Binomial Tree Model in Python

Thomas Roberts

October 2025

1 Introduction

This report presents a Python implementation of the binomial tree model for pricing European options. The binomial model is a discrete-time framework that approximates the evolution of an asset's price by allowing it to move up or down at each step. By iteratively discounting expected payoffs under a risk-neutral probability, the model provides a computationally simple yet powerful method for valuing derivative securities.

The implementation begins with the one-step model, which serves as the foundation for understanding the approach. It is then extended to two-step and multi-step versions, demonstrating how the method generalises to more accurate approximations of the continuous-time Black–Scholes framework.

2 Methodology

The binomial model assumes that the stock price can move up by a factor u or down by a factor d in each time step Δt. Under the risk-neutral measure, the probability of an upward move is given by:

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

where r is the continuously compounded risk-free rate. The option value at each node is the discounted expected value of its future payoffs:

$$C_i = e^{-r\Delta t}[pC_{i+1}^{(u)} + (1-p)C_{i+1}^{(d)}].$$

This recursive process continues until the initial time step is reached.

3 Python Implementation

3.1 One-Step Model

Listing 1: Python Code for a One-Step European Call Option

```python
import math

def one_step_call_price(S0, E, u, d, r, T):
    """Price a European call option using the one-step binomial model."""
    p = (math.exp(r * T) - d) / (u - d)
    Cu = max(S0 * u - E, 0)
    Cd = max(S0 * d - E, 0)
    C0 = math.exp(-r * T) * (p * Cu + (1 - p) * Cd)
    return {"Cu": round(Cu, 3), "Cd": round(Cd, 3),
            "p": round(p, 5), "C0": round(C0, 3)}

# Example 8.5
S0, E, u, d, r, T = 40, 42, 1.1, 0.9, 0.12, 0.25
print(one_step_call_price(S0, E, u, d, r, T))

#Expected output: {'Cu': 2, 'Cd': 0, 'p': 0.65227, 'C0': 1.266}
```

Output:

```
{'Cu': 2.0, 'Cd': 0, 'p': 0.65227, 'C0': 1.266}
```

3.2 Two-Step Model

Listing 2: Python Code for a Two-Step Binomial Option Model

```python
def two_step_option_price(S0, E, u, d, r, T, option_type="call"):
    """Price a European option using a two-step binomial model."""
    t = T / 2
    p = (math.exp(r * t) - d) / (u - d)

    Suu, Sud, Sdd = S0 * u**2, S0 * u * d, S0 * d**2

    if option_type == "call":
        Cuu, Cud, Cdd = max(Suu - E, 0), max(Sud - E, 0), max(Sdd - E, 0)
    else:
        Cuu, Cud, Cdd = max(E - Suu, 0), max(E - Sud, 0), max(E - Sdd, 0)

    C0 = math.exp(-r * T) * (
        p**2 * Cuu + 2*p*(1-p)*Cud + (1-p)**2 * Cdd
    )
    return {"p": round(p,5), "Cuu": round(Cuu,3),
            "Cud": round(Cud,3), "Cdd": round(Cdd,3),
            "C0": round(C0,3)}

# Example 9.6
S0, E, u, d, r, T = 40, 32, 1.2, 0.8, 0.1, 0.5
print(two_step_option_price(S0, E, u, d, r, T, "put"))

# Expected output: {'p': 0.56329, 'Cuu': 0, 'Cud': 0, 'Cdd': 6.4, 'C0': 1.161}
```

Output:

```
{'p': 0.56329, 'Cuu': 0, 'Cud': 0, 'Cdd': 6.4, 'C0': 1.161}
```

3.3 Multi-Step Model

Listing 3: Python Code for an N-Step Binomial Option Model

```python
def binomial_option_price(S0, E, r, T, u, d, N, option_type="call"):
    """Price a European option using an N-step binomial tree."""
    dt = T / N
    p = (math.exp(r * dt) - d) / (u - d)
    prices = [S0 * (u ** j) * (d ** (N - j)) for j in range(N + 1)]

    if option_type == "call":
        values = [max(price - E, 0) for price in prices]
    else:
        values = [max(E - price, 0) for price in prices]

    for i in range(N - 1, -1, -1):
        values = [
            math.exp(-r * dt) * (p * values[j + 1] + (1 - p) * values[j])
            for j in range(i + 1)
        ]
    return {"p": round(p,5), "option_price": round(values[0],2)}

# Example: Multi-step call option (N=3)
S0, E, r, T, u, d, N = 40, 42, 0.12, 0.25, 1.1, 0.9, 3
print(binomial_option_price(S0, E, r, T, u, d, N, "call"))
```

Output:

```
{'p': 0.55025, 'option_price': 2.44}
```

4 Discussion

The Python implementation reproduces the core mechanics of the binomial tree model for European options. In each version, the risk-neutral probability p governs the weighting between upward and downward price movements, ensuring that expected discounted values are consistent with the risk-free return. As the number of steps N increases, the discrete model converges towards the continuous-time Black–Scholes price.

The one-step and two-step examples yield values that align with analytical solutions from standard references, confirming the correctness of the implementation. The modular structure of the functions allows easy adjustment for different parameters and option types.

5 Conclusion

This work demonstrates the design of a Python-based framework for pricing European options via the binomial tree method. Starting with the single-step case and extending to multi-step implementations, the results illustrate the power of backward induction for option valuation.

The Python results are consistent with theoretical expectations, confirming both accuracy and computational transparency. The modular design also provides a foundation for extending the framework to American options or other derivatives, making it a practical and educational tool for financial analysis.