Pricing European and American Options Using the Binomial and Black-Scholes Model

Thomas Tsiftilis – 8/15/2024

This paper investigates the pricing of European and American options using the CRR binomial model and the Black-Scholes model. We explore the convergence of the binomial model into the Black-Scholes model for pricing European options. We also analyze the accuracy and the early exercise boundaries for American options. Additionally, we see the impact of dividend yields on option prices and the early exercise boundaries.

CRR Binomial Model

The parameters for the 'Binomial' function include:

Option Type: C for call options and P for put options

Strike Price: K

Time to maturity: T (years)

Initial Stock Price: S0

Volatility of Stock: σ

Risk-free-rate: r

Dividend: q

# of steps in model: N

Exercise Type: A for American and E for European

One of the key steps to implement in the CRR binomial model includes determining the time step size (Δt), which sets the length of each time step in the binomial tree. A smaller Δt with more time steps generally increases the model's accuracy. The up-state and down-state factors, derived from the volatility (σ) and time step size, represent the potential upward and downward movements of the stock price for each step. The risk-neutral probability ensures that the expected return of the stock under the risk-neutral measure equals the risk-free rate, adjusted for continuous dividend yield (q) when applicable.

Constructing the stock price calculation builds the binomial tree by calculating the stock prices at each node, starting from maturity. Option price initialization sets the option prices at maturity based on the payoff function for calls or puts. The backward induction step calculates the option prices at each node by discounting the expected future values. For American options, this step also checks for early exercise at each node. Finally, measuring computational time helps evaluate the efficiency of the algorithm by tracking the time taken to compute the option price.

We test to see the functionality by giving it a set of parameters and allowing the function to print out the "Option Price" and "Computation Time", shown in the code and output.

```
Option Price: 9.188224825024529
Computation Time: 0.0020270347595214844 seconds
```

Black-Scholes Model

The parameters for the 'Black-Scholes' function include:

Stock Price: S

Strike Price: K

Time to maturity: T (years)

Risk-free-rate: r

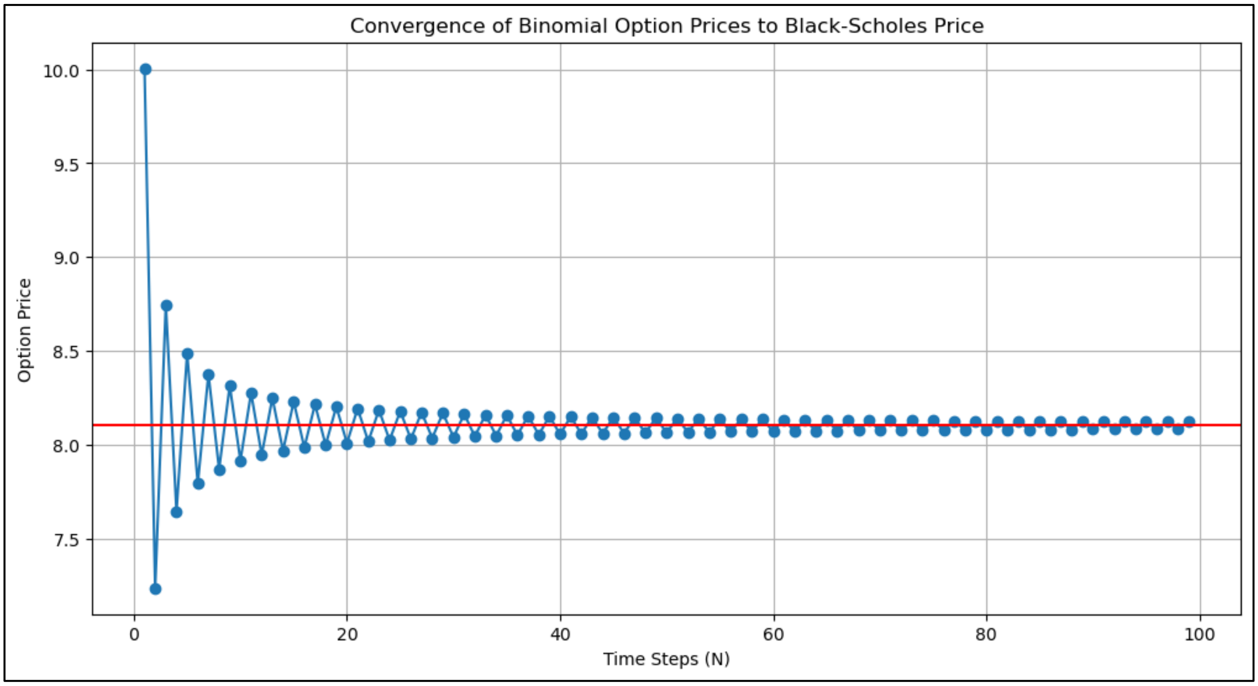
Dividend: q

Volatility of Stock: σ

Option Type: C for call options and P for put options

To demonstrate the convergence of the binomial model to the Black-Scholes option price, we consider a 1-year European call option with a strike price K = 100. The current stock price is also 100. Other parameters include a risk-free interest rate r = 0.05, a continuous dividend yield q = 0.04, and a volatility σ = 0.2.

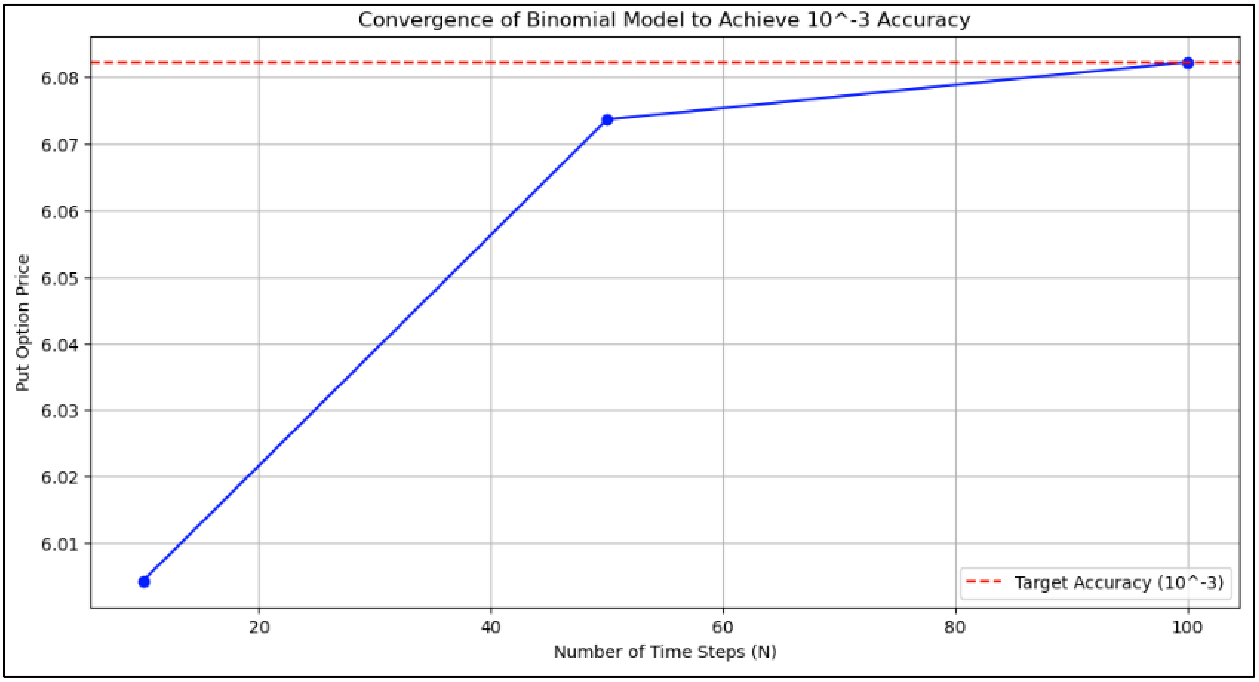
First, we use the Black-Scholes formula to compute the theoretical price of the call option. Next, we implement the binomial model and calculate the option prices for a sequence of increasing numbers of time steps N. We then verify that the binomial option prices converge to the Black-Scholes option price as N increases. Finally, we construct a plot to visualize this convergence.



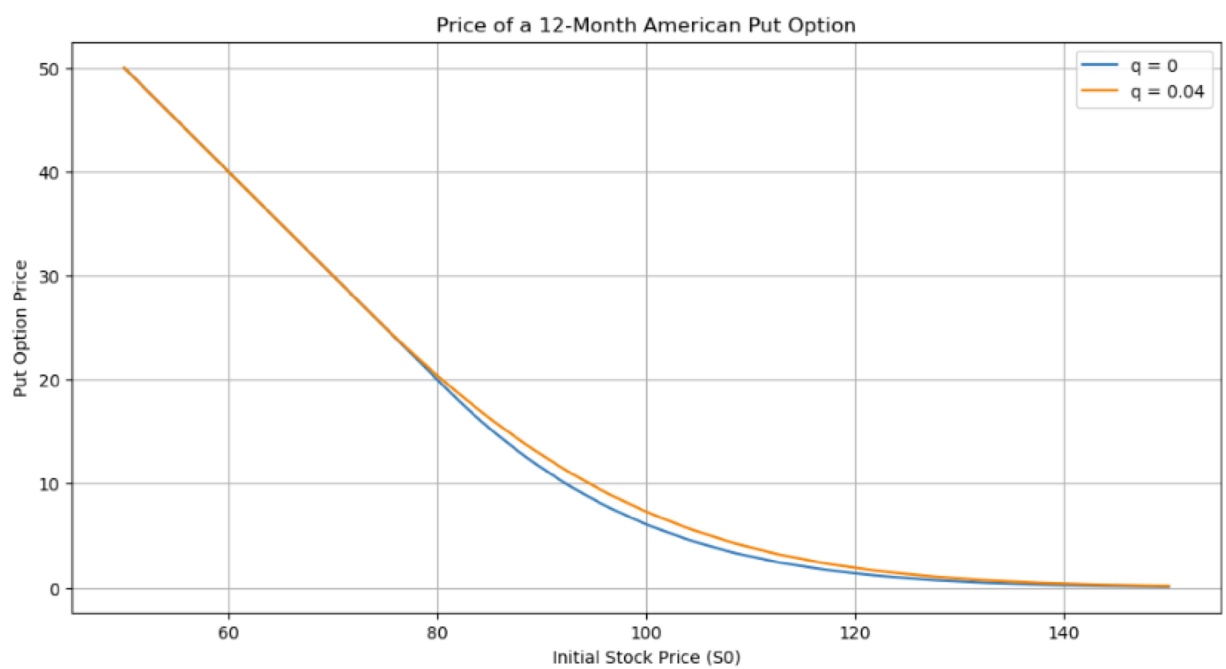
To address the problem of pricing American put options with various times to maturity and evaluating the effect of different dividend yields, we break down the task into several key steps. This involves implementing the binomial model to achieve a specified accuracy, calculating the option price for a range of stock prices, and determining the early exercise boundary.

First, we implement the binomial model for American put options with parameters K = 100, σ = 0.2, r = 0.05, S0 = 100, and a continuous dividend yield q. The time step size is adjusted to ensure the model achieves an accuracy of 10^−3. The up-state and down-state factors are calculated based on the volatility and time step size. Using the risk-neutral probability, we construct the binomial tree for the stock prices and initialize the option prices at maturity based on the payoff function.

Next, we verify the convergence of the binomial option prices by continuously decreasing the time step size and observing the option price. This ensures the accuracy is within 10^-3. We find an approximate and appropriate value for N to be 100.



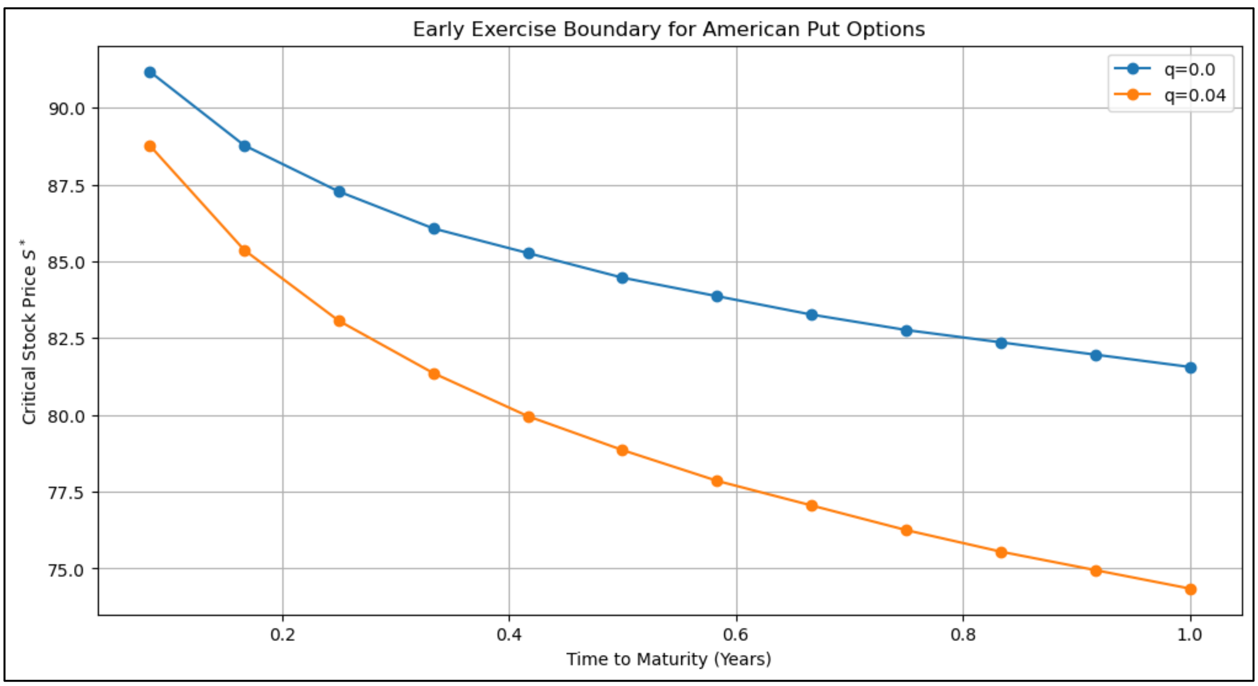
We then calculate and plot the price of a 12-month put option as a function of the initial stock price S0. Using the binomial model, we calculate the option price for a range of S0 values and plot the results to visualize the relationship.



We determine the early exercise boundary by identifying the critical stock price S*(i) for each time to maturity from 1 month to 12 months. This is done by comparing the option price with the intrinsic value and finding the largest stock price where the difference between the option price and intrinsic value is within the error tolerance level $\epsilon = 0.005$. We report and plot the early exercise boundary S*(i) against the time to maturity.

To analyze the effect of dividend yield, we repeat the analysis for a dividend yield of q = 0.04. We calculate the option prices and determine the early exercise boundaries for the new dividend yield, observing the changes in put prices and the early exercise boundary. Generally, a dividend will lower the early exercise compared to a no dividend scenario. This is due to dividends reducing the stock's future value which can increase the intrinsic value of the put option.

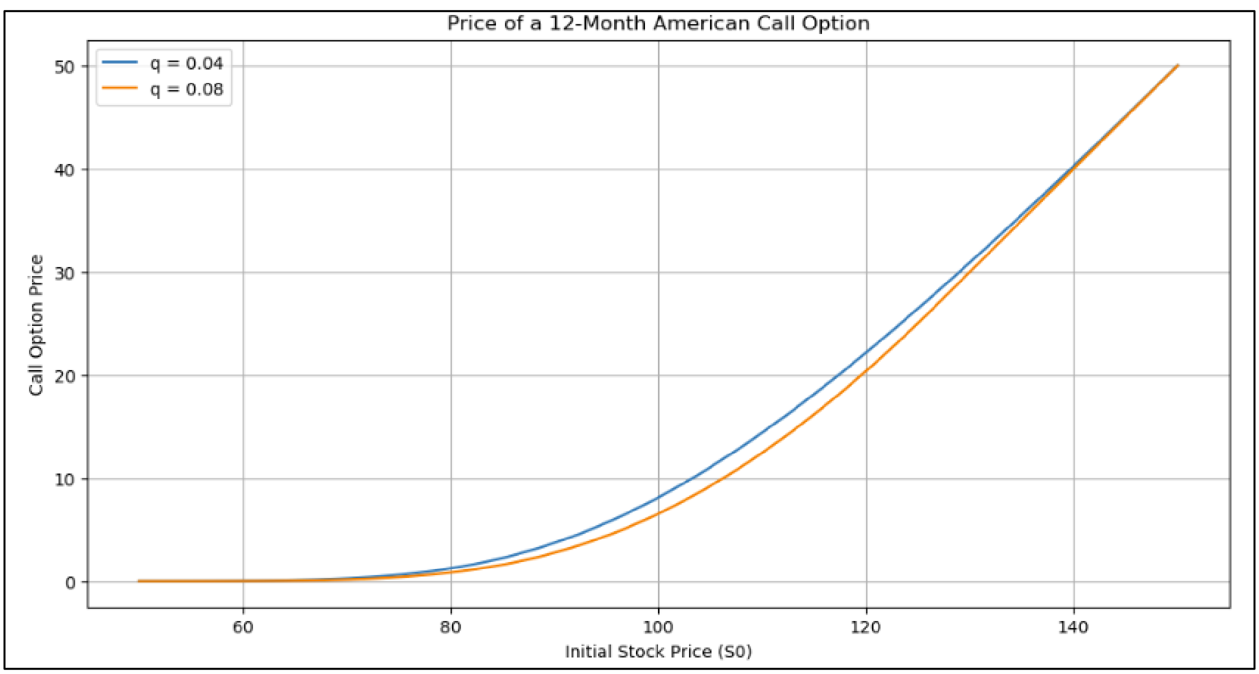
Finally, to ensure we achieve the required accuracy for American options, we continuously decrease the time step size, observe the convergence, and generate a benchmark price using a sufficiently small time step size. This helps verify that the accuracy is within the specified 10^-3.



	q=0.0	q=0.04
0.083333	91.182365	88.777555
0.166667	88.777555	85.370741
0.250000	87.274549	83.066132
0.333333	86.072144	81.362725
0.416667	85.270541	79.959920
0.500000	84.468938	78.857715
0.583333	83.867735	77.855711
0.666667	83.266533	77.054108
0.750000	82.765531	76.252505
0.833333	82.364729	75.551102
0.916667	81.963928	74.949900
1.000000	81.563126	74.348697

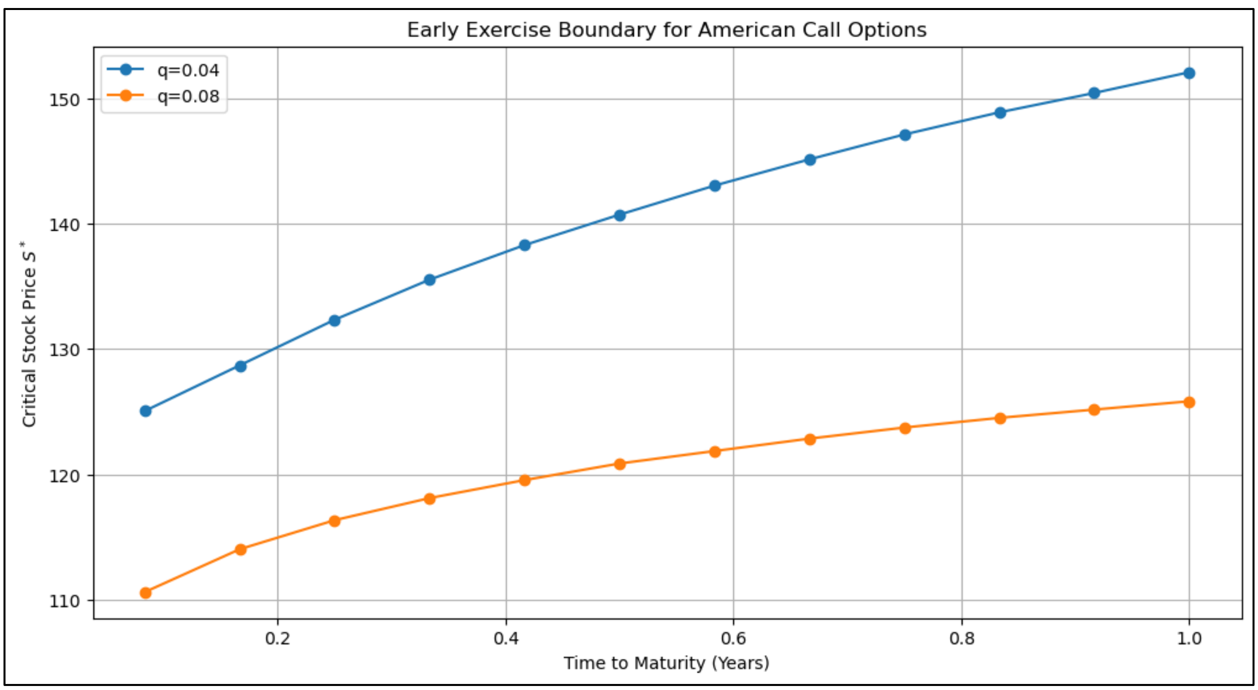
By following these steps, we comprehensively evaluate the pricing of American put options and understand the impact of dividend yields on option prices and early exercise boundaries. The implementation and analysis provide a clear methodology to study the behavior of American options under different market conditions.

To address the pricing of American call options with the parameters K = 100, σ = 0.2, r = 0.05, and q = 0.04, we investigate the required time step size to achieve a 10^-3 accuracy. We adjust the time steps and calculate the option price for each step size to ensure the model's precision. By decreasing the time step size, we verify convergence to the desired accuracy and plot the price of a 12-month call option as a function of the initial stock price S0.



We also determine the critical stock price S*(i) on the early exercise boundary for various maturities from 1 month to 12 months. This involves comparing the option price with the intrinsic value and identifying the stock prices where the difference falls within an error tolerance $\epsilon = 0.005$. The results are plotted to visualize how the early exercise boundary changes with time to maturity.

To analyze the effect of a higher dividend yield, we repeat the process for q = 0.08. We compare how the call prices and early exercise boundaries shift with the increased dividend yield. The intuition behind this is also like that of the put option in early exercise but inversed. As the continuous dividend yield increases from 0.04 to 0.08, the call option prices typically decrease. This is because higher dividends reduce the stock price's expected growth, which lowers the value of the call option. Dividends make holding the stock less attractive compared to the call option, reducing the call's value.



	q=0.04	q=0.08
0.083333	125.060120	110.621242
0.166667	128.697395	114.038076
0.250000	132.334669	116.352705
0.333333	135.531062	118.116232
0.416667	138.286573	119.549098
0.500000	140.711423	120.871743
0.583333	143.026052	121.863727
0.666667	145.120240	122.855711
0.750000	147.104208	123.737475
0.833333	148.867735	124.509018
0.916667	150.410822	125.170341
1.000000	152.064128	125.831663

References

Black, F., & Scholes, M. (1973). The Pricing of Options and Corporate Liabilities. *Journal of Political Economy*, 81(3), 637-654. doi:10.1086/260062

Cox, J. C., Ross, S. A., & Rubinstein, M. (1979). Option Pricing: A Simplified Approach. *Journal of Financial Economics*, 7(3), 229-263. doi:10.1016/0304-405X(79)90015-1

Hull, J. (2017). *Options, Futures, and Other Derivatives* (10th ed.). Pearson.

Merton, R. C. (1973). Theory of Rational Option Pricing. *The Bell Journal of Economics and Management Science*, 4(1), 141-183. doi:10.2307/3003143

Code is written below but can also be found in the zip file.

```
import pandas as pd

import numpy as np

import time

from scipy.stats import norm

import matplotlib.pyplot as plt


# Part 1, CRR Binomial Model (European and American puts and calls on stock with continious dividend yield)

def Binomial(Option, K, T, S0, sigma, r, q, N, Exercise):


    # Starts timer for computation

    start = time.time()
```

```
    # Total time for each step
    delta_time = T / N

    # Up & Down State
    up_state = np.exp(sigma * (np.sqrt(delta_time)))
    down_state = np.exp(-sigma * (np.sqrt(delta_time)))

    # P-star
    p = (np.exp((r - q) * delta_time) - down_state) / (up_state - down_state)

    # Creating a vector for stock prices starting from maturity
    stock_price = np.zeros(N + 1)

    # Calculates stock price at node (N, J)
    for j in range(N + 1):
        stock_price[j] = (up_state ** j) * (down_state ** (N - j)) * S0

    # Option Value at maturity
    if Option == 'C':
        option_price = np.maximum(0, stock_price - K)
    elif Option == 'P':
        option_price = np.maximum(0, K - stock_price)

    # Backward Induction
    for i in range(N - 1, -1, -1):
        for j in range(i + 1):
            stock_price[j] = S0 * (up_state ** j) * (down_state ** (i - j)) # Updates stock prices
            option_price[j] = np.exp(-r * delta_time) * (p * option_price[j + 1] + (1 - p) * option_price[j]) #
Discounts option prices
```

```
        # Early exercise for American Options
        if Exercise == 'A':
            if Option == 'C':
                option_price[j] = np.maximum(option_price[j], stock_price[j] - K)
            elif Option == 'P':
                option_price[j] = np.maximum(option_price[j], K - stock_price[j])

    # Calculate total computation time
    end = time.time()
    computation_time = end - start

    return option_price[0], computation_time

# Test the function
Option = 'C'
K = 100
T = 1
S0 = 100
sigma = 0.2
r = 0.05
q = 0.02
N = 100
Exercise = 'E'

option_price, computation_time = Binomial(Option, K, T, S0, sigma, r, q, N, Exercise)

print(f"Option Price: {option_price}")
print(f"Computation Time: {computation_time} seconds")
```

```
def black_scholes(S, K, T, r, q, sigma, Option):
    d1 = (np.log(S / K) + (r - q + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)

    if Option == 'C':
        option_price = S * np.exp(-q * T) * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2)
    elif Option == 'P':
        option_price = K * np.exp(-r * T) * norm.cdf(-d2) - S * np.exp(-q * T) * norm.cdf(-d1)

    return option_price

# Parameters for the European call option
S = 100
K = 100
T = 1  # 1 year
r = 0.05
q = 0.04
sigma = 0.2
Option = 'C'  # Call option
Exercise = 'E'

black_scholes_price = black_scholes(S, K, T, r, q, sigma, Option)
print(f"Black-Scholes Price: {black_scholes_price}")

N_values = range(1,100)
binomial_prices = [Binomial(Option, K, T, S, sigma, r, q, N, Exercise)[0] for N in N_values]

plt.figure(figsize=(12, 6))
```

```
plt.plot(N_values, binomial_prices, marker='o', label='Binomial Prices')
plt.axhline(y=black_scholes_price, linestyle='-', color = 'red', label='Black-Scholes Price')
plt.xlabel('Time Steps (N)')
plt.ylabel('Option Price')
plt.title('Convergence of Binomial Option Prices to Black-Scholes Price')
plt.grid(True)
plt.show()

accuracies = []
time_steps = [10, 50, 100]
target_accuracy = 1e-3

for N in time_steps:
  put_price, _ = Binomial('P', K, 1, 100, sigma, r, 0, N, 'A')
  accuracies.append(put_price)

plt.figure(figsize=(12, 6))
plt.plot(time_steps, accuracies, marker='o', linestyle='-', color='b')
plt.axhline(y=accuracies[-1], color='r', linestyle='--', label='Target Accuracy (10^-3)')
plt.xlabel('Number of Time Steps (N)')
plt.ylabel('Put Option Price')
plt.title('Convergence of Binomial Model to Achieve 10^-3 Accuracy')
plt.grid(True)
plt.legend()
plt.show()

# Define parameters
S_values = np.linspace(50, 150, 100)
q_puts = [0.0, 0.04]
```

```
put_prices = {q: [] for q in q_puts}
for q in q_puts:
    for S0 in S_values:
        put_price, _ = Binomial("P", 100, 1, S0, 0.2, 0.05, q, 200, "A")
        put_prices[q].append(put_price)

plt.figure(figsize=(12, 6))
for q in q_puts:
    plt.plot(S_values, put_prices[q], label=f'q = {q}')
plt.xlabel('Initial Stock Price $S_0$')
plt.ylabel('Put Option Price')
plt.title('Price of a 12-Month American Put Option')
plt.legend()
plt.grid(True)
plt.show()

def early_puts(N, K, sigma, r, q, S, T):
    s_star = []
    for j in range(len(T)):
        for i in reversed(range(len(S))):
            intrinsic_value_n = max((K - S[i]), 0)
            p_n, _ = Binomial("P", K, T[j], S[i], sigma, r, q, N, 'A')
            difn = abs(intrinsic_value_n - p_n)
            if difn < 0.005:
                st = i
                for k in reversed(range(st)):
                    p, _ = Binomial("P", K, T[j], S[k], sigma, r, q, N, 'A')
                    intrinsic_value = max((K - S[k]), 0)
                    dif = abs(intrinsic_value - p)
```

```
                if dif < 0.005:
                    s_star.append(S[k])
                    break
            break

    return s_star

# Parameters
K = 100
sigma = 0.2
r = 0.05
N = 100  # Number of time steps
S = np.linspace(50, 100, 500)  # Stock price range
T = np.linspace(1/12, 1, 12)  # Maturities from 1/12 to 1 year
q_values = [0.0, 0.04]  # Dividend yield values

results = pd.DataFrame()

for q in q_values:
    s_star = early_puts(N, K, sigma, r, q, S, T)
    results[f'q={q}'] = s_star

print(results)

plt.figure(figsize=(12, 6))
for q in q_values:
    plt.plot(T, results[f'q={q}'], marker='o', linestyle='-', label=f'q={q}')
plt.xlabel('Time to Maturity (Years)')
plt.ylabel('Critical Stock Price $S^*$')
```

```
plt.title('Early Exercise Boundary for American Put Options')
plt.legend()
plt.grid(True)
plt.show()

# Define parameters
S_values = np.linspace(50, 150, 100)  # Example range for S0
q_calls = [0.04, 0.08]  # Different dividend yields
call_prices = {q: [] for q in q_calls}

for q in q_calls:
    for S0 in S_values:
        call_price, _ = Binomial("C", 100, 1, S0, 0.2, 0.05, q, 200, "A")
        call_prices[q].append(call_price)
    plt.plot(S_values, call_prices[q], label=f'q = {q}')

# Plotting the results
plt.xlabel('Initial Stock Price $S_0$')
plt.ylabel('Call Option Price')
plt.title('Price of a 12-Month American Call Option')
plt.legend()
plt.grid(True)
plt.show()

def count_call_s(q, N, S, T, K=100, sigma=0.2, r=0.05, tol=0.005):
    s_star = []

    for j in range(len(T)):
        for i in range(len(S)):
```

```
        intrinsic_value_tmp = max((S[i] - K), 0)

        p_n, _ = Binomial("C", K, T[j], S[i], sigma, r, q, N, 'A')

        if isinstance(p_n, (list, np.ndarray)):

          p_n = np.array(p_n).item()

        difn = abs(intrinsic_value_tmp - p_n)

        if difn < tol:

          st = i

          for k in range(st, len(S)):

            p, _ = Binomial("C", K, T[j], S[k], sigma, r, q, N, 'A')

            if isinstance(p, (list, np.ndarray)):

              p = np.array(p).item()  # Convert to scalar

            intrinsic_value = max((S[k] - K), 0)

            dif = abs(intrinsic_value - p)

            if dif < tol:

              s_star.append(S[k])

              break

          break


    return s_star


K = 100

sigma = 0.2

r = 0.05

N = 100  # Number of time steps

S = np.linspace(105, 160, 500)  # Stock price range adjusted for calls

T = np.linspace(1/12, 1, 12)  # Maturities from 1/12 to 1 year

q_values = [0.04, 0.08]  # Dividend yield values


results = pd.DataFrame(index=T)
```

```
for q in q_values:
    s_star = count_call_s(q, N, S, T)
    results[f'q={q}'] = s_star


print(results)


plt.figure(figsize=(12, 6))
for q in q_values:
    plt.plot(results.index, results[f'q={q}'], marker='o', linestyle='-', label=f'q={q}')
plt.xlabel('Time to Maturity (Years)')
plt.ylabel('Critical Stock Price $S^*$')
plt.title('Early Exercise Boundary for American Call Options')
plt.legend()
plt.grid(True)
plt.show()
```