Numerical Method for Financial Engineering

Final Report Project

- Yu-Ching Liao ycliao3@illinois.edu

Basic Import

In [12]:
```
import numpy as np
import math
from datetime import datetime
```

Task 1

Price European put vanilla. Spot=100, sigma=20%, valuation date=4/28/2023, expiry=12/15/2023, rate=2.5% per annum, K=100

In [33]:
```
# Parameters
S0 = 100
K = 100
sigma = 0.2
r = 0.025
valuation_date = "4/28/2023"
expiry_date = "12/15/2023"

valuation_datetime = datetime.strptime(valuation_date, "%m/%d/%Y")
expiry_datetime = datetime.strptime(expiry_date, "%m/%d/%Y")
T_days = (expiry_datetime - valuation_datetime).days
T = T_days / 365.0

# PDE grid parameters
M = 100  # number of grid points in the asset price direction
N = 1000  # number of time steps

# Set up the grid
```

```
dt = T / N
dS = 2 * S0 / M
S = np.linspace(0, 2 * S0, M + 1)
tau = np.linspace(0, T, N + 1)

# Initialize the option price grid
V = np.zeros((M + 1, N + 1))

# Set up boundary conditions
V[:, -1] = np.maximum(K - S, 0)  # Terminal condition (at expiry)
V[-1, :-1] = 0  # Upper boundary (large asset price)
V[0, :-1] = K * np.exp(-r * (T - tau[:-1]))  # Lower boundary (zero asset pr

# Time-stepping loop
for n in range(N - 1, -1, -1):
    for m in range(1, M):
        V[m, n] = V[m, n+1] + dt * (
            0.5 * sigma ** 2 * S[m] ** 2 * (V[m+1, n+1] - 2 * V[m, n+1] + V[
            + r * S[m] * (V[m+1, n+1] - V[m-1, n+1]) / (2 * dS)
            - r * V[m, n+1]
        )

put_price = V[M//2, 0]
print("European put option price:", put_price)
```

```
European put option price: 5.525524804683739
```

Task 2

Price American put vanilla. Spot=100, sigma=20%, valuation date=4/28/2023, expiry=12/15/2023, rate=2.5% per annum, K=100

In [32]:

```
# Parameters
S0 = 100
K = 100
sigma = 0.2
r = 0.025
valuation_date = "4/28/2023"
expiry_date = "12/15/2023"

valuation_datetime = datetime.strptime(valuation_date, "%m/%d/%Y")
expiry_datetime = datetime.strptime(expiry_date, "%m/%d/%Y")
T_days = (expiry_datetime - valuation_datetime).days
T = T_days / 365.0

# PDE grid parameters
M = 100  # number of grid points in the asset price direction
N = 1000  # number of time steps

# Set up the grid
dt = T / N
dS = 2 * S0 / M
S = np.linspace(0, 2 * S0, M + 1)
tau = np.linspace(0, T, N + 1)
```

```
# Initialize the option price grid
V = np.zeros((M + 1, N + 1))

# Set up boundary conditions
V[:, -1] = np.maximum(K - S, 0)  # Terminal condition (at expiry)
V[-1, :-1] = 0  # Upper boundary (large asset price)
V[0, :-1] = K * np.exp(-r * (T - tau[:-1]))  # Lower boundary (zero asset pr

# Time-stepping loop
for n in range(N - 1, -1, -1):
    for m in range(1, M):
        V[m, n] = V[m, n+1] + dt * (
            0.5 * sigma ** 2 * S[m] ** 2 * (V[m+1, n+1] - 2 * V[m, n+1] + V[
            + r * S[m] * (V[m+1, n+1] - V[m-1, n+1]) / (2 * dS)
            - r * V[m, n+1]
        )
        # Early exercise condition
        V[m, n] = max(V[m, n], K - S[m])

american_put_price = V[M//2, 0]
print("American put option price:", american_put_price)
```

```
American put option price: 5.658390797709389
```

Task 3

Price Continuous down and in put barrier option where barrier level is =80, Spot=100, sigma=20%, valuation date=4/28/2023, expiry=12/15/2023, rate=2.5% per annum, K=100.

In [30]:
```
# Parameters
S0 = 100
K = 100
sigma = 0.2
r = 0.025
valuation_date = "4/28/2023"
expiry_date = "12/15/2023"
barrier_level = 80

valuation_datetime = datetime.strptime(valuation_date, "%m/%d/%Y")
expiry_datetime = datetime.strptime(expiry_date, "%m/%d/%Y")
T_days = (expiry_datetime - valuation_datetime).days
T = T_days / 365.0

# PDE grid parameters
M = 100  # number of grid points in the asset price direction
N = 1000  # number of time steps

# Set up the grid
dt = T / N
dS = 2 * S0 / M
S = np.linspace(0, 2 * S0, M + 1)
```

```
tau = np.linspace(0, T, N + 1)

# Initialize the option price grid
V = np.zeros((M + 1, N + 1))

# Set up boundary conditions
V[:, -1] = np.maximum(K - S, 0)  # Terminal condition (at expiry)
V[-1, :-1] = 0  # Upper boundary (large asset price)
V[0, :-1] = K * np.exp(-r * (T - tau[:-1]))  # Lower boundary (zero asset pr

# Time-stepping loop
for n in range(N - 1, -1, -1):
    for m in range(1, M):
        V[m, n] = V[m, n+1] + dt * (
            0.5 * sigma ** 2 * S[m] ** 2 * (V[m+1, n+1] - 2 * V[m, n+1] + V[
            + r * S[m] * (V[m+1, n+1] - V[m-1, n+1]) / (2 * dS)
            - r * V[m, n+1]
        )

european_put_price = V[M//2, 0]

# Calculate Down-and-Out put option price
down_and_out_put_price = down_and_out_put(S0, K, r, sigma, T, barrier_level)

# Calculate Down-and-In put option price
down_and_in_put_price_pde = european_put_price - down_and_out_put_price
print("PDE continuous down-and-in put barrier option price:", down_and_in_pu
```

```
PDE continuous down-and-in put barrier option price: 5.774434321909334
```

Task 4

Use Black Scholes price to reconcile your PDE price with prices from a and c. Use trinomial tree to reconcile the price from b. Note, you need to find out how to price a continuous barrier using Analytic formula.

Reconciling A and C

In [44]:
```
def d1(S, K, r, sigma, T):
    return (np.log(S / K) + (r + 0.5 * sigma ** 2) * T) / (sigma * np.sqrt(T

def d2(S, K, r, sigma, T):
    return d1(S, K, r, sigma, T) - sigma * np.sqrt(T)

def black_scholes_put(S, K, r, sigma, T):
    d1_value = d1(S, K, r, sigma, T)
    d2_value = d2(S, K, r, sigma, T)
    return K * np.exp(-r * T) * norm.cdf(-d2_value) - S * norm.cdf(-d1_value

def down_and_out_put(S, K, r, sigma, T, H):
    P = black_scholes_put(S, K, r, sigma, T)
    lambda_ = (r + 0.5 * sigma ** 2) / sigma ** 2
```

```
    y = (np.log(H ** 2 / (S * K)) / (sigma * np.sqrt(T))) + lambda_ * sigma
    DOP = P * (H / S) ** (2 * lambda_) * norm.cdf(y) - K * np.exp(-r * T) *
    return DOP

def down_and_in_put(S, K, r, sigma, T, H):
    P = black_scholes_put(S, K, r, sigma, T)
    DOP = down_and_out_put(S, K, r, sigma, T, H)
    return P - DOP


# Parameters
S0 = 100
K = 100
sigma = 0.2
r = 0.025
valuation_date = "4/28/2023"
expiry_date = "12/15/2023"

valuation_datetime = datetime.strptime(valuation_date, "%m/%d/%Y")
expiry_datetime = datetime.strptime(expiry_date, "%m/%d/%Y")
T_days = (expiry_datetime - valuation_datetime).days
T = T_days / 365.0

# Black-Scholes European put price
bs_european_put_price = black_scholes_put(S0, K, r, sigma, T)
print("Reconciling A: Black-Scholes European put option price:", bs_europear

# Calculate Down-and-In put price
down_and_in_put_price = down_and_in_put(S0, K, r, sigma, T, barrier_level)
print("Reconsiling C: Analytical continuous down-and-in put barrier option p
```

```
Reconciling A: Black-Scholes European put option price: 5.537141658960508
Reconsiling C: Analytical continuous down-and-in put barrier option price:
5.786051176186103
```

Reconciling B

In [40]:

```
import numpy as np
import math
from datetime import datetime

# Parameters
S0 = 100
K = 100
sigma = 0.2
r = 0.025
valuation_date = "4/28/2023"
expiry_date = "12/15/2023"

valuation_datetime = datetime.strptime(valuation_date, "%m/%d/%Y")
expiry_datetime = datetime.strptime(expiry_date, "%m/%d/%Y")
T_days = (expiry_datetime - valuation_datetime).days
T = T_days / 365.0

# Trinomial tree parameters
```

```
N = 1000  # number of time steps
dt = T / N

# Trinomial probabilities
u = np.exp(sigma * np.sqrt(2 * dt))
d = 1 / u
p_u = ((np.exp(r * dt / 2) - np.exp(-sigma * np.sqrt(dt / 2))) / (np.exp(sig
p_d = ((np.exp(sigma * np.sqrt(dt / 2)) - np.exp(r * dt / 2)) / (np.exp(sigm
p_m = 1 - p_u - p_d

# Initialize the trinomial tree
tree = np.zeros((2 * N + 1, N + 1))

# Set up the terminal payoff
for i in range(2 * N + 1):
    tree[i, -1] = max(K - S0 * u ** (N - i), 0)

# Backward induction
for n in range(N - 1, -1, -1):
    for i in range(2 * n + 1):
        tree[i, n] = max(K - S0 * u ** (n - i), np.exp(-r * dt) * (p_u * tre

american_put_price_trinomial = tree[0, 0]
print("Trinomial tree American put option price:", american_put_price_trinom
```

```
Trinomial tree American put option price: 5.670929416020242
```

Task 5

Use a Monte Carlo pricer to reconcile with price from a.

In [43]:

```
# Parameters
S0 = 100
K = 100
sigma = 0.2
r = 0.025
valuation_date = "4/28/2023"
expiry_date = "12/15/2023"

valuation_datetime = datetime.strptime(valuation_date, "%m/%d/%Y")
expiry_datetime = datetime.strptime(expiry_date, "%m/%d/%Y")
T_days = (expiry_datetime - valuation_datetime).days
T = T_days / 365.0

# Monte Carlo parameters
num_simulations = 100000

# Generate random paths
np.random.seed(42)
Z = np.random.randn(num_simulations)
S_T = S0 * np.exp((r - 0.5 * sigma ** 2) * T + sigma * np.sqrt(T) * Z)

# Calculate the discounted payoffs
payoffs = np.maximum(K - S_T, 0) * np.exp(-r * T)
```

```
# Calculate the Monte Carlo price
monte_carlo_price = np.mean(payoffs)
print("Monte Carlo European put option price:", monte_carlo_price)
```

```
Monte Carlo European put option price: 5.5365699207100505
```

Why minor difference?

The minor differences in the option prices obtained from the PDE grid method, trinomial tree method, and Monte Carlo simulation, when compared to the analytical Black-Scholes prices, are normal and expected. These differences can be attributed to discretization errors, the choice of grid points, time steps, or the number of simulations in each method. It is common for these numerical methods to exhibit small differences in price when compared to analytical models.

In practice, increasing the numerical resolution (more grid points, time steps, or simulations) can help reduce these differences and improve the accuracy of the calculated prices. However, this improvement comes at the cost of increased computation time. Therefore, it is essential to strike a balance between accuracy and computational efficiency, especially in real-time pricing and risk management applications.