dividends and 3.1. Binomial Option Pricing Model Approach 1 in the Appendix for relevant Python code)

3. If the binomial tree is for the put option, price the call option using put-call parity and the put option value from the binomial model.

```
Call option price for the put price of $16.85 derived from Binomial Option
Pricing Model is $21.15
Call option price for the put price of $16.81 derived from Binomial Option
Pricing Model is $21.11
Call option price for the put price of $16.81 derived from Binomial Option
Pricing Model is $21.11
```

(Refer Sections 1. Put-Call Parity for pricing European options with dividends and 3.2. Binomial Option Pricing Model Approach 2 in the Appendix for relevant Python code)

4. Report the binomial option prices and option prices using Put-Call Parity. What happens to the difference between the put-call parity price and the binomial option price?

Binomial option prices:

```
Call option price from 100 binomial tree time steps of Binomial Option Pricing
Model: $21.15
Call option price from 500 binomial tree time steps of Binomial Option Pricing
Model: $21.1
Call option price from 1000 binomial tree time steps of Binomial Option
Pricing Model: $21.11
```

```
Put option price from 100 binomial tree time steps of Binomial Option Pricing
Model: $16.85
Put option price from 500 binomial tree time steps of Binomial Option Pricing
Model: $16.81
Put option price from 1000 binomial tree time steps of Binomial Option Pricing
Model: $16.81
```

Option prices using Put-Call Parity:

```
Call option price for the put price of $16.85 derived from Binomial Option
Pricing Model is $21.15
Call option price for the put price of $16.81 derived from Binomial Option
Pricing Model is $21.11
Call option price for the put price of $16.81 derived from Binomial Option
Pricing Model is $21.11
```

```
Put option price for the call price of $21.15 derived from Binomial Option
Pricing Model is $16.85
Put option price for the call price of $21.1 derived from Binomial Option
Pricing Model is $16.8
Put option price for the call price of $21.11 derived from Binomial Option
Pricing Model is $16.81
```

Difference between the put-call parity price and the binomial option price:

```
Variation in call option prices calculated using Binomial Option Pricing Model
(100 binomial tree time steps) & Put-Call Parity: $0.0
Variation in call option prices calculated using Binomial Option Pricing Model
(500 binomial tree time steps) & Put-Call Parity: $0.0
Variation in call option prices calculated using Binomial Option Pricing Model
(1000 binomial tree time steps) & Put-Call Parity: $0.0

Variation in put option prices calculated using Binomial Option Pricing Model
(100 binomial tree time steps) & Put-Call Parity: $0.0
Variation in put option prices calculated using Binomial Option Pricing Model
(500 binomial tree time steps) & Put-Call Parity: $0.0
Variation in put option prices calculated using Binomial Option Pricing Model
(1000 binomial tree time steps) & Put-Call Parity: $0.0
```

The variations in option prices calculated using these two methods are equal to zero.

(Refer Section 3.3. Comparing Call & Put Option Prices calculated using methods: Binomial Option Pricing Model & Put-Call Parity in the Appendix for relevant Python code)

BLACK-SCHOLES METHOD

We created two Python functions – `BSMforCallOption()` and `BSMforPutOption()` – to calculate the call and put option prices using the Black-Scholes Method.

Call option price for a stock that pays annualized dividends using Black-Scholes Method:

$$C = Exp(-q\,T) * S_0 * N(d_1) - Exp\left(-r_f\,T\right) * X * N(d_2)$$

Put option price for a stock that pays annualized dividends using Black-Scholes Method:

$$P = Exp\left(-r_f\,T\right) * X * N(-d_2) - Exp(-q\,T) * S_0 * N(-d_1)$$

where $d_1 = \frac{ln\left(\frac{S_0}{X}\right) + \left(r_f - q + 0.5\sigma^2\right) * T}{\sigma\sqrt{T}}$ and $d_2 = d_1 - \sigma\sqrt{T}$

(Refer Sections 4.1. Black-Scholes Method Approach 1 and 4.2. Black-Scholes Method Approach 2 in the Appendix for relevant Python code)

4. Compare the Monte Carlo and Binomial option prices to the prices from the Black-Scholes option pricing model. Which model, if any, is closest in value to the option price from Black-Scholes?

Upon comparing the call and put option prices calculated using the Black-Scholes Method to those derived from Monte Carlo Simulation and Binomial Option Pricing Model, we find that the option prices calculated using the Binomial Option Pricing Model are the closest to those calculated using the Black-Scholes Method for both European call and put options. This is an expected result as it is consistent with the theory which states that the Black-Scholes Method is essentially just a special case of the Binomial Option Pricing Model where the number of binomial tree time steps extends to infinity.

(Refer Section 4.3. Comparing Call & Put Option Prices calculated using methods: Monte Carlo Simulation, Binomial Option Pricing Model & Black-Scholes Method in the Appendix for relevant Python code)

BIBLIOGRAPHY

The computer code attached in the appendix contains Python functions for Put-Call Parity, Monte Carlo Simulation, Binomial Option Pricing Model, Black-Scholes Method used to price European call and put options for an annualized dividend-paying stock. The following articles were used to ensure the correctness of code logic implemented to solve the unique set of Assignment 2 problems.

Pipis, G. (2020). Pricing European Options with Monte Carlo. R-bloggers. https://www.r-bloggers.com/2020/12/pricing-of-european-options-with-monte-carlo/

John. (2020). Binomial Option Pricing Model with Python. CodeArmo. https://www.codearmo.com/python-tutorial/options-trading-binomial-pricing-model

---

APPENDIX: COMPUTER CODE

MODEL PARAMETERS

```python
sigma = 0.2  # annualized implied volatility of log returns
S0 = 200     # stock price
q = 0.03     # annualized; continuous payment
rf = 0.06    # annualized; continuous compounding
X = 205      # both call & put options have the same strike price
T = 1.583    # time to expiry (in years). both options expire 1 year & 7 months from now (399 trading days where 1 year has 252 trading days)
```

✓ 0.0s Python

Installing essential Python packages...

```python
import numpy as np              # for array operations

import math                     # for mathematical operations

import scipy
from scipy import stats         # for statistical methods
```

✓ 0.0s Python

1. PUT-CALL PARITY FOR PRICING EUROPEAN OPTIONS WITH DIVIDENDS

```python
def ParityPutPrice(C):
    P = C + X * np.exp(-rf * T) - S0 * np.exp(-q * T)
    return round(P, 2)
```

✓ 0.0s Python

```python
def ParityCallPrice(P):
    C = P + S0 * np.exp(-q * T) - X * np.exp(-rf * T)
    return round(C, 2)
```

✓ 0.0s Python

2. MONTE CARLO SIMULATION FOR PRICING EUROPEAN OPTIONS WITH DIVIDENDS

2.1. Monte Carlo Simulation Approach 1

- Find Call Option Price using Monte Carlo Simulation
- Find Put Option Price using Put-Call Parity

```
def MCSforCallOption(simcount):

    np.random.seed(123)                                                 # to ensure reproducible results
    Z = np.random.normal(size = simcount)                               # generating random variables that follow a normal distribution
    St = S0 * np.exp((rf - q - sigma**2/2) * T) * np.exp(sigma * np.sqrt(T) * Z)  # stock price at maturity

    ExpectedCallPayout = (1/simcount) * np.sum(np.maximum(St - X, 0))   # expected call option price for 'n' simulations
    CallOptionPrice = ExpectedCallPayout * np.exp(-rf * T)              # discounted call option price
    return round(CallOptionPrice, 2)
```

✓ 0.0s Python

```
print("Call option price from 1000 simulations of Monte Carlo Simulation: ${0}".format(MCSforCallOption(1000)))
print("Call option price from 10000 simulations of Monte Carlo Simulation: ${0}".format(MCSforCallOption(10000)))
print("Call option price from 100000 simulations of Monte Carlo Simulation: ${0}".format(MCSforCallOption(100000)))
```

✓ 0.0s Python

```
Call option price from 1000 simulations of Monte Carlo Simulation: $19.94
Call option price from 10000 simulations of Monte Carlo Simulation: $21.37
Call option price from 100000 simulations of Monte Carlo Simulation: $21.16
```

```
print("Put option price for the simulated call price of ${0} is ${1}".format(MCSforCallOption(1000), ParityPutPrice(MCSforCallOption(1000))))
print("Put option price for the simulated call price of ${0} is ${1}".format(MCSforCallOption(10000), ParityPutPrice(MCSforCallOption(10000))))
print("Put option price for the simulated call price of ${0} is ${1}".format(MCSforCallOption(100000), ParityPutPrice(MCSforCallOption(100000))))
```

✓ 0.0s Python

```
Put option price for the simulated call price of $19.94 is $15.64
Put option price for the simulated call price of $21.37 is $17.07
Put option price for the simulated call price of $21.16 is $16.86
```

2.2. Monte Carlo Simulation Approach 2

- Find Put Option Price using Monte Carlo Simulation
- Find Call Option Price using Put-Call Parity

```
def MCSforPutOption(simcount):

    np.random.seed(123)                                                 # to ensure reproducible results
    Z = np.random.normal(size = simcount)                               # generating random variables that follow a normal distribution
    St = S0 * np.exp((rf - q - sigma**2/2) * T) * np.exp(sigma * np.sqrt(T) * Z)  # stock price at maturity

    ExpectedPutPayout = (1/simcount) * np.sum(np.maximum(X - St, 0))    # expected put option price for 'n' simulations
    PutOptionPrice = ExpectedPutPayout * np.exp(-rf * T)                # discounted put option price
    return round(PutOptionPrice, 2)
```

✓ 0.0s Python

```
print("Put option price from 1000 simulations of Monte Carlo Simulation: ${0}".format(MCSforPutOption(1000)))
print("Put option price from 10000 simulations of Monte Carlo Simulation: ${0}".format(MCSforPutOption(10000)))
print("Put option price from 100000 simulations of Monte Carlo Simulation: ${0}".format(MCSforPutOption(100000)))
```

✓ 0.0s Python

```
Put option price from 1000 simulations of Monte Carlo Simulation: $17.53
Put option price from 10000 simulations of Monte Carlo Simulation: $16.63
Put option price from 100000 simulations of Monte Carlo Simulation: $16.8
```

```
print("Call option price for the simulated put price of ${0} is ${1}".format(MCSforPutOption(1000), ParityCallPrice(MCSforPutOption(1000))))
print("Call option price for the simulated put price of ${0} is ${1}".format(MCSforPutOption(10000), ParityCallPrice(MCSforPutOption(10000))))
print("Call option price for the simulated put price of ${0} is ${1}".format(MCSforPutOption(100000), ParityCallPrice(MCSforPutOption(100000))))
```

✓ 0.0s Python

```
Call option price for the simulated put price of $17.53 is $21.83
Call option price for the simulated put price of $16.63 is $20.93
Call option price for the simulated put price of $16.8 is $21.1
```

2.3. Comparing Call & Put Option Prices calculated using methods: Monte Carlo Simulation & Put-Call Parity

```
print("Variation in call option prices calculated using Monte Carlo Simulation (1000 Simulations) & Put-Call Parity: ${0}".format(round(abs(MCSforCallOption(1000) - ParityCallPrice(MCSforPutOption(1000))), 2)))
print("Variation in call option prices calculated using Monte Carlo Simulation (10000 Simulations) & Put-Call Parity: ${0}".format(round(abs(MCSforCallOption(10000) - ParityCallPrice(MCSforPutOption(10000))), 2)))
print("Variation in call option prices calculated using Monte Carlo Simulation (100000 Simulations) & Put-Call Parity: ${0}".format(round(abs(MCSforCallOption(100000) - ParityCallPrice(MCSforPutOption(100000))), 2)))
```

✓ 0.0s Python

```
Variation in call option prices calculated using Monte Carlo Simulation (1000 Simulations) & Put-Call Parity: $1.89
Variation in call option prices calculated using Monte Carlo Simulation (10000 Simulations) & Put-Call Parity: $0.44
Variation in call option prices calculated using Monte Carlo Simulation (100000 Simulations) & Put-Call Parity: $0.06
```

```
print("Variation in put option prices calculated using Monte Carlo Simulation (1000 Simulations) & Put-Call Parity: ${0}".format(round(abs(MCSforPutOption(1000) - ParityPutPrice(MCSforCallOption(1000))), 2)))
print("Variation in put option prices calculated using Monte Carlo Simulation (10000 Simulations) & Put-Call Parity: ${0}".format(round(abs(MCSforPutOption(10000) - ParityPutPrice(MCSforCallOption(10000))), 2)))
print("Variation in put option prices calculated using Monte Carlo Simulation (100000 Simulations) & Put-Call Parity: ${0}".format(round(abs(MCSforPutOption(100000) - ParityPutPrice(MCSforCallOption(100000))), 2)))
```

✓ 0.0s Python

```
Variation in put option prices calculated using Monte Carlo Simulation (1000 Simulations) & Put-Call Parity: $1.89
Variation in put option prices calculated using Monte Carlo Simulation (10000 Simulations) & Put-Call Parity: $0.44
Variation in put option prices calculated using Monte Carlo Simulation (100000 Simulations) & Put-Call Parity: $0.06
```

3. BINOMIAL OPTION PRICING MODEL FOR PRICING EUROPEAN OPTIONS WITH DIVIDENDS

3.1. Binomial Option Pricing Model Approach 1

- Find Call Option Price using Binomial Option Pricing Model
- Find Put Option Price using Put-Call Parity

```python
def combination(N, i):
    return math.factorial(N)/(math.factorial(N - i) * math.factorial(i))
```
✓ 0.0s Python

```python
def BOPMforCallOption(binomial_time_step):

    dt = T/binomial_time_step

    u = np.exp(sigma * np.sqrt(dt))           # upward movement factor
    d = 1/u                                   # downward movement factor
    Pu = (np.exp((rf - q) * dt) - d)/(u - d)  # risk-neutral probability of upward movement factor
    Pd = 1 - Pu                               # risk-neutral probability of downward movement factor

    CallOption = 0

    for i in reversed(range(0, binomial_time_step+1)):
        St = S0 * u**i * d**(binomial_time_step - i)                              # stock price at a specific binomial tree step
        P = combination(binomial_time_step, i) * Pu**i * Pd**(binomial_time_step - i)  # upward/downward probability at a specific binomial tree step
        CallOption += P * np.maximum(St - X, 0)                                   # call option price at a specific binomial tree step
    DiscCallOption = np.exp(-rf * T) * CallOption                                 # discounted expected call option price
    return round(DiscCallOption, 2)
```
✓ 0.0s Python

```python
print("Call option price from 100 binomial tree time steps of Binomial Option Pricing Model: ${0}".format(BOPMforCallOption(100)))
print("Call option price from 500 binomial tree time steps of Binomial Option Pricing Model: ${0}".format(BOPMforCallOption(500)))
print("Call option price from 1000 binomial tree time steps of Binomial Option Pricing Model: ${0}".format(BOPMforCallOption(1000)))
```
✓ 0.1s Python

```
Call option price from 100 binomial tree time steps of Binomial Option Pricing Model: $21.15
Call option price from 500 binomial tree time steps of Binomial Option Pricing Model: $21.1
Call option price from 1000 binomial tree time steps of Binomial Option Pricing Model: $21.11
```

```python
print("Put option price for the call price of ${0} derived from Binomial Option Pricing Model is ${1}".format(BOPMforCallOption(100), ParityPutPrice(BOPMforCallOption(100))))
print("Put option price for the call price of ${0} derived from Binomial Option Pricing Model is ${1}".format(BOPMforCallOption(500), ParityPutPrice(BOPMforCallOption(500))))
print("Put option price for the call price of ${0} derived from Binomial Option Pricing Model is ${1}".format(BOPMforCallOption(1000), ParityPutPrice(BOPMforCallOption(1000))))
```
✓ 0.2s Python

```
Put option price for the call price of $21.15 derived from Binomial Option Pricing Model is $16.85
Put option price for the call price of $21.1 derived from Binomial Option Pricing Model is $16.8
Put option price for the call price of $21.11 derived from Binomial Option Pricing Model is $16.81
```

3.2. Binomial Option Pricing Model Approach 2

- Find Put Option Price using Binomial Option Pricing Model
- Find Call Option Price using Put-Call Parity

```python
def BOPMforPutOption(binomial_time_step):

    dt = T/binomial_time_step

    u = np.exp(sigma * np.sqrt(dt))           # upward movement factor
    d = 1/u                                   # downward movement factor
    Pu = (np.exp((rf - q) * dt) - d)/(u - d)  # risk-neutral probability of upward movement factor
    Pd = 1 - Pu                               # risk-neutral probability of downward movement factor

    PutOption = 0

    for i in reversed(range(0, binomial_time_step+1)):
        St = S0 * u**i * d**(binomial_time_step - i)                              # stock price at a specific binomial tree step
        P = combination(binomial_time_step, i) * Pu**i * Pd**(binomial_time_step - i)  # upward/downward probability at a specific binomial tree step
        PutOption += P * np.maximum(X - St, 0)                                    # put option price at a specific binomial tree step
    DiscPutOption = np.exp(-rf * T) * PutOption                                   # discounted expected put option price
    return round(DiscPutOption, 2)
```
✓ 0.0s Python

```python
print("Put option price from 100 binomial tree time steps of Binomial Option Pricing Model: ${0}".format(BOPMforPutOption(100)))
print("Put option price from 500 binomial tree time steps of Binomial Option Pricing Model: ${0}".format(BOPMforPutOption(500)))
print("Put option price from 1000 binomial tree time steps of Binomial Option Pricing Model: ${0}".format(BOPMforPutOption(1000)))
```
✓ 0.1s Python

```
Put option price from 100 binomial tree time steps of Binomial Option Pricing Model: $16.85
Put option price from 500 binomial tree time steps of Binomial Option Pricing Model: $16.81
Put option price from 1000 binomial tree time steps of Binomial Option Pricing Model: $16.81
```

```python
print("Call option price for the put price of ${0} derived from Binomial Option Pricing Model is ${1}".format(BOPMforPutOption(100), ParityCallPrice(BOPMforPutOption(100))))
print("Call option price for the put price of ${0} derived from Binomial Option Pricing Model is ${1}".format(BOPMforPutOption(500), ParityCallPrice(BOPMforPutOption(500))))
print("Call option price for the put price of ${0} derived from Binomial Option Pricing Model is ${1}".format(BOPMforPutOption(1000), ParityCallPrice(BOPMforPutOption(1000))))
```
✓ 0.2s Python

```
Call option price for the put price of $16.85 derived from Binomial Option Pricing Model is $21.15
Call option price for the put price of $16.81 derived from Binomial Option Pricing Model is $21.11
Call option price for the put price of $16.81 derived from Binomial Option Pricing Model is $21.11
```

3.3. Comparing Call & Put Option Prices calculated using methods: Binomial Option Pricing Model & Put-Call Parity

```python
print("Variation in call option prices calculated using Binomial Option Pricing Model (100 binomial tree time steps) & Put-Call Parity: ${0}".format(round(abs(BOPMforCallOption(100) - ParityCallPrice(BOPMforPutOption(100))), 1)))
print("Variation in call option prices calculated using Binomial Option Pricing Model (500 binomial tree time steps) & Put-Call Parity: ${0}".format(round(abs(BOPMforCallOption(500) - ParityCallPrice(BOPMforPutOption(500))), 1)))
print("Variation in call option prices calculated using Binomial Option Pricing Model (1000 binomial tree time steps) & Put-Call Parity: ${0}".format(round(abs(BOPMforCallOption(1000) - ParityCallPrice(BOPMforPutOption(1000))), 1)))
```
✓ 0.2s Python

```
Variation in call option prices calculated using Binomial Option Pricing Model (100 binomial tree time steps) & Put-Call Parity: $0.0
Variation in call option prices calculated using Binomial Option Pricing Model (500 binomial tree time steps) & Put-Call Parity: $0.0
Variation in call option prices calculated using Binomial Option Pricing Model (1000 binomial tree time steps) & Put-Call Parity: $0.0
```

```python
print("Variation in put option prices calculated using Binomial Option Pricing Model (100 binomial tree time steps) & Put-Call Parity: ${0}".format(round(abs(BOPMforPutOption(100) - ParityPutPrice(BOPMforCallOption(100))), 1)))
print("Variation in put option prices calculated using Binomial Option Pricing Model (500 binomial tree time steps) & Put-Call Parity: ${0}".format(round(abs(BOPMforPutOption(500) - ParityPutPrice(BOPMforCallOption(500))), 1)))
print("Variation in put option prices calculated using Binomial Option Pricing Model (1000 binomial tree time steps) & Put-Call Parity: ${0}".format(round(abs(BOPMforPutOption(1000) - ParityPutPrice(BOPMforCallOption(1000))), 1)))
```
✓ 0.2s Python

```
Variation in put option prices calculated using Binomial Option Pricing Model (100 binomial tree time steps) & Put-Call Parity: $0.0
Variation in put option prices calculated using Binomial Option Pricing Model (500 binomial tree time steps) & Put-Call Parity: $0.0
Variation in put option prices calculated using Binomial Option Pricing Model (1000 binomial tree time steps) & Put-Call Parity: $0.0
```

4. BLACK-SCHOLES METHOD FOR PRICING EUROPEAN OPTIONS WITH DIVIDENDS

4.1. Black Scholes Method Approach 1

- Find Call Option Price using Black-Scholes Method
- Find Put Option Price using Put-Call Parity

```python
from scipy.stats import norm  # for CDF calculation

def BSMforCallOption():
    d1 = (np.log(S0/X) + (rf - q + 0.5*sigma**2)*T)/(sigma * np.sqrt(T))
    d2 = d1 - sigma*np.sqrt(T)
    C = np.exp(-q * T) * S0 * norm.cdf(d1) - X * np.exp(-rf * T) * norm.cdf(d2)
    return round(C, 2)
```

✓ 0.0s Python

```python
print("Call option price calculated using Black-Scholes Method is ${0}".format(BSMforCallOption()))
print("Put option price for the call price of ${0} derived from Black-Scholes Method is ${1}".format(BSMforCallOption(), ParityPutPrice(BSMforCallOption())))
```

✓ 0.0s Python

```
Call option price calculated using Black-Scholes Method is $21.11
Put option price for the call price of $21.11 derived from Black-Scholes Method is $16.81
```

4.2. Black-Scholes Method Approach 2

- Find Put Option Price using Black-Scholes Method
- Find Call Option Price using Put-Call Parity

```python
def BSMforPutOption():
    d1 = (np.log(S0/X) + (rf - q + 0.5*sigma**2)*T)/(sigma * np.sqrt(T))
    d2 = d1 - sigma*np.sqrt(T)
    P = X * np.exp(-rf * T)* norm.cdf(-d2) - S0 * np.exp(-q * T) * norm.cdf(-d1)
    return round(P, 2)
```

✓ 0.0s Python

```python
print("Put option price calculated using Black-Scholes Method is ${0}".format(BSMforPutOption()))
print("Call option price for the put price of ${0} derived from Black-Scholes Method is ${1}".format(BSMforPutOption(), ParityCallPrice(BSMforPutOption())))
```

✓ 0.0s Python

```
Put option price calculated using Black-Scholes Method is $16.81
Call option price for the put price of $16.81 derived from Black-Scholes Method is $21.11
```

4.3. Comparing Call & Put Option Prices calculated using methods: Monte Carlo Simulation, Binomial Option Pricing Model & Black-Scholes Method

```python
print("Variation in call option prices calculated using Monte Carlo Simulation (1000 simulations) & Black-Scholes Method: ${0}".format(round(abs(MCSforCallOption(1000) - BSMforCallOption()), 2)))
print("Variation in call option prices calculated using Monte Carlo Simulation (10000 simulations) & Black-Scholes Method: ${0}".format(round(abs(MCSforCallOption(10000) - BSMforCallOption()), 2)))
print("Variation in call option prices calculated using Monte Carlo Simulation (100000 simulations) & Black-Scholes Method: ${0}".format(round(abs(MCSforCallOption(100000) - BSMforCallOption()), 2)))
```

✓ 0.0s Python

```
Variation in call option prices calculated using Monte Carlo Simulation (1000 simulations) & Black-Scholes Method: $1.17
Variation in call option prices calculated using Monte Carlo Simulation (10000 simulations) & Black-Scholes Method: $0.26
Variation in call option prices calculated using Monte Carlo Simulation (100000 simulations) & Black-Scholes Method: $0.05
```

```python
print("Variation in call option prices calculated using Binomial Option Pricing Model (100 binomial tree time steps) & Black-Scholes Method: ${0}".format(round(abs(BOPMforCallOption(100) - BSMforCallOption()), 2)))
print("Variation in call option prices calculated using Binomial Option Pricing Model (500 binomial tree time steps) & Black-Scholes Method: ${0}".format(round(abs(BOPMforCallOption(500) - BSMforCallOption()), 2)))
print("Variation in call option prices calculated using Binomial Option Pricing Model (1000 binomial tree time steps) & Black-Scholes Method: ${0}".format(round(abs(BOPMforCallOption(1000) - BSMforCallOption()), 2)))
```

✓ 0.1s Python

```
Variation in call option prices calculated using Binomial Option Pricing Model (100 binomial tree time steps) & Black-Scholes Method: $0.04
Variation in call option prices calculated using Binomial Option Pricing Model (500 binomial tree time steps) & Black-Scholes Method: $0.01
Variation in call option prices calculated using Binomial Option Pricing Model (1000 binomial tree time steps) & Black-Scholes Method: $0.0
```

```python
print("Variation in put option prices calculated using Monte Carlo Simulation (1000 simulations) & Black-Scholes Method: ${0}".format(round(abs(MCSforPutOption(1000) - BSMforPutOption()), 2)))
print("Variation in put option prices calculated using Monte Carlo Simulation (10000 simulations) & Black-Scholes Method: ${0}".format(round(abs(MCSforPutOption(10000) - BSMforPutOption()), 2)))
print("Variation in put option prices calculated using Monte Carlo Simulation (100000 simulations) & Black-Scholes Method: ${0}".format(round(abs(MCSforPutOption(100000) - BSMforPutOption()), 2)))
```

✓ 0.0s Python

```
Variation in put option prices calculated using Monte Carlo Simulation (1000 simulations) & Black-Scholes Method: $0.72
Variation in put option prices calculated using Monte Carlo Simulation (10000 simulations) & Black-Scholes Method: $0.18
Variation in put option prices calculated using Monte Carlo Simulation (100000 simulations) & Black-Scholes Method: $0.01
```

```python
print("Variation in put option prices calculated using Binomial Option Pricing Model (100 binomial tree time steps) & Black-Scholes Method: ${0}".format(round(abs(BOPMforPutOption(100) - BSMforPutOption()), 2)))
print("Variation in put option prices calculated using Binomial Option Pricing Model (500 binomial tree time steps) & Black-Scholes Method: ${0}".format(round(abs(BOPMforPutOption(500) - BSMforPutOption()), 2)))
print("Variation in put option prices calculated using Binomial Option Pricing Model (1000 binomial tree time steps) & Black-Scholes Method: ${0}".format(round(abs(BOPMforPutOption(1000) - BSMforPutOption()), 2)))
```

✓ 0.1s Python

```
Variation in put option prices calculated using Binomial Option Pricing Model (100 binomial tree time steps) & Black-Scholes Method: $0.04
Variation in put option prices calculated using Binomial Option Pricing Model (500 binomial tree time steps) & Black-Scholes Method: $0.0
Variation in put option prices calculated using Binomial Option Pricing Model (1000 binomial tree time steps) & Black-Scholes Method: $0.0
```