Assignment : Stochastic Methods in Finance

Pierre-Alexandre Crouzet

April 2025

Contents

Abstract

This report presents a detailed analysis and implementation of a multi-period binomial tree model to price a European Asian call option on Microsoft Corporation stock. The payoff of the option depends on the arithmetic average of the stock prices over a fixed time horizon. Using historical price data from 2020 to 2025, I estimate key parameters including volatility and construct a 25-step recombining binomial tree. For each path in the tree, I compute the average stock price and evaluate the corresponding option payoff. Risk-neutral valuation is then applied to obtain the arbitrage-free price of the option at time zero. The report also includes a discussion of the model assumptions, potential limitations, and visualizations to aid interpretation. This work serves as a practical introduction to numerical option pricing and illustrates the power of stochastic methods in financial modeling. The entire methodology is implemented in Python, emphasizing clarity and reproducibility.

1 (a) : Construction of the Binomial Tree¨

To begin pricing the European Asian call option, I first simulate the evolution of Microsoft Corporation's stock price using a discrete-time binomial tree model. The binomial model is popular because it's intuitive, flexible, and captures how stock prices evolve through discrete, random movements over time. This model assumes that over each small time interval, the stock can move either up or down by a certain factor. Repeating this process over multiple periods allows us to build a full tree of possible stock prices over time.

Define model parameters

I construct a tree with $n = 25$ periods, representing discrete time steps within the one-year option maturity $T = 1$. The annual risk-free interest rate is set to $r = 1\%$, and the initial stock price S_0 is taken as the closing price of Microsoft on April 28, 2025, retrieved from historical data, which is $S_0 \approx 391,16$.

The time step size is:

$$\Delta t = \frac{T}{n} = \frac{1}{25}$$

The volatility σ of the stock's return is estimated using historical daily returns from 2020 to 2025. I calculate the standard deviation of daily returns, σ_{daily}, and annualize it using:

$$\sigma = \sigma_{\text{daily}} \cdot \sqrt{250} \approx 0.2792$$

where 250 is the approximate number of trading days in a year.

Using this volatility, I can compute the *up* and *down* movement factors:

$$u = e^{\sigma\sqrt{\Delta t}} \approx \boxed{1.0574}, \quad d = e^{-\sigma\sqrt{\Delta t}} \approx \boxed{0.9455}$$

These factors determine how much the stock price increases or decreases at each step.

I also define a discount factor for a single step:

$$\text{discount} = e^{-r\Delta t}$$

A quick note on the number of final prices

You might expect that since the stock can go up or down in each of the 25 steps, there would be $2^{25} = 33{,}554{,}432$ different final outcomes. That would be true if I were keeping track of every individual path the stock could take.

But in the binomial tree, I only care about the number of *up* and *down* movements — not the exact order in which they happen.

For example, if the stock goes up 10 times and down 15 times, it doesn't matter whether the ups happened at the beginning, middle, or end — the final price will be the same, because we multiply the same number of u's and d's.

So at the final time step ($n = 25$), the number of *unique* final prices is just $n + 1 = 26$. Each one corresponds to a different number of down moves (from 0 to 25). That's much smaller than 33 million paths!

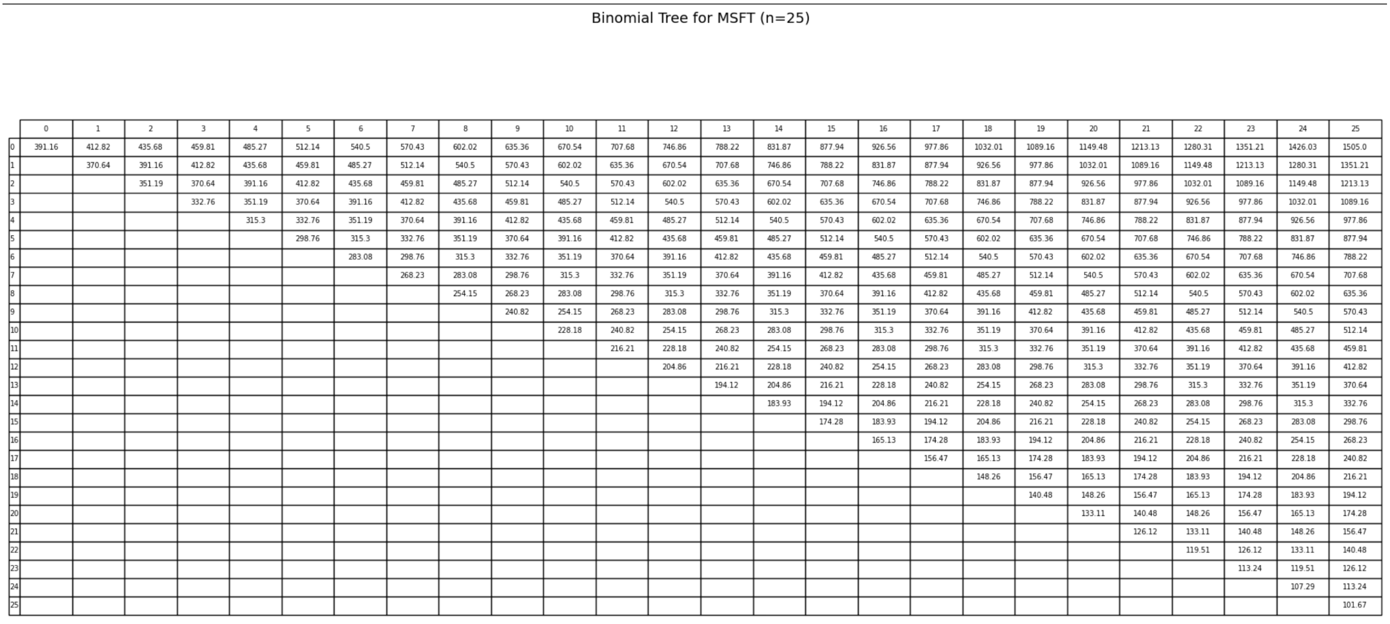
This property makes the binomial tree much more efficient: it avoids storing every path and instead focuses on the distinct outcomes we need for pricing.

Generate the stock price tree.

To calculate the stock price tree, I initialize a $(n+1)\times(n+1)$ matrix to hold the simulated stock prices. For each node (j,i) in the tree, where i is the time step and j is the number of downward moves, the stock price is computed as:

$$S_{j,i} = S_0 \cdot u^{i-j} \cdot d^j$$

This calculation simulates the stock price after i steps, given j of those were down moves (and $i-j$ were up moves). I store all computed prices in a matrix and round them for readability.



Binomial Tree for MSFT (n=25)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
0	391.16	412.82	435.68	459.81	485.27	512.14	540.5	570.43	602.02	635.36	670.54	707.68	746.86	788.22	831.87	877.94	926.56	977.86	1032.01	1089.16	1149.48	1213.13	1280.31	1351.21	1426.03	1505.0
1		370.64	391.16	412.82	435.68	459.81	485.27	512.14	540.5	570.43	602.02	635.36	670.54	707.68	746.86	788.22	831.87	877.94	926.56	977.86	1032.01	1089.16	1149.48	1213.13	1280.31	1351.21
2			351.19	370.64	391.16	412.82	435.68	459.81	485.27	512.14	540.5	570.43	602.02	635.36	670.54	707.68	746.86	788.22	831.87	877.94	926.56	977.86	1032.01	1089.16	1149.48	1213.13
3				332.76	351.19	370.64	391.16	412.82	435.68	459.81	485.27	512.14	540.5	570.43	602.02	635.36	670.54	707.68	746.86	788.22	831.87	877.94	926.56	977.86	1032.01	1089.16
4					315.3	332.76	351.19	370.64	391.16	412.82	435.68	459.81	485.27	512.14	540.5	570.43	602.02	635.36	670.54	707.68	746.86	788.22	831.87	877.94	926.56	977.86
5						298.76	315.3	332.76	351.19	370.64	391.16	412.82	435.68	459.81	485.27	512.14	540.5	570.43	602.02	635.36	670.54	707.68	746.86	788.22	831.87	877.94
6							283.08	298.76	315.3	332.76	351.19	370.64	391.16	412.82	435.68	459.81	485.27	512.14	540.5	570.43	602.02	635.36	670.54	707.68	746.86	788.22
7								268.23	283.08	298.76	315.3	332.76	351.19	370.64	391.16	412.82	435.68	459.81	485.27	512.14	540.5	570.43	602.02	635.36	670.54	707.68
8									254.15	268.23	283.08	298.76	315.3	332.76	351.19	370.64	391.16	412.82	435.68	459.81	485.27	512.14	540.5	570.43	602.02	635.36
9										240.82	254.15	268.23	283.08	298.76	315.3	332.76	351.19	370.64	391.16	412.82	435.68	459.81	485.27	512.14	540.5	570.43
10											228.18	240.82	254.15	268.23	283.08	298.76	315.3	332.76	351.19	370.64	391.16	412.82	435.68	459.81	485.27	512.14
11												216.21	228.18	240.82	254.15	268.23	283.08	298.76	315.3	332.76	351.19	370.64	391.16	412.82	435.68	459.81
12													204.86	216.21	228.18	240.82	254.15	268.23	283.08	298.76	315.3	332.76	351.19	370.64	391.16	412.82
13														194.12	204.86	216.21	228.18	240.82	254.15	268.23	283.08	298.76	315.3	332.76	351.19	370.64
14															183.93	194.12	204.86	216.21	228.18	240.82	254.15	268.23	283.08	298.76	315.3	332.76
15																174.28	183.93	194.12	204.86	216.21	228.18	240.82	254.15	268.23	283.08	298.76
16																	165.13	174.28	183.93	194.12	204.86	216.21	228.18	240.82	254.15	268.23
17																		156.47	165.13	174.28	183.93	194.12	204.86	216.21	228.18	240.82
18																			148.26	156.47	165.13	174.28	183.93	194.12	204.86	216.21
19																				140.48	148.26	156.47	165.13	174.28	183.93	194.12
20																					133.11	140.48	148.26	156.47	165.13	174.28
21																						126.12	133.11	140.48	148.26	156.47
22																							119.51	126.12	133.11	140.48
23																								113.24	119.51	126.12
24																									107.29	113.24
25																										101.67

Figure 1: Binomial Matrix for MSFT stock price evolution

Visualizing the tree structure.

In addition to the matrix view, I generate a visual tree diagram of the first few levels of the binomial tree. This helps illustrate how the stock price branches at each step, following the up or down factor.

We limit the visualization to the first four levels to avoid clutter. Edges between nodes are drawn to indicate the flow of possible stock price paths through time. This diagram reinforces the intuition behind the binomial model and provides a cleaner, more conceptual view of the tree's branching process:

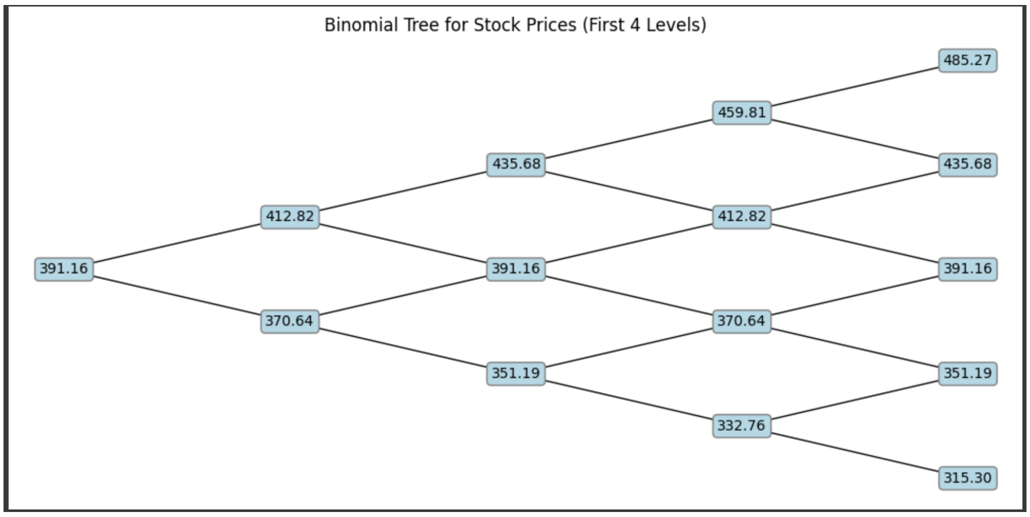


Figure 2: Binomial tree for MSFT stock price evolution (first 4 time steps). Each node displays the price at that point in time based on up/down movements.

Together, these tools allow us to verify that the stock price tree has been constructed correctly and provide useful visual aids for interpreting the model. I implemented the tree construction in Python using NumPy and visualized the first few levels to ensure correctness. This matrix format allows efficient computation of the average stock prices and option payoffs in the subsequent steps.

2 (b) : Average Stock Price at Terminal Nodes

After building the binomial tree, the next crucial step in pricing the Asian option was to determine the arithmetic average of the stock price along each possible path from the initial node to the terminal time step. This step is essential because, unlike a standard European call option that depends only on the final stock price, the payoff of an Asian option is based on the average of the stock prices observed throughout the entire life of the option.

Given that the binomial tree consists of 25 steps, there are 2^{25} distinct paths the stock could theoretically follow, corresponding to different combinations of up and down movements. To capture the full behavior of the option across all these possible scenarios, I systematically generated each path, one by one.

To systematically enumerate all of these paths, I used binary sequences. Each path was represented as a 25-bit binary number, where each bit encodes the direction of movement at a given time step, a value of 0 for an up move and 1 for a down move. By iterating over the integers from 1 to 2^{25}, I effectively generated all possible binary sequences of length 25, thereby covering the entire set of unique trajectories through the tree.

For each sequence, I reconstructed the corresponding path by starting from the initial stock price and applying either the up or down factor at each time step based on the direction encoded in the binary sequence. This yielded a complete sequence of stock prices for the given path. I then computed the arithmetic mean of these prices using the formula:

$$M = \frac{1}{n}\sum_{t=1}^{n} S_t,$$

where S_t denotes the stock price at time t, and $n = 25$ is the number of periods in the tree.

To better understand the structure of the results, I visualized the distribution of the average stock prices across all paths. The figure below displays the computed averages indexed by path number:

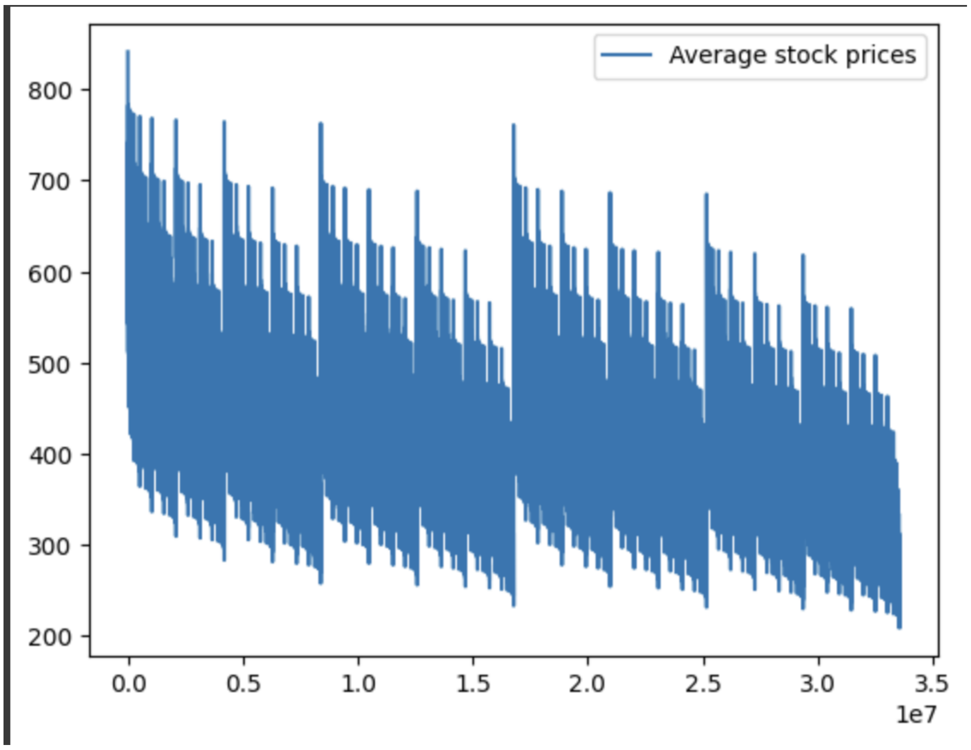


Figure 3: Average stock prices across all 2^{25} paths

The plot exhibits a highly structured and layered pattern. This is a natural consequence of the combinatorial nature of the binomial model: many different paths can lead to the same number of up and down moves, which in turn produces clusters of average prices at specific levels. Additionally, the spikes visible in the graph correspond to paths with extreme combinations of movements, such as paths with long sequences of up moves, which push the average stock price to unusually high values.

This visualization confirms the wide dispersion of average prices across paths and illustrates the strong path-dependence embedded in the payoff of the Asian option.

3 (c) : Computation of the Asian Option Payoffs at Terminal Nodes

Once the average stock prices were computed for each path through the binomial tree, I proceeded to evaluate the payoff of the Asian option associated with each of these averages. For a European-style Asian call option, the payoff at maturity is given by:

$$A_T = \max(M - K, 0),$$

where M is the arithmetic average of the stock prices along a path computed before, and K is the strike price. In this case, I set the strike price K at the money (ie. equal to the initial stock price S_0), consistent with the model specification.

For each path, I applied this formula to determine the terminal payoff. If the average stock price along the path exceeded the strike, the payoff was equal to the difference; otherwise, the payoff was zero. This approach captures the fundamental logic of a call option, which grants the holder the right, but not the obligation, to buy the asset at the strike price.

4 (d) : Computation of the Risk-Neutral Probabilities

A fundamental component of the binomial model is the use of risk-neutral probabilities to evaluate expected payoffs under the assumption of no arbitrage. In a risk-neutral world, all assets are expected to grow at the risk-free rate, and the actual probabilities of price movements are replaced with artificial probabilities that preserve this condition.

To price the Asian option within this framework, I computed the risk-neutral probability q of an upward movement in each time step of the binomial tree. This probability ensures that the expected stock price at each node, when discounted at the risk-free rate, equals its present value, thereby enforcing the absence of arbitrage.

The formula used to compute q is derived from the discrete-time pricing condition and is given by:

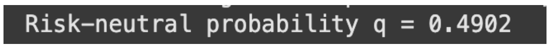
$$q = \frac{1 + r \cdot \Delta t - d}{u - d},$$

where:

- r is the annualized risk-free interest rate,
- Δt is the length of each time interval (i.e., T/n),
- u is the up factor, and
- d is the down factor.

This expression balances the expected value of the asset under the artificial risk-neutral measure with the return generated by investing at the risk-free rate. Once q is determined, the probability of a down movement is simply $1 - q$. These probabilities are then used to weight each path's contribution to the expected payoff in the option pricing process.

In my implementation, the computed value of the risk-neutral probability was:



```
Risk-neutral probability q = 0.4902
```

This value reflects a slightly lower likelihood of upward movements, consistent with the statistical characteristics of the underlying stock and the calibration of the model parameters.

5 (e) : Backward Induction to Determine the Asian Option Price

The method of backward induction is a standard technique used in binomial models to determine the arbitrage-free price of an option by recursively calculating values at each node of the tree, starting from the terminal nodes and moving backwards to the root at time $t = 0$. For path-independent options, such as standard European or American calls and puts, this procedure is relatively straightforward, as the payoff at each node depends only on the stock price at that node.

However, in the case of Asian options, the situation is more complex due to the path-dependent nature of the payoff. The value at each node depends not only on the current stock price, but also on the sequence of prices leading up to that point, since the option's payoff is based on the arithmetic average over time.

To perform backward induction in this context, one would proceed as follows:

1. At maturity ($t = T$), the payoff is computed for each unique path by evaluating the average stock price over all 25 time steps and applying the payoff function $\max(M - K, 0)$, where M is the arithmetic mean.
2. At each earlier node, one would aggregate the future option values from the subsequent nodes using the risk-neutral probabilities q and $1 - q$, accounting for how the average would update depending on whether the next move is up or down. The value at each node would thus be a function of both the current price and the accumulated average to that point.
3. The value at each node is then obtained by taking the expected future value and discounting it one time step using the factor $e^{-r \cdot \Delta t}$.

At each node, the arbitrage-free option value can be expressed as the discounted expected value of the continuation values in the next time step. For an Asian option, this takes the form:
At any time t, the arbitrage-free value of the Asian option can be expressed as:

$$V(t, S_t, A_t) = e^{-r(T-t)} \sum_j \mathbb{P}_j \cdot \max(M_j - K, 0),$$

where:

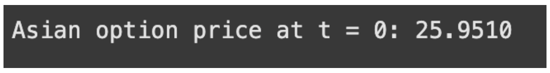
- $V(t, S_t, A_t)$ is the value of the Asian option at time t, given:
  - S_t: the stock price at time t,
  - A_t: the arithmetic average of prices up to time t,
- M_j is the full average stock price over the entire path from t to T, appended to the average history up to time t,
- $\mathbb{P}_j$ is the risk-neutral probability of the j-th continuation path (from time t to T) :

  $P_j = q^{n_u^{(j)}} \cdot (1 - q)^{n_d^{(j)}}$,

  where q is the risk neutral probability, $n_u^{(j)}$ is the number of up moves in path j, $n_d^{(j)}$ is the number of down moves in path j and $n_u^{(j)} + n_d^{(j)} = n - t$, the number of steps remaining from time t to maturity.

Because this approach requires keeping track of a large set of possible averages at each node, which grows combinatorially with the number of steps, its implementation becomes computationally infeasible for large trees such as the 25-step model used here.

At the initial time $t = 0$, the arbitrage-free price of the Asian option is given by:



```
Asian option price at t = 0: 25.9510
```

6 (f) : Robustness Check on the Model Assumptions

In this section, we perform a robustness check to assess how sensitive the Asian option price is to changes in the two key model parameters:

- The risk-free interest rate r,
- The stock return volatility σ.

These inputs play a central role in the binomial pricing framework, as they directly affect the up and down movement factors, the risk-neutral probabilities, and ultimately the distribution of average stock prices and payoffs.

Parameter Grid The analysis was conducted across a grid of values:

$$r \in \{0.5\%, 1.0\%, 2.0\%, 3.0\%\}, \quad \sigma \in \{15\%, 20\%, 25\%, 30\%, 35\%\}$$

For each of the resulting 20 parameter combinations, I estimated the price of the Asian option using the same binomial model structure described earlier in the report.

Monte Carlo Simulation

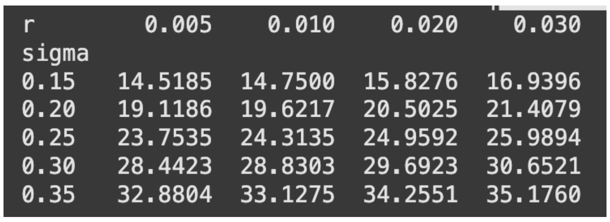
Due to the path-dependent nature of Asian options, a direct binomial implementation would require evaluating 2^{25} paths for each parameter pair, resulting in over 670 million path computations. This makes a full enumeration computationally infeasible.

To overcome this limitation, I used Monte Carlo simulation to approximate the expected payoff of the Asian option. Specifically, for each (r, σ) pair, I simulated 100,000 independent price paths based on the binomial model, using the corresponding up/down factors and risk-neutral probabilities. For each simulated path, I computed the arithmetic average of the stock prices, evaluated the payoff $\max(M - K, 0)$, and discounted the expected payoff back to present value using the formula:

$$V_0 \approx e^{-rT} \cdot \frac{1}{N} \sum_{i=1}^{N} \max(M_i - K, 0)$$

where N is the number of simulated paths, and M_i is the average price along the i-th simulated trajectory.

Results. The resulting option prices are summarized in the following table:



```
r         0.005    0.010    0.020    0.030
sigma
0.15    14.5185  14.7500  15.8276  16.9396
0.20    19.1186  19.6217  20.5025  21.4079
0.25    23.7535  24.3135  24.9592  25.9894
0.30    28.4423  28.8303  29.6923  30.6521
0.35    32.8804  33.1275  34.2551  35.1760
```

Comments : The results show a clear and expected trend:

- **Volatility effect:** As the volatility σ increases, the value of the Asian call option increases for all interest rates. This is consistent with the fact that higher volatility increases the dispersion of possible average prices, which increases the probability that the option ends up in-the-money.

- **Interest rate effect:** For a fixed volatility, higher values of r also lead to higher option prices. This can be attributed to two effects: (1) higher discounting tends to reduce the present value, but (2) higher r shifts the expected price upward under the risk-neutral measure, increasing the average and hence the likelihood of positive payoffs. In this case, the upward shift dominates the discounting effect.

Overall, the robustness check confirms that the model behaves consistently with financial theory, and it highlights how sensitive the Asian option price is to volatility in particular.

(g) Normal Approximation Using Empirical Average Prices

In this section, I apply a normal approximation to estimate the price of the Asian call option using the empirical distribution of average prices computed in part (b). Rather than approximating the distribution of the arithmetic average theoretically, I base the approximation directly on the average prices already computed along all 2^{25} binomial paths.

Methodology

From part (b), I already obtained a list of arithmetic average prices, denoted M_i, for each binomial path $i = 1, \dots, 2^n$, as well as the corresponding path probabilities $\mathbb{P}_i$ (implicitly used in computing the weighted payoffs). To estimate the price using a normal approximation, I followed these steps:

1. I recomputed the risk-neutral probabilities $\mathbb{P}_i$ for each path based on the number of up and down moves using the binomial model:

$$\mathbb{P}_i = q^{k_i}(1-q)^{n-k_i}, \quad \text{where } q = \frac{1 + r \cdot \Delta t - d}{u - d}$$

2. Using the precomputed average prices M_i and the corresponding $\mathbb{P}_i$, I calculated the expected value μ and standard deviation σ of the average stock price under the risk-neutral measure:

$$\mu = \sum_i \mathbb{P}_i M_i, \qquad \sigma = \sqrt{\sum_i \mathbb{P}_i (M_i - \mu)^2}$$

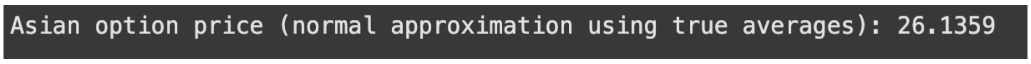
3. Assuming that the distribution of M is approximately normal with these parameters, I used the pricing formula for the Asian call option:

$$V_0 \approx e^{-rT} \left[(\mu - K) \cdot \Phi(d) + \sigma \cdot \phi(d) \right], \qquad d = \frac{\mu - K}{\sigma}$$

   where Φ and ϕ denote the cumulative distribution function (CDF) and probability density function (PDF) of the standard normal distribution, respectively.

Result

Using this approach, I obtained the following approximation for the price of the Asian option:



```
Asian option price (normal approximation using true averages): 26.1359
```

This result is consistent with the exact pricing result obtained earlier (approximately 25.95), confirming the accuracy of the normal approximation when applied to the true empirical distribution of average prices.

7 Appendix : Code

```python
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm

msft = pd.read_csv("/content/drive/MyDrive/HSG PA /MSFT_2020_2025 .csv")[1:]


msft = msft.dropna().reset_index(drop=True).drop('Price', axis = 1)


for col in msft.columns:
    if col != 'Date':
        msft[col] = msft[col].astype(float)

#(a) : Binomial Tree (visual and matrix )
n = 25
T = 1
r = 0.01
S0 = msft['Close'].values[-1]
delta_t = T / n

sigma_daily = msft['returns'].std()
sigma_annualized = sigma_daily * np.sqrt(250)

u = np.exp(sigma_annualized * np.sqrt(delta_t))
d = np.exp(-sigma_annualized * np.sqrt(delta_t))
K = S0
discount = np.exp(-r * delta_t)


stock_tree = np.zeros((n + 1, n + 1))
for i in range(n + 1):
    for j in range(i + 1):
        stock_tree[j, i] = S0 * (u ** (i - j)) * (d ** j)

stock_tree_df = pd.DataFrame(stock_tree).replace(0, np.nan).round(2)
stock_tree_df


fig, ax = plt.subplots(figsize=(24, 12))

text_matrix = stock_tree_df.fillna("").astype(str).values

table = ax.table(
    cellText=text_matrix,
    rowLabels=stock_tree_df.index,
    colLabels=stock_tree_df.columns,
    cellLoc='center',
    loc='center'
)

table.auto_set_font_size(False)
table.set_fontsize(7)
table.scale(1.0, 1.5)

ax.axis('off')
plt.title("Binomial Tree for MSFT (n=25)", fontsize=14)

plt.savefig("binomial_tree_matrix_readable.png", bbox_inches='tight', dpi=300)
plt.show()

def plot_binomial_tree_visual(stock_tree_df, levels=4):
    fig, ax = plt.subplots(figsize=(10, 5))
    positions = {}

    for i in range(levels + 1):
        for j in range(i + 1):
            price = stock_tree_df.iloc[j, i]
            if not np.isnan(price):
                x = i
                y = -j + i / 2
                positions[(j, i)] = (x, y)
                ax.text(x, y, f'{price:.2f}', ha='center', va='center',
                        bbox=dict(boxstyle="round", facecolor="lightblue",
                            edgecolor="gray"))

    for i in range(levels):
        for j in range(i + 1):
            if (j, i) in positions:
                x, y = positions[(j, i)]
                if (j, i + 1) in positions:
                    xu, yu = positions[(j, i + 1)]
                    ax.plot([x, xu], [y, yu], 'k-', lw=1)
                if (j + 1, i + 1) in positions:
                    xd, yd = positions[(j + 1, i + 1)]
                    ax.plot([x, xd], [y, yd], 'k-', lw=1)

    ax.set_title("Binomial Tree for Stock Prices (First 4 Levels)")
    ax.axis('off')
    plt.tight_layout()
    plt.show()

plot_binomial_tree_visual(stock_tree_df, levels=4)


#(b)-(c) : Average price and payoff  :


n = 25
T = 1
r = 0.01
S0 = msft['Close'].values[-1]
delta_t = T / n

sigma_daily = msft['returns'].std()
sigma_annualized = sigma_daily * np.sqrt(250)

u = np.exp(sigma_annualized * np.sqrt(delta_t))
d = np.exp(-sigma_annualized * np.sqrt(delta_t))
K = S0


payoffs = []
prices = []
weighted_payoffs = []
q = (1 + r * delta_t - d) / (u - d)


for i in tqdm(range(0, 2**n)):
    path = [(i >> j) & 1 for j in range(n - 1, -1, -1)]

    multipliers = np.where(np.array(path) == 0, u, d)
    num_ups = np.sum(np.array(path) == 0)
    num_downs = len(path) - num_ups

    path_probability = (q ** num_ups) * ((1 - q) ** num_downs)

    price = S0 * np.cumprod(multipliers)
    M = np.mean(price)
    prices.append(M)
    payoffs.append(max(M - K, 0))
    weighted_payoffs.append(max(M - K, 0) * path_probability)


#(d) Risk-neutral probability :

q = (1 + r * delta_t - d) / (u - d)
print(f"Risk-neutral probability q = {q:.4f}")


#(e) Option Price

```

```python
option_price = np.exp(-r * T) * np.sum(weighted_payoffs)
print(f"Asian option price at t = 0: {option_price:.4f}")


#(f) Robustness Check :

num_simulations = 100_000

r_values = [0.005, 0.01, 0.02, 0.03]
sigma_values = [0.15, 0.20, 0.25, 0.30, 0.35]

results = []


for r, sigma in tqdm(product(r_values, sigma_values),
    total=len(r_values)*len(sigma_values), desc="Monte Carlo Robustness Check"):

    u = np.exp(sigma * np.sqrt(delta_t))
    d = np.exp(-sigma * np.sqrt(delta_t))
    q = (1 + r * delta_t - d) / (u - d)

    payoffs = []

    for _ in range(num_simulations):

        path = np.random.choice([0, 1], size=n, p=[q, 1 - q])
        multipliers = np.where(path == 0, u, d)
        price_path = S0 * np.cumprod(multipliers)
        average_price = np.mean(price_path)
        payoff = max(average_price - K, 0)
        payoffs.append(payoff)

    option_price = np.exp(-r * T) * np.mean(payoffs)
    results.append((r, sigma, option_price))
    print(f"r = {r:.3f}, sigma = {sigma:.2f}     Monte Carlo Price =
        {option_price:.4f}")


df_results_mc = pd.DataFrame(results, columns=["r", "sigma", "option_price"])
pivot_mc = df_results_mc.pivot(index="sigma", columns="r", values="option_price")
print(pivot_mc.round(4))


#(g) Normal Approximation :

from scipy.stats import norm
import numpy as np

r = 0.01
sigma_annualized = sigma_daily * np.sqrt(250)
delta_t = T / n
u = np.exp(sigma_annualized * np.sqrt(delta_t))
d = np.exp(-sigma_annualized * np.sqrt(delta_t))
q = (1 + r * delta_t - d) / (u - d)
K = S0
path_probs = []

for i in range(2**n):
    path = [(i >> j) & 1 for j in range(n - 1, -1, -1)]
    num_ups = sum(bit == 0 for bit in path)
    num_downs = n - num_ups
    prob = (q ** num_ups) * ((1 - q) ** num_downs)
    path_probs.append(prob)


prices_np = np.array(prices)
path_probs_np = np.array(path_probs)


path_probs_np /= path_probs_np.sum()


mu = np.sum(prices_np * path_probs_np)
```

```
var = np.sum(path_probs_np * (prices_np - mu) ** 2)
std = np.sqrt(var)


d = (mu - K) / std
option_price_normal = np.exp(-r * T) * ((mu - K) * norm.cdf(d) + std * norm.pdf(d))

print(f"Asian option price (normal approximation using true averages):
    {option_price_normal:.4f}")
```