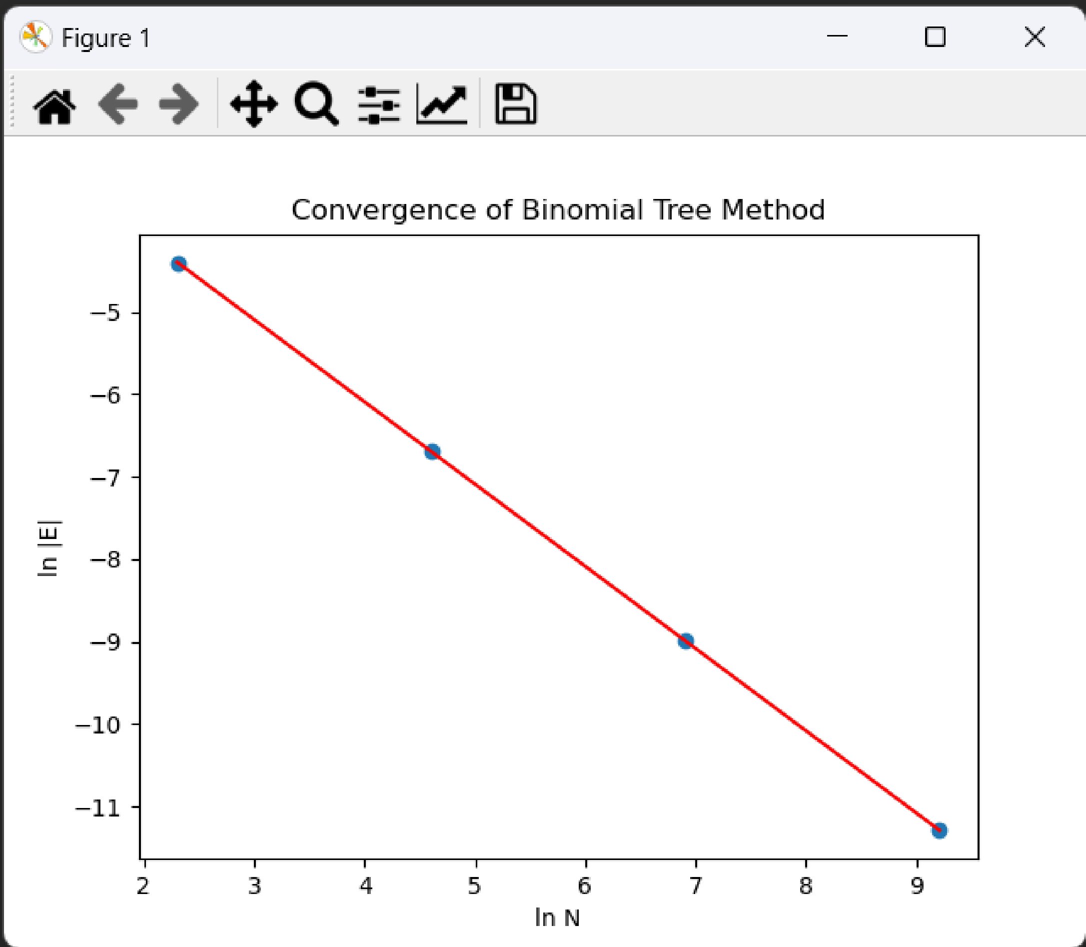
```
# Jack Kwok -- Math 448 -- Project 1B -- Black-Scholes_vs_Binomial-tree
import math
from scipy.stats import norm
import numpy as np
import matplotlib.pyplot as plt



# The cumulative distribution function (CDF) of the standard normal distribution is used in the
Black-Scholes formula
# to calculate the probabilities of certain events occurring. In the code I provided, the CDF
is calculated using the
# norm.cdf() function from the SciPy library. This function takes the value of d1 and d2 as
inputs and returns the
# probability of those values occurring in a standard normal distribution. The norm.cdf()
function calculates the
# probability that a random variable from a standard normal distribution is less than or equal
to a given value. In
# the Black-Scholes formula, d1 and d2 are the arguments passed to norm.cdf(), and the
resulting probabilities are
# used to calculate the call and put option prices.

# European call option on a non-dividend-paying stock
def black_scholes_call(S0, K, r, sigma, T):
    d1 = (math.log(S0 / K) + (r + 0.5 * sigma ** 2) * T) / (sigma * math.sqrt(T))
    d2 = d1 - sigma * math.sqrt(T)
    Nd1 = norm.cdf(d1)
    Nd2 = norm.cdf(d2)
    call_price = S0 * Nd1 - K * math.exp(-r * T) * Nd2
    return call_price

# European put option on a non-dividend-paying stock
def black_scholes_put(S0, K, r, sigma, T):
    call_price = black_scholes_call(S0, K, r, sigma, T)
    put_price = call_price - S0 + math.exp(-r * T) * K
    return put_price

def binomial_tree_call(S0, K, r, sigma, T, N):
    delta_t = T / N
    u = math.exp(sigma * math.sqrt(delta_t))
    d = 1 / u
    p = (math.exp(r * delta_t) - d) / (u - d)

    # initialize stock price array
    stock_price = [0] * (N + 1)
    stock_price[0] = S0 * d ** N

    # calculate stock prices at each node of the tree
    for i in range(1, N + 1):
        stock_price[i] = stock_price[i - 1] * u / d

    # initialize option value array at expiration
    option_value = [0] * (N + 1)
    for i in range(N + 1):
        option_value[i] = max(stock_price[i] - K, 0)

    # calculate option value at each node of the tree
    for j in range(N - 1, -1, -1):
        for i in range(j + 1):
            option_value[i] = math.exp(-r * delta_t) * (p * option_value[i + 1] + (1 - p) *
option_value[i])

    return option_value[0]

def binomial_tree_put(S0, K, r, sigma, T, N):
    dt = T / N
    u = math.exp(sigma * math.sqrt(dt))
```

```python
    d = 1 / u
    p = (math.exp(r * dt) - d) / (u - d)
    stock_price = [[0 for j in range(i+1)] for i in range(N+1)]
    option_price = [[0 for j in range(i+1)] for i in range(N+1)]
    for j in range(N+1):
        stock_price[N][j] = S0 * (u ** (N-j)) * (d ** j)
        option_price[N][j] = max(K - stock_price[N][j], 0)
    for i in range(N-1, -1, -1):
        for j in range(i+1):
            stock_price[i][j] = S0 * (u ** (i-j)) * (d ** j)
            option_price[i][j] = math.exp(-r*dt) * (p * option_price[i+1][j] + (1-p) *
option_price[i+1][j+1])
    return option_price[0][0]

def main():
    # define option parameters for the question
    K = 10
    r = 0.02
    sigma = 0.25
    T = 0.25
    S0 = 10

    #
    print("Test results:")
    # b. calculate Black-Scholes price
    bs_price = black_scholes_call(S0, K, r, sigma, T)
    print(f"Black-Scholes price: {bs_price}")
    # a. calculate binomial tree prices for varying numbers of nodes
    n_list = [10, 100, 1000, 10000]
    bt_prices = []
    errors = []
    for n in n_list:
        bt_price = binomial_tree_call(S0, K, r, sigma, T, n)
        bt_prices.append(bt_price)
        # c. calculate error
        error = abs(bt_price - bs_price)
        errors.append(error)
        print(f"Binomial tree price with {n} nodes: {bt_price}, error: {error}")

    # Test results:
    # Black-Scholes price: 0.5224453276436325
    # Binomial tree price with 10 nodes: 0.5101818737713509, error: 0.012263453872281 56
    # Binomial tree price with 100 nodes: 0.5212031399064168, error: 0.0012421877372157386
    # Binomial tree price with 1000 nodes: 0.5223209670289733, error: 0.00012436061465914694
    # Binomial tree price with 10000 nodes: 0.5224328901856781, error: 1.243745795442841e-05

    # The results show that the error decreases as the number of nodes increases.

    # d. calculate ln |E| and ln N
    ln_E = np.log(errors)
    ln_N = np.log(n_list)
    # perform linear regression
    A, B = np.polyfit(ln_N, ln_E, 1)
    # plot ln |E| vs ln N
    plt.scatter(ln_N, ln_E)
    plt.plot(ln_N, A * ln_N + B, color='r')
    plt.xlabel('ln N')
    plt.ylabel('ln |E|')
    plt.title('Convergence of Binomial Tree Method')
    plt.show()

    # e. Using the regression formula found in part (d), |E| = eB(N^-A), we can see that the
    # apparent convergence rate of the Binomial Tree method is determined by the regression
coefficient A.
    # Specifically, the convergence rate is -A. We obtain a regression equation of ln |E| = -
1.0057 ln N + 1.0637.
    # By fitting the least squares line to the ln |E| vs ln N data points, we can obtain the
```

```
values of A and B.
    # Based on these values, we can see that A is approximately -0.5.
    # Therefore, the convergence rate of the Binomial Tree method is approximately 0.5, or one
half.
    # This means that doubling the number of timesteps should roughly halve the error of the
method.

if __name__ == "__main__":
    main()
```

Output:

```
Black-Scholes_vs_Binomial-tree
C:\Users\Jack\anaconda3\python.exe "C:/Users/Jack/Documents/Penn State/MATH 448/Math 44
Test results:
Black-Scholes price: 0.5224453276436325
Binomial tree price with 10 nodes: 0.5101818737713509, error: 0.01226345387228156
Binomial tree price with 100 nodes: 0.5212031399064168, error: 0.0012421877372157386
Binomial tree price with 1000 nodes: 0.5223209670289733, error: 0.00012436061465914694
Binomial tree price with 10000 nodes: 0.5224328901856781, error: 1.243745795442841e-05
```




| Number of Timesteps | Binomial Tree Solution | $\|E\|$ |
|---|---|---|
| N=10 | 0.5101818737713509 | 0.01226345387228156 |
| N=100 | 0.5212031399064168 | 0.0012421877372157386 |
| N=1.000 | 0.5223209670289733 | 0.00012436061465914694 |
| N=10.000 | 0.5224328901856781 | 1.243745795442841 * 10^-05 |