1. Prices using Binomial Tree method are displayed in the table in (c) part.
2. Price using Black-Scholes: 0.5224453276436325
3. Table:

|  |  |  |
| --- | --- | --- |
| **Number of Timesteps** | **Binomial Tree Solution** | **|E|** |
| 10 | 0.510182 | 0.0122635 |
| 100 | 0.521203 | 0.0012422 |
| 1000 | 0.522321 | 0.0001244 |
| 10000 | 0.522433 | 0.0000124 |

1. The least squares line is: ln|E| = -0.9981 lnN + -2.0992

A screen shot of a graph

Description automatically generated

1. The convergence rate of the Binomial Tree method is: 0.9981

**Python Code:**

# Import libraries

import numpy as np

from scipy.stats import norm

import matplotlib.pyplot as plt

import math

# Parameters

K = 10                                                                  # Strike price of option

r = 0.02                                                                # Constant risk-free interest rate

sigma = 0.25                                                            # Constant volatility of the stock price

T = 0.25                                                                # Time to maturity

S0 = 10                                                                 # Current stock price

# Function to calculate u and d for the binomial tree

def calculate\_u\_d(sigma, T, N):

    dt = T / N

    u = math.exp(sigma \* math.sqrt(dt))                                 # Compute u

    d = 1 / u                                                           # Compute d

    return u, d

# Function to calculate option price using binomial tree method

def calculate\_price\_binomial\_tree(N):

    dt = T / N

    u, d = calculate\_u\_d(sigma, T, N)

    p = (math.exp(r \* dt) - d) / (u - d)                                # Probability of up movement

    stock\_price = [0] \* (N + 1)                                         # Initialize the stock price at maturity

    option\_value = [0] \* (N + 1)                                        # Initialize the option values at maturity

    # Loop through each time step

    for i in range(N + 1):

        stock\_price[i] = S0 \* (u \*\* (N - i)) \* (d \*\* i)                 # Calculate stock price at maturity

        option\_value[i] = max(stock\_price[i] - K, 0)                    # Calculate option value at maturity

    # Calculate option values at earlier time steps using backward recursion

    for j in range(N - 1, -1, -1):

        for i in range(j + 1):

            option\_value[i] = math.exp(-r \* dt) \* (p \* option\_value[i] + (1 - p) \* option\_value[i + 1])

    # Return option value at time 0

    return option\_value[0]

# Function to calculate option price using Black-Scholes formula

def calculate\_price\_black\_scholes():

    d1 = (math.log(S0 / K) + (r + sigma \*\* 2 / 2) \* T) / (sigma \* math.sqrt(T))

    d2 = d1 - sigma \* math.sqrt(T)

    price = S0 \* norm.cdf(d1) - K \* math.exp(-r \* T) \* norm.cdf(d2)     # Black-Scholes formula

    return price

if \_\_name\_\_ == "\_\_main\_\_":

    # (b) Calculate the option price using the Black-Scholes formula

    price\_black\_scholes = calculate\_price\_black\_scholes()

    # (a) Calculate option prices using binomial tree method for number of time steps N = 10, 100, 1000, and 10000

    prices\_binomial\_tree = []

    errors = []

    for N in [10, 100, 1000, 10000]:

        price = calculate\_price\_binomial\_tree(N)

        prices\_binomial\_tree.append(price)

        error = abs(price - price\_black\_scholes)

        errors.append(error)

    # (c) Print the table and Black-Scholes price

    print("Price using Black-Scholes: ", price\_black\_scholes)

    print("N\tBinomial Tree\t|E|")

    for i in range(len(prices\_binomial\_tree)):

        print(f"{[10, 100, 1000, 10000][i]}\t{prices\_binomial\_tree[i]:.6f}\t{errors[i]:.7f}")

    # (d) Create log-log plot of ln|E| vs ln N

    ln\_N = [math.log(N) for N in [10, 100, 1000, 10000]]                # Calculate ln(N) for each value of N

    ln\_E = [math.log(E) for E in errors]                                # Calculate ln(errors) for each error value

    plt.plot(ln\_N, ln\_E, 'bo')                                          # Plot ln|E| vs ln N with blue circles

    plt.xlabel('ln N')                                                  # Set x-axis label as 'ln N'

    plt.ylabel('ln|E|')                                                 # Set y-axis label as 'ln|E|'

    plt.title('Log-Log Plot of Error vs Number of Time Steps')          # Set plot title

    # (d) Perform linear regression to find slope and intercept of the least squares line

    slope, intercept = np.polyfit(ln\_N, ln\_E, 1)

    # (d) Plot the least squares line on the plot with red color

    plt.plot(ln\_N, slope\*np.array(ln\_N) + intercept, 'r')

    # (d) Print the equation of the least squares line

    print(f"The least squares line is: ln|E| = {slope:.4f} lnN + {intercept:.4f}")

    # (e) Calculate the convergence rate (-A) from the slope

    convergence\_rate = -slope

    # (e) Print the convergence rate

    print(f"The convergence rate of the Binomial Tree method is: {convergence\_rate:.4f}")

    # Display the plot

    plt.show()