Comparison of Binomial Option Pricing Model and Black-Scholes Model

Finsearch Team E4 (22B3307 | 22B3312 | 22B3315 | 22B2494)

July 13, 2024

1 Introduction

The Binomial Option Pricing Model and the Black-Scholes Model are two popular Options pricing models. This report compares these models, detailing their mechanics, advantages, disadvantages, and effectiveness.

2 Binomial Option Pricing Model

2.1 Working

The Binomial Option Pricing Model discretizes the time to maturity of an option into a number of steps. At each step, the underlying asset's price can move up or down by a specific factor. The model works by calculating the option's value at each step, starting from maturity and moving backwards to the present.

The model assumes:

- The asset price can move up by a factor u or down by a factor d.where u & d is variance which can be calculate using historical data.

```
#Function to calculate Volatility using historical Data
def get_stock_data(ticker):
    stock = yf.Ticker(ticker)
    hist = stock.history(period="1y")
    current_price = stock.history(period="1d")['Close'][0]
    returns = hist['Close'].pct_change().dropna()
    volatility = returns.std() * np.sqrt(252)  # Annualize the volatility

    return current_price, volatility

# Fetch stock data
S, sigma = get_stock_data(ticker) #here 'S' is Current stock price & sigma is volatility
```

- The risk-neutral probability p of the up move is calculated as:

$$p = \frac{e^{r\Delta t} - d}{u - d}$$

  where r is the risk-free interest rate and Δt is the time step.

- Getting Risk free rate using Fred API

```
fred = Fred(api_key = '1b3896dbdbd6878090ae00d80913a34b')
ten_year_treasury_rate = fred.get_series_latest_release('GS10')/100
r = ten_year_treasury_rate.iloc[-1] #riskfreerate
```

- Calculating Risk-neutral probability

```
dt = T / N
    u = np.exp(sigma * np.sqrt(dt))  # Up factor
    d = 1 / u                        # Down factor
    p = (np.exp(r * dt) - d) / (u - d)  # Risk-neutral probability
    #Here u,d,p, t(time to maturity), N(no of steps in bionomil model), sigma(volatility)
```

- Calculating Asset price at Maturity

```
 asset_prices = np.zeros(N + 1)
    for i in range(N + 1):
        asset_prices[i] = S * (u ** i) * (d ** (N - i))
```

- Calculating Option price at Maturity

```
option_values = np.maximum(0, asset_prices - K) if option_type == 'call' else np.maximum(0, K - asset_prices)
    for i in range(N - 1, -1, -1):
        for j in range(i + 1):
            option_values[j] = np.exp(-r * dt) * (p * option_values[j + 1] + (1 - p) * option_values[j])
```

3 Google Call Example Using Binomial Option model

Inputs:

- Stock Ticker: GOOGL
- Strike Price: $180
- Time to Maturity: 1 year
- Number of step in binomial model: 1000
- Option Type: Call

Output:

- Call Option Price: $26.49

4 Google Reduced step Example Using Binomial Option model

Inputs:

- Stock Ticker: GOOGL
- Strike Price: $180
- Time to Maturity: 1 year
- Number of step in binomial model: 100
- Option Type: Call

Output:

- Call Option Price: $26.53

5 Microsoft Call Example Using Binomial Option model

Inputs:

- Stock Ticker: MSFT
- Strike Price: $450
- Time to Maturity: 1 year
- Number of step in binomial model: 1000
- Option Type: Call

Output:

- Call Option Price: $48.10

6 Microsoft Put Example Using Binomial Option model

Inputs:

- Stock Ticker:MSFT
- Strike Price: $450
- Time to Maturity: 1 year
- Number of step in binomial model: 1000
- Option Type: Put

Output:

- Put Option Price: $25.56

7 Black-Scholes Model

7.1 Working

The Black-Scholes Model provides a closed-form solution for European call and put options. It assumes that the price of the underlying asset follows a geometric Brownian motion with constant volatility and interest rates.

The option price is given by:

$$C = S_0 N(d_1) - K e^{-rT} N(d_2)$$

- Calculating Options price

```
    if option_type == 'call':
        option_price = S * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2)
    elif option_type == 'put':
        option_price = K * np.exp(-r * T) * norm.cdf(-d2) - S * norm.cdf(-d1)
```

where:

$$d_1 = \frac{\ln(S_0/K) + (r + \sigma^2/2)T}{\sigma\sqrt{T}}$$

$$d_2 = d_1 - \sigma\sqrt{T}$$

- Calculating d1 and d2

```
    d1 = (np.log(S / K) + (r + 0.5 * sigma ** 2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
```

8 Google Call Example Using Black-scholes Model

Inputs:

– Stock Ticker: GOOGL
– Strike Price: \$180
– Time to Maturity: 1 year
– Option Type: Call

Output:

– Call Option Price: \$26.49

9 Google Put Example Using Black-scholes Model

Inputs:

– Stock Ticker: GOOGL
– Strike Price: \$180
– Time to Maturity: 1 year
– Option Type: Call

Output:

– Call Option Price: \$13.82

10 Microsoft Call Example Using Black-scholes Model

Inputs:

– Stock Ticker: MSFT
– Strike Price: \$450
– Time to Maturity: 1 year
– Option Type: Call

Output:

– Call Option Price: \$48.10

11 Microsoft Put Example Using Black-scholes Model

Inputs:

- Stock Ticker: MSFT
- Strike Price: $450
- Time to Maturity: 1 year
- Option Type: Call

Output:

- Call Option Price: $25.56

12 Conclusion

Hence If the binomial model uses a high number of steps (e.g., 1000), the prices from both models converge, demonstrating their equivalence. However, with fewer steps, there may be discrepancies due to the binomial model's approximation.

Advantages and Disadvantages Of Both Models :-

The big advantage the binomial model has over the Black-Scholes model is that it can be used to accurately price American options. This is because with the binomial model it's possible to check at every step of the binomial tree for the possibility of early exercise (eg where, due to eg a dividend, or a put being deeply in the money the option price at that point is less than its intrinsic value).

The main advantage of the Black-Scholes model is speed – it lets you calculate a very large number of option prices in a very short time, Whereas the binomial model is relatively slow speed. It's great for half a dozen calculations at a time but even with today's fastest PCs it's not a practical solution for the calculation of thousands of prices in a few seconds.

Relation Between Both Models :-

The same underlying assumptions regarding stock prices underpin both the binomial and Black-Scholes models: that stock prices follow a stochastic process described by geometric brownian motion. As a result, for European options, the binomial model converges on the Black-Scholes formula as the number of binomial calculation steps increases. In fact the Black-Scholes model for European options is really a special case of the binomial model where the number of binomial steps is infinite. In other words, the binomial model provides discrete approximations to the continuous process underlying the Black-Scholes model

13 Binomial Model Code

```python
import numpy as np
import yfinance as yf
from datetime import datetime, timedelta
from fredapi import Fred

#Getting inputs from user
ticker = input("Enter the stock ticker: ").strip().upper()
#print(get_stock_data(ticker))
K = float(input("Enter the strike price: "))
T = float(input("Enter time to maturity (in years): "))
N = int(input("Enter the number of steps in the binomial model: "))
option_type = input("Enter the option type ('call' or 'put'): ").strip().lower()

# Fetch stock data
S, sigma = get_stock_data(ticker)

#Getting risk_free_rate using Fred_Api
fred = Fred(api_key = '1b3896dbdbd6878090ae00d80913a34b')
ten_year_treasury_rate = fred.get_series_latest_release('GS10')/100
r = ten_year_treasury_rate.iloc[-1] #riskfreerate

def binomial_option_pricing(S, K, T, r, sigma, N, option_type='call'):
    dt = T / N
    u = np.exp(sigma * np.sqrt(dt))  # Up factor
    d = 1 / u  # Down factor
    p = (np.exp(r * dt) - d) / (u - d)  # Risk-neutral probability

    # Initialize asset prices at maturity
    asset_prices = np.zeros(N + 1)
    for i in range(N + 1):
        asset_prices[i] = S * (u ** i) * (d ** (N - i))

    # Initialize option values at maturity
    option_values = np.maximum(0, asset_prices - K) if option_type == 'call' else np.maximum(0, K - asset_prices)

    # Step back through the tree
    for i in range(N - 1, -1, -1):
        for j in range(i + 1):
            option_values[j] = np.exp(-r * dt) * (p * option_values[j + 1] + (1 - p) * option_values[j])

    return option_values[0]

def get_stock_data(ticker):
    stock = yf.Ticker(ticker)
    hist = stock.history(period="1y")
    current_price = stock.history(period="1d")['Close'][0]
    returns = hist['Close'].pct_change().dropna()
    volatility = returns.std() * np.sqrt(252)  # Annualize the volatility

    return current_price, volatility

option_price = binomial_option_pricing(S, K, T, r, sigma, N, option_type)
print(f"{option_type.capitalize()} Option Price: {option_price:.2f}")
```

14 Black-Scholes Model Code

```
import yfinance as yf
import numpy as np
from scipy.stats import norm
from fredapi import Fred

# User inputs
ticker = input("Enter the stock ticker: ").strip().upper()
K = float(input("Enter the strike price: "))
T = float(input("Enter time to maturity (in years): "))
#r = float(input("Enter the risk-free interest rate (annual): "))
option_type = input("Enter the option type ('call' or 'put'): ").strip().lower()


#Getting risk_free_rate using Fred_Api
fred = Fred(api_key = '1b3896dbdbd6878090ae00d80913a34b')
ten_year_treasury_rate = fred.get_series_latest_release('GS10')/100
r = ten_year_treasury_rate.iloc[-1] #riskfreerate

def get_stock_data(ticker):
    stock = yf.Ticker(ticker)
    hist = stock.history(period="1y")
    current_price = stock.history(period="1d")['Close'][0]
    returns = hist['Close'].pct_change().dropna()
    volatility = returns.std() * np.sqrt(252)  # Annualize the volatility

    return current_price, volatility

# Fetch stock data
S, sigma = get_stock_data(ticker)


def black_scholes(S, K, T, r, sigma, option_type='call'):
    d1 = (np.log(S / K) + (r + 0.5 * sigma ** 2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)

    if option_type == 'call':
        option_price = S * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2)
    elif option_type == 'put':
        option_price = K * np.exp(-r * T) * norm.cdf(-d2) - S * norm.cdf(-d1)
    else:
        raise ValueError("Invalid option type. Use 'call' or 'put'.")

    return option_price

# Calculate option price
option_price = black_scholes(S, K, T, r, sigma, option_type)
print(f"{option_type.capitalize()} Option Price: {option_price:.2f}")
```