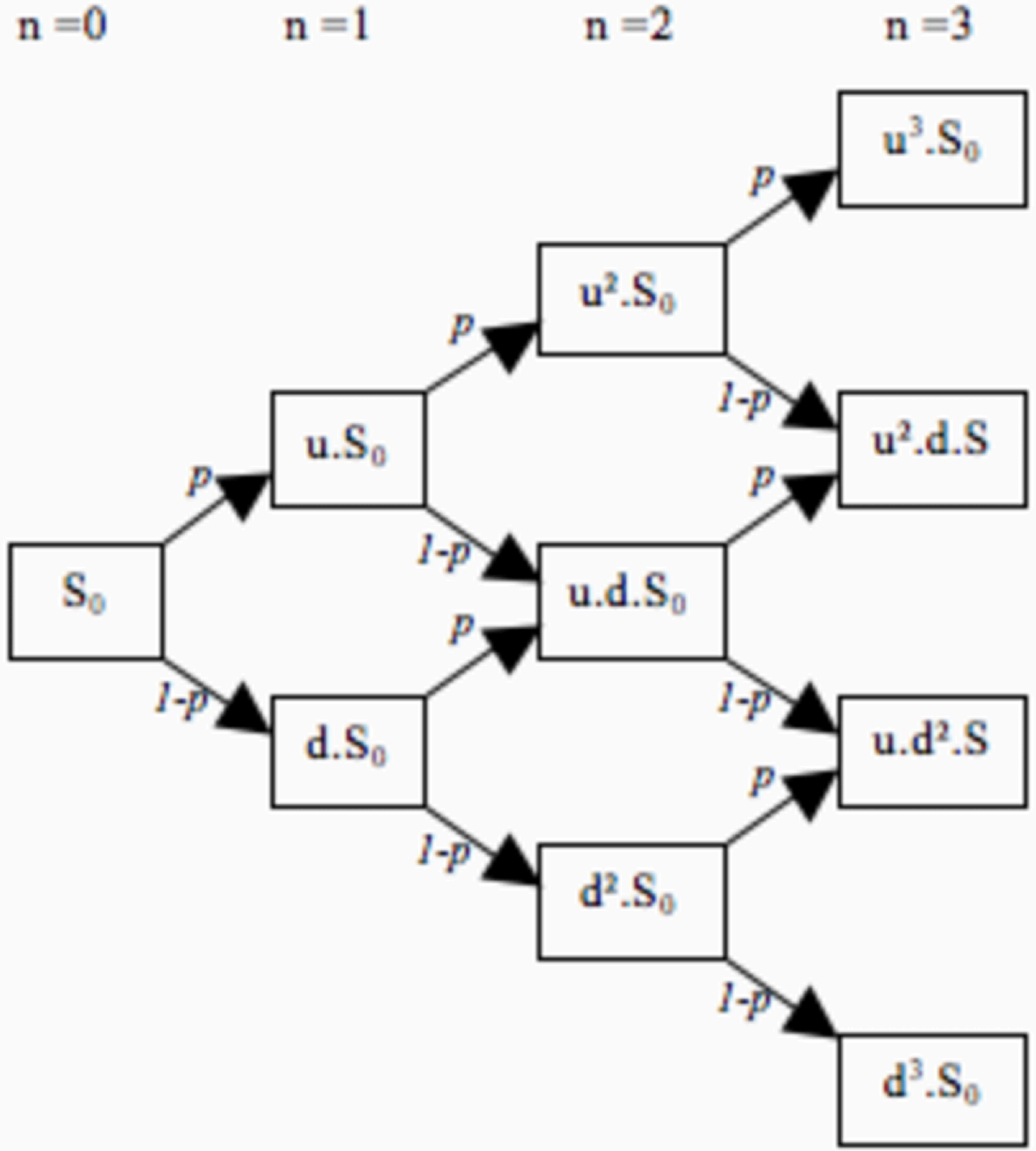

n =0
n =1
n =2
n =3
S0
u.S0
d.S0
u².S0
u.d.S0
d².S0
u³.S0
u².d.S
u.d².S
d³.S0
p
1-p

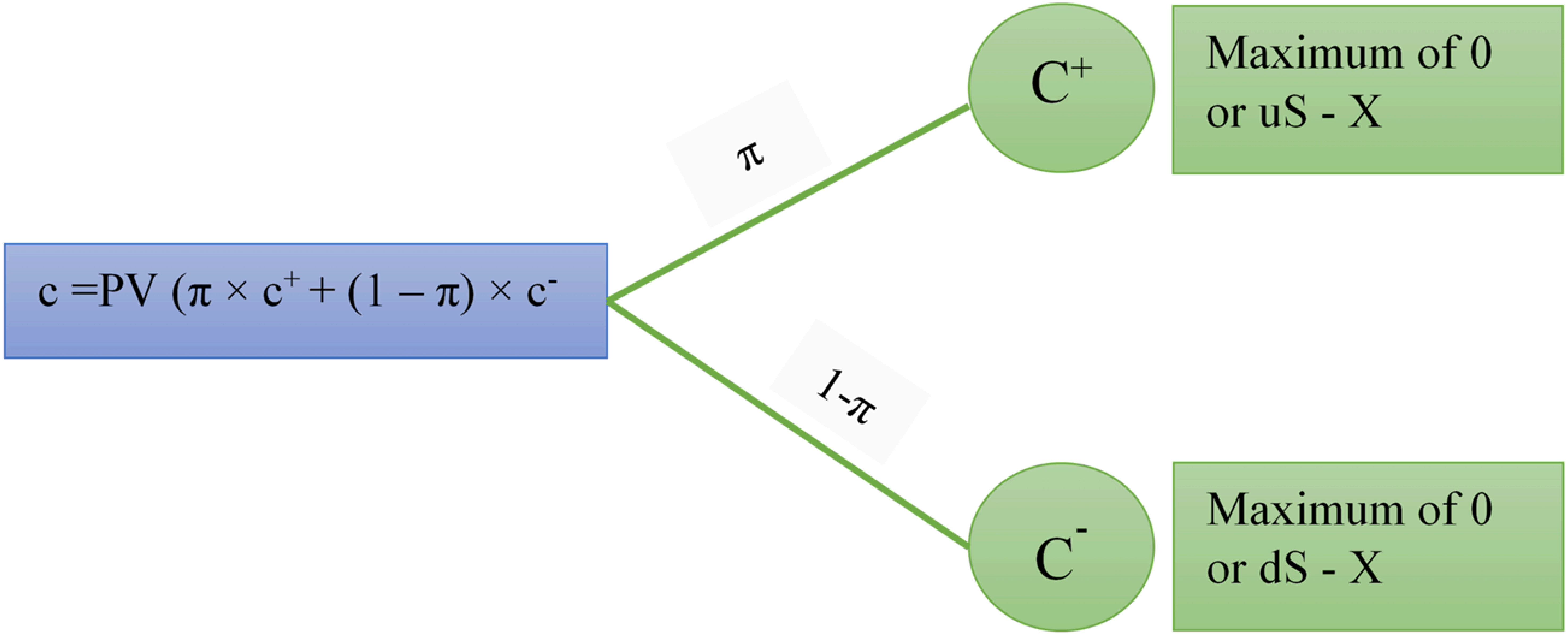
Call Option Payoff

1. **Definition:** A call option gives the holder the right to buy an underlying asset at a predetermined price (strike price) before or at expiration.
2. **Payoff Formula:**

$$\text{Payoff} = \max(0, S - K)$$

   Where:

   - S = Price of the underlying asset at expiration.
   - K = Strike price.
3. **Scenarios:**
   - **In-the-Money (ITM):** If $S > K$:
     - Payoff = $S - K$ (profit).
     - Example: $S = 70, K = 50$ → Payoff = $70 - 50 = 20$.
   - **At-the-Money (ATM):** If $S = K$:
     - Payoff = 0.
     - Example: $S = K = 50$ → Payoff = 0.
   - **Out-of-the-Money (OTM):** If $S < K$:
     - Payoff = 0.
     - Example: $S = 30, K = 50$ → Payoff = 0.


$c = PV(\pi \times c^{+} + (1 - \pi) \times c^{-}$
π
$1-\pi$
C^{+}
Maximum of 0 or uS - X
C^{-}
Maximum of 0 or dS - X

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

$$u = e^{\sigma\sqrt{\Delta t}}$$

$$d = e^{-\sigma\sqrt{\Delta t}}$$

📝 Exercise – Pricing a European Call Option Using a Binomial Tree

You are asked to **price a European call option** using a **discrete-time binomial tree model**.

Given:

- **Current stock price (S0)** = 100 €
- **Strike price (K)** = 105 €
- **Time to maturity (T)** = 1 year
- **Risk-free interest rate (r)** = 5% per year (0.05)
- **Volatility (sigma)** = 20% per year (0.2)
- **Number of steps in the binomial tree (N)** = 3

Binomial Tree Characteristics

1. **Discrete time steps:** The total time to maturity is divided into `N` equal intervals.
2. **Stock price evolution:** At each step, the stock can move **up** by factor $u = e^{\sigma\sqrt{\Delta t}}$ or **down** by factor $d = 1/u$, where $\Delta t = T/N$.
3. **Risk-neutral probability:**

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

4. **Option valuation:**
   - Compute the **option payoff at maturity**: $\max(S_T - K, 0)$ for a call.
   - Work **backwards through the tree** using:

$$V_j = e^{-r\Delta t}(p \cdot V_{\text{up}} + (1-p) \cdot V_{\text{down}})$$

```python
# Parameters
S0 = 100        # Current stock price
K = 105         # Strike price
T = 1           # Time to maturity in years
r = 0.05        # Risk-free rate (annual, continuous)
sigma = 0.2     # Volatility (annual)
N = 3           # Number of steps in the binomial tree

# Step 1: Compute parameters
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

# Step 2: Initialize stock price tree
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)

# Step 3: Initialize option value at maturity
option_tree = np.zeros_like(stock_tree)
option_tree[:, N] = np.maximum(stock_tree[:, N] - K, 0)  # call option payoff

# Step 4: Backward induction to price the option
for i in range(N-1, -1, -1):
    for j in range(i+1):
        option_tree[j, i] = np.exp(-r*dt) * (p * option_tree[j, i+1] + (1-p) * option_tree[j+1, i+1]

# Step 5: Option price at root
option_price = option_tree[0, 0]
print(f"European Call Option Price: {option_price:.2f}")
```

$$c = S_0\, N(d_1) - K\, e^{-rT} N(d_2)$$

$$p = K\, e^{-rT}\, N(-d_2) - S_0\, N(-d_1)$$

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

$$d_2 = \frac{\ln(S_0 / K) + (r - \sigma^2 / 2)T}{\sigma\sqrt{T}} = d_1 - \sigma\sqrt{T}$$

📝 Exercise – Pricing a European Option Using Black-Scholes

You are asked to **price a European call and put option** using the **Black-Scholes formula**.

Given:

- **Current stock price (S0)** = 100 €
- **Strike price (K)** = 105 €
- **Time to maturity (T)** = 1 year
- **Risk-free interest rate (r)** = 5% per year (0.05)
- **Volatility (sigma)** = 20% per year (0.2)

Task:

1. Compute d_1 and d_2 using the Black-Scholes formulas:

$$d_1 = \frac{\ln(S_0/K) + (r + \sigma^2/2)T}{\sigma\sqrt{T}}, \quad d_2 = d_1 - \sigma\sqrt{T}$$

2. Compute the **call and put prices**:

$$C = S_0 N(d_1) - Ke^{-rT}N(d_2), \quad P = Ke^{-rT}N(-d_2) - S_0 N(-d_1)$$

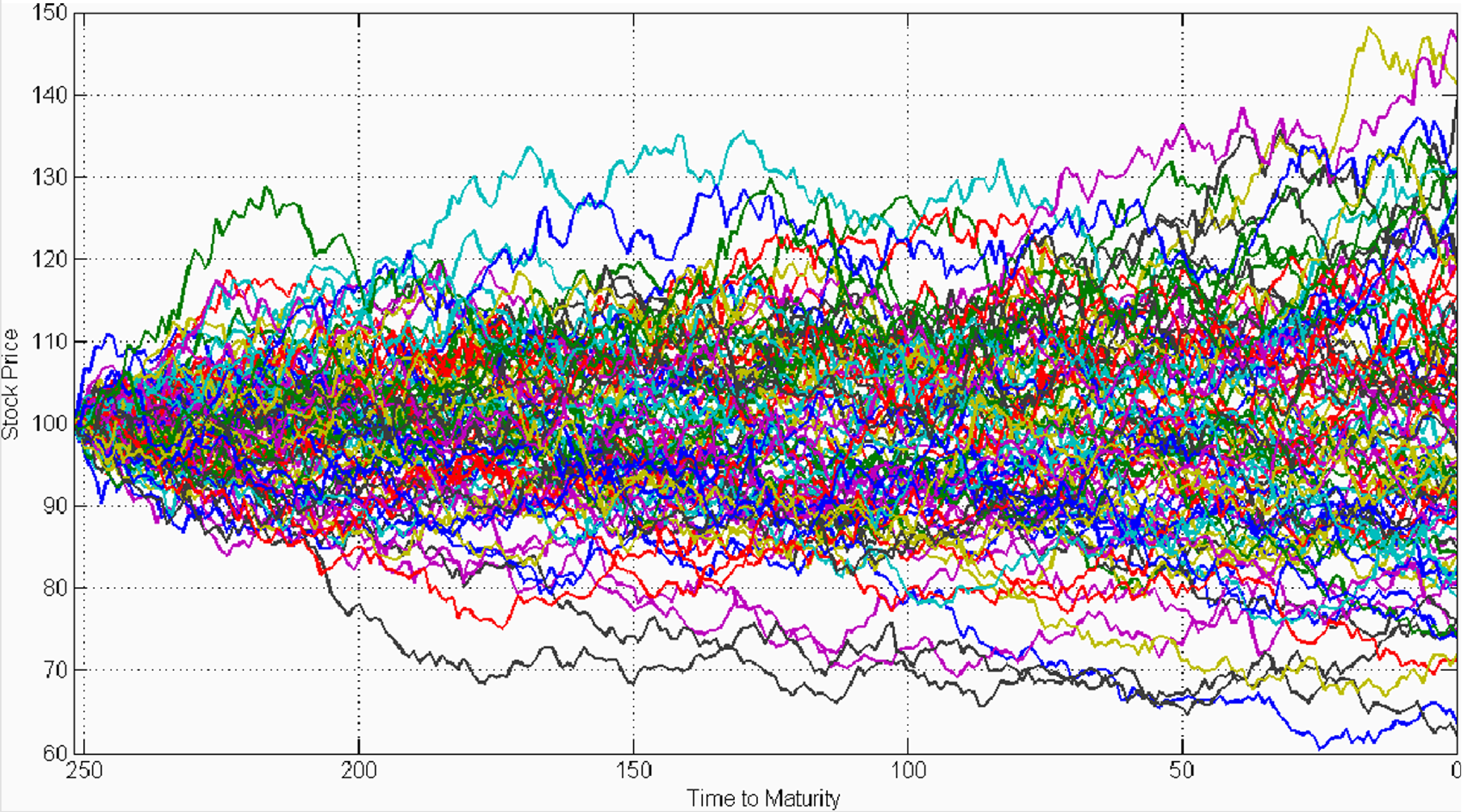
3. Use Python and `scipy.stats.norm.cdf()` to compute $N(d_1)$ and $N(d_2)$.

💡 **Hint:**

- $N(d_1)$ and $N(d_2)$ are cumulative probabilities of a standard normal distribution:

```python
from scipy.stats import norm
Nd1 = norm.cdf(d1)
Nd2 = norm.cdf(d2)
```

```python
import numpy as np
from scipy.stats import norm

# Parameters
S0 = 100        # Current stock price
K = 105         # Strike price
T = 1           # Time to maturity (years)
r = 0.05        # Risk-free interest rate (annual)
sigma = 0.2     # Volatility (annual)

# Step 1: Compute d1 and d2
d1 = (np.log(S0 / K) + (r + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
d2 = d1 - sigma * np.sqrt(T)

# Step 2: Compute cumulative normal values
Nd1 = norm.cdf(d1)
Nd2 = norm.cdf(d2)

# Step 3: Compute call and put prices
call_price = S0 * Nd1 - K * np.exp(-r * T) * Nd2
put_price = K * np.exp(-r * T) * norm.cdf(-d2) - S0 * norm.cdf(-d1)

# Step 4: Print results
print(f"European Call Option Price: {call_price:.2f}")
print(f"European Put Option Price: {put_price:.2f}")
```


150
140
130
120
110
100
90
80
70
60
Stock Price
250
200
150
100
50
0
Time to Maturity

$$S(\Delta t) = S(0)\exp\left[\left(\mu - \frac{\sigma^2}{2}\right)\Delta t + \left(\sigma\sqrt{\Delta t}\right)\varepsilon\right]$$

📝 Exercise – Pricing a European Call Option Using Monte Carlo

You are asked to **price a European call option** using a **Monte Carlo simulation** approach.

Given:

- **Current stock price (S0)** = 100 €
- **Strike price (K)** = 105 €
- **Time to maturity (T)** = 1 year
- **Risk-free interest rate (r)** = 5% per year (0.05)
- **Volatility (sigma)** = 20% per year (0.2)
- **Number of simulations (M)** = 100,000

Monte Carlo Approach

1. **Simulate terminal stock prices** under the **risk-neutral measure**:

$$S_T = S_0 \cdot e^{(r - 0.5\sigma^2)T + \sigma\sqrt{T} \cdot Z}$$

where $Z \sim N(0,1)$ (standard normal random variable).

2. **Compute the payoff** for each simulated path:

$$\text{payoff} = \max(S_T - K, 0) \quad \text{for a call}$$

3. **Discount the average payoff** to present value:

$$C = e^{-rT} \cdot \text{mean}(\text{payoff})$$

```python
import numpy as np

# Parameters
S0 = 100       # Initial stock price
T = 1          # Time to maturity (years)
r = 0.05       # Risk-free rate
sigma = 0.2    # Volatility
N = 5          # Number of time steps
M = 3          # Number of simulated paths

dt = T / N

# Initialize array to store stock paths
stock_paths = np.zeros((M, N+1))

# Loop over each path
for m in range(M):
    stock_paths[m, 0] = S0  # initial stock price
    # Loop over each time step
    for i in range(1, N+1):
        Z = np.random.normal()
        stock_paths[m, i] = stock_paths[m, i-1] * np.exp((r - 0.5*sigma**2)*dt + sigma*np.sqrt(dt)*Z

# Print evolution of each path
for m in range(M):
    print(f"Path {m+1}: {stock_paths[m, :]}")
```

```python
import numpy as np

# Parameters
S0 = 100        # Current stock price
K = 105         # Strike price
T = 1           # Time to maturity (years)
r = 0.05        # Risk-free rate
sigma = 0.2     # Volatility
M = 100_000     # Number of simulations

# Step 1: Simulate terminal stock prices
Z = np.random.normal(0, 1, M)  # standard normal random variables
ST = S0 * np.exp((r - 0.5 * sigma**2) * T + sigma * np.sqrt(T) * Z)

# Step 2: Compute payoffs for a call option
payoffs = np.maximum(ST - K, 0)

# Step 3: Discount average payoff to present value
call_price = np.exp(-r * T) * np.mean(payoffs)

# Step 4: Print the result
print(f"European Call Option Price (Monte Carlo): {call_price:.2f}")
```

Exotic Barrier Options – Explanation

Definition:

A **barrier option** is a type of **exotic option** whose **payoff depends not only on the price of the underlying at maturity** but also on whether the underlying **crosses a certain barrier level** during its life.

Key Characteristics

1. **Barrier Level (H):**
   - The price level that triggers the barrier condition.
2. **Types of Barrier Options:**
   - **Knock-in:** the option becomes **active** only if the underlying **hits the barrier**.
     - Example: Up-and-in call → becomes valid if stock rises above H.
   - **Knock-out:** the option **expires worthless** if the underlying **hits the barrier**.
     - Example: Down-and-out put → ceases to exist if stock falls below H.
3. **Direction:**
   - **Up:** barrier is above the initial stock price.
   - **Down:** barrier is below the initial stock price.
4. **Payoff:**
   - Once the barrier condition is satisfied (or not), the payoff is usually like a **standard European call or put**, but conditional on the barrier event.

Example (Conceptual)

- **Up-and-out call**:
  - Strike = 100, Barrier = 120, Stock starts at 100.
  - If at any time before maturity the stock **reaches 120**, the option **expires worthless**.
  - If it never hits 120, payoff = $\max(S_T - 100, 0)$.

Why Exotic?

- Unlike **plain vanilla options**, barrier options' value **depends on the entire path** of the underlying, not just the final price.
- This makes them **path-dependent** and **more complex to price**.

📝 Exercise – Pricing Barrier Options

You are asked to **price European barrier options** using a **Monte Carlo simulation**.

Given:

- **Current stock price (S0)** = 100 €
- **Strike price (K)** = 105 €
- **Time to maturity (T)** = 1 year
- **Risk-free interest rate (r)** = 5% per year (0.05)
- **Volatility (sigma)** = 20% per year (0.2)
- **Number of simulations (M)** = 100,000
- **Number of time steps per path (N)** = 50

Options to Price

1. **Up-and-Out Call:**
   - Barrier $H = 120$
   - Payoff = 0 if the stock **ever reaches or exceeds H**; otherwise $\max(S_T - K, 0)$
2. **Down-and-Out Put:**
   - Barrier $H = 80$
   - Payoff = 0 if the stock **ever falls below H**; otherwise $\max(K - S_T, 0)$

Task:

1. Simulate **stock price paths** using **geometric Brownian motion**.
2. For each path, check if the barrier is breached.
3. Compute the **option payoff** depending on the barrier condition.
4. Compute the **Monte Carlo estimate** of the option price by discounting the average payoff to present value.

```python
S0 = 100        # Initial stock price
K = 105         # Strike price
T = 1           # Time to maturity
r = 0.05        # Risk-free rate
sigma = 0.2     # Volatility
M = 100_000     # Number of simulations
N = 50          # Time steps per path

dt = T / N

# Barriers
H_up = 120      # Up-and-Out barrier
H_down = 80     # Down-and-Out barrier

# Simulate stock paths
ST = np.zeros((M, N+1))
ST[:, 0] = S0

for i in range(1, N+1):
    Z = np.random.normal(0, 1, M)
    ST[:, i] = ST[:, i-1] * np.exp((r - 0.5 * sigma**2) * dt + sigma * np.sqrt(dt) * Z)

# Up-and-Out Call payoff
payoff_up_out = np.maximum(ST[:, -1] - K, 0)
# Set payoff to 0 if barrier crossed
for m in range(M):
    if np.any(ST[m, :] >= H_up):
        payoff_up_out[m] = 0

# Down-and-Out Put payoff
payoff_down_out = np.maximum(K - ST[:, -1], 0)
# Set payoff to 0 if barrier crossed
for m in range(M):
    if np.any(ST[m, :] <= H_down):
        payoff_down_out[m] = 0

# Discount payoffs to present value
price_up_out = np.exp(-r * T) * np.mean(payoff_up_out)
price_down_out = np.exp(-r * T) * np.mean(payoff_down_out)

# Print results
print(f"Up-and-Out Call Price: {price_up_out:.2f}")
print(f"Down-and-Out Put Price: {price_down_out:.2f}")
```

KIKO Option (Knock-In Knock-Out)

1. **Definition:**
   - A **KIKO option** is an **exotic option** that combines both **knock-in** and **knock-out** barriers.
   - The option only **becomes active** if it hits a **knock-in barrier**, but it can also **expire worthless** if it hits a **knock-out barrier**.
2. **Two Barriers:**
   - **Knock-In Barrier (H_in):** The option **activates** when the underlying reaches this level.
   - **Knock-Out Barrier (H_out):** The option **expires** if the underlying touches this level.
3. **Payoff:**
   - If the knock-in is triggered **before maturity** and the knock-out is **not triggered**, the payoff is usually like a **vanilla call or put**.
   - If knock-out is triggered, **payoff = 0**.
   - If knock-in is never triggered, **payoff = 0**.
4. **Path-Dependent:**
   - Like barrier options, the **payoff depends on the entire path** of the underlying, not just the final price.
5. **Use Case:**
   - KIKO options are often used in **FX markets** or by **structured products** to create specific risk/reward profiles.

📝 Exercise – Pricing a KIKO (Knock-In Knock-Out) Option

You are asked to **price a European KIKO option** using a **Monte Carlo simulation**.

Given:

- **Current stock price (S0)** = 100 €
- **Strike price (K)** = 105 €
- **Time to maturity (T)** = 1 year
- **Risk-free interest rate (r)** = 5% per year (0.05)
- **Volatility (sigma)** = 20% per year (0.2)
- **Number of simulations (M)** = 100,000
- **Number of time steps per path (N)** = 50
- **Knock-In Barrier (H_in)** = 110
- **Knock-Out Barrier (H_out)** = 120

Task:

1. Simulate **stock price paths** using **geometric Brownian motion**.
2. For each path, determine:
   - **Knock-In:** check if the stock **ever reaches or exceeds H_in**.
   - **Knock-Out:** check if the stock **ever reaches or exceeds H_out**.
3. Compute the **payoff at maturity:**
   - If **knock-in is triggered** and **knock-out is not triggered**, payoff = $\max(S_T - K, 0)$ (for a call).
   - Otherwise, payoff = 0.
4. Compute the **Monte Carlo estimate** of the KIKO option price by **discounting the average payoff** to present value.

```python
S0 = 100        # Initial stock price
K = 105         # Strike price
T = 1           # Time to maturity
r = 0.05        # Risk-free rate
sigma = 0.2     # Volatility
M = 100_000     # Number of simulations
N = 50          # Number of time steps

H_in = 110      # Knock-In barrier
H_out = 120     # Knock-Out barrier

dt = T / N

# Simulate stock paths
ST = np.zeros((M, N+1))
ST[:, 0] = S0

for i in range(1, N+1):
    Z = np.random.normal(0, 1, M)
    ST[:, i] = ST[:, i-1] * np.exp((r - 0.5*sigma**2)*dt + sigma*np.sqrt(dt)*Z)

# Compute KIKO payoffs
payoffs = np.zeros(M)

for m in range(M):
    path = ST[m, :]
    knock_in = np.any(path >= H_in)
    knock_out = np.any(path >= H_out)
    if knock_in and not knock_out:
        payoffs[m] = max(path[-1] - K, 0)
    else:
        payoffs[m] = 0

# Discount to present value
kiko_price = np.exp(-r*T) * np.mean(payoffs)

print(f"KIKO Call Option Price: {kiko_price:.2f}")
```