Numerical Method for Financial Engineering

Final Report Project

- Yu-Ching Liao ycliao3@illinois.edu
- Hyoung-Woo Hahm hwhahm2@illinois.edu

In the following tasks, we are using "Explicit" scheme to price.

Basic Import

In [1]:
```
import numpy as np
import math
from datetime import datetime
from scipy.stats import norm
import scipy.interpolate as interp
import scipy.stats as stats
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
```

Task A

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

Code with Necessary Comments and Result (Totally 4 pts)

In [2]:
```
# Parameters
S0 = 100  # Current asset price
K = 100  # Option strike price
sigma = 0.2  # Asset volatility
r = 0.025  # Risk-free interest rate
```

```
valuation_date = "4/28/2023"
expiry_date = "12/15/2023"

# Convert dates to datetime objects and calculate time to expiry in years
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 asset price direction
N = 1000  # Number of time steps

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

# 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). This represents the payoff of the put opti
V[-1, :-1] = 0  # Upper boundary (large asset price). The put option becomes
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):
        # Calculate the option price at each grid point using the explicit f
        # This approximates the Black-Scholes PDE, which describes the evolu
        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)  # Convectio
            - r * V[m, n+1]  # Discount term
        )

# Calculate the price of the put option at the current asset price and valua
put_price = V[M//2, 0]
print("European put option price by Explicit PDE grid:", put_price)
```

```
European put option price by Explicit PDE grid: 5.525524804683739
```

Description or Math Process for this task (4 pts)

This Python code calculates the price of a European put option using the finite difference method, specifically the explicit method. The governing equation is the Black-Scholes Partial Differential Equation (PDE). The solution process involves discretizing the PDE on a grid and using a time-stepping loop to solve the equation.

The Black-Scholes PDE for a European put option is given by:

$$\frac{\partial V}{\partial t} + rS\frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} - rV = 0 \tag{1}$$

where:

- $V(S,t)$ is the option price as a function of the asset price S and time t.
- r is the risk-free interest rate.
- σ is the asset price volatility.

The Explicit Finite Difference method is used to discretize and solve the PDE. The discretization process results in the following equation:

$$V_{m,n} = V_{m,n+1} + \Delta t \left[\frac{1}{2}\sigma^2 S_m^2 \frac{V_{m+1,n+1} - 2V_{m,n+1} + V_{m-1,n+1}}{\Delta S^2} + rS_m \frac{V_{m+1,n+1} - }{2\Delta} \right.$$
$$\left. - rV_{m,n+1} \right]$$

where $V_{m,n}$ represents the option price at grid point (m,n), and Δt and ΔS are the time and asset price step sizes, respectively.

Task B

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

Code with Necessary Comments and Result (Totally 4 pts)

In [3]:
```
# Parameters
S0 = 100  # Current asset price
K = 100  # Option strike price
sigma = 0.2  # Asset volatility
r = 0.025  # Risk-free interest rate
valuation_date = "4/28/2023"
expiry_date = "12/15/2023"

# Convert dates to datetime objects and calculate time to expiry in years
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 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):
        # Update the option price grid using the explicit finite difference
        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]
        )
        # Apply the early exercise condition for the American option
        V[m, n] = max(V[m, n], K - S[m])

#Compute the American put option price at the initial time and asset price S
american_put_price = V[M//2, 0]
print("American put option price by Explicit PDE grid:", american_put_price)
```

```
American put option price by Explicit PDE grid: 5.658390797709389
```

Description or Math Process for this task (4 pts)

This Python code calculates the price of an American put option using the finite difference method, specifically the explicit method. The governing equation is the Black-Scholes Partial Differential Equation (PDE), but with an additional early exercise condition that differentiates an American option from a European option. The solution process involves discretizing the PDE on a grid and using a time-stepping loop to solve the equation.

The Black-Scholes PDE for a European put option is given by:

$$\frac{\partial V}{\partial t} + rS\frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} - rV = 0 \tag{3}$$

For an American option, an early exercise condition is added, meaning that the value of the option is always at least the immediate exercise value:

$$V(S,t) \geq K - S \tag{4}$$

The Explicit Finite Difference method is used to discretize and solve the PDE. The discretization process results in the following equation:

$$V_{m,n} = \max\left(K - S_m,\ V_{m,n+1} + \Delta t\left[\frac{1}{2}\sigma^2 S_m^2 \frac{V_{m+1,n+1} - 2V_{m,n+1} + V_{m-1,n+1}}{\Delta S^2} + rS_m \frac{V_n}{}\right.\right.$$

where $V_{m,n}$ represents the option price at grid point (m, n), and Δt and ΔS are the time and asset price step sizes, respectively.

Task C

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.

Code with Necessary Comments and Result (Totally 4 pts)

In [4]:
```
# Parameters
S0 = 100  # Current asset price
K = 100  # Option strike price
sigma = 0.2  # Asset volatility
r = 0.025  # Risk-free interest rate
valuation_date = "4/28/2023"
expiry_date = "12/15/2023"
B = 80  # Barrier level

# Convert dates to datetime objects and calculate time to expiry in years
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 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
# For a down-and-in barrier option, the payoff at expiry is only if the asse
V[:, -1] = np.maximum(K - S, 0) * (S <= B)  # 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):
        # Update the option price grid using the explicit finite difference
        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]
        )

# Compute the down-and-in put option price at the initial time and asset pri
put_price = V[M//2, 0]
print("Continuous down-and-in put barrier option price:", put_price)
```

```
Continuous down-and-in put barrier option price: 2.1621694296734653
```

Description or Math Process for this task (4 pts)

This Python code calculates the price of a continuous down-and-in barrier put option using the finite difference method, specifically the explicit method. The governing equation is the Black-Scholes Partial Differential Equation (PDE). The solution process involves discretizing the PDE on a grid and using a time-stepping loop to solve the equation.

The Black-Scholes PDE for a European put option is given by:

$$\frac{\partial V}{\partial t} + rS\frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} - rV = 0 \tag{6}$$

For a continuous down-and-in barrier option, the payoff at expiry is only if the asset price has gone below a certain barrier level B during the option's life:

$$V(S,T) = \max(K - S, 0) \cdot I(S \leq B) \tag{7}$$

where I is the indicator function.

The Explicit Finite Difference method is used to discretize and solve the PDE. The discretization process results in the following equation:

$$V_{m,n} = V_{m,n+1} + \Delta t \left[\frac{1}{2}\sigma^2 S_m^2 \frac{V_{m+1,n+1} - 2V_{m,n+1} + V_{m-1,n+1}}{\Delta S^2} + rS_m \frac{V_{m+1,n+1} - }{2\Delta} \right.$$
$$\left. - rV_{m,n+1} \right]$$

where $V_{m,n}$ represents the option price at grid point (m, n), and Δt and ΔS are the time and asset price step sizes, respectively.

Task D

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 using BSM

Code with Necessary Comments and Result

In [5]:
```
# Define the Black-Scholes d1 and d2 parameters
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)

# Define the Black-Scholes formula for a European put option
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

# Convert valuation_date and expiry_date to datetime objects
valuation_datetime = datetime.strptime(valuation_date, "%m/%d/%Y")
expiry_datetime = datetime.strptime(expiry_date, "%m/%d/%Y")

# Calculate time to expiry in years
T_days = (expiry_datetime - valuation_datetime).days
T = T_days / 365.0

# Calculate and print the 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
```

```
Reconciling A: Black-Scholes European put option price: 5.537141658960508
```

Comment on the result:

This result is confirming the result that we obtain in Task A, which as well confirms the validity of using Explicit PDE grid to price European Put Option.

Description or Math Process for this task

This Python code calculates the price of a European put option using the Black-Scholes formula. The Black-Scholes formula is a solution to the Black-Scholes PDE, which is the fundamental equation in financial mathematics for option pricing. The parameters of the model are the spot price of the underlying asset, the strike price of the option, the risk-free interest rate, the volatility of the underlying asset, and the time to expiry of the option.

The Black-Scholes formula for a European put option is given by:

$$P(S, K, r, \sigma, T) = Ke^{-rT}N(-d_2) - SN(-d_1) \tag{9}$$

where

$$d_1 = \frac{\ln\left(\frac{S}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}},$$
$$d_2 = d_1 - \sigma\sqrt{T},$$

$N(\cdot)$ is the cumulative distribution function of the standard normal distribution, S is the spot price of the underlying asset, K is the strike price of the option, r is the risk-free interest rate, σ is the volatility of the underlying asset, and T is the time to expiry of the option.

Reconciling C using BSM

Code with Necessary Comments and Result

In [6]:
```
# Define the cumulative distribution function of the standard normal distrib
def N(x):
  return stats.norm.cdf(x)

# Define the analytical formula for a Down-and-In European put option
def closed_form_down_and_in_european_put_option(S, K, B, r, sigma, T):
    lambda_ = (r + ((sigma * sigma) / 2)) / (sigma * sigma)
    temp = 2 * lambda_ - 2.0
    x1 = (np.log(S / B) / (sigma * np.sqrt(T))) + (lambda_ * sigma * np.sqrt
    y = (np.log(B * B / (S * K)) / (sigma * np.sqrt(T))) + (lambda_ * sigma
    y1 = (np.log(B / S) / (sigma * np.sqrt(T))) + (lambda_ * sigma * np.sqrt
    return (-S * norm.cdf(-x1) + K * np.exp(-r * T) * norm.cdf(-x1 + sigma *
            S * pow(B / S, 2 * lambda_) * (norm.cdf(y) - norm.cdf(y1)) -
            K * np.exp(-r * T) * pow(B / S, temp) * (norm.cdf(y - sigma * np

# Calculate the Down-and-In put option price
down_and_in_put_price = closed_form_down_and_in_european_put_option(S0, K, B

# Print the price of the Down-and-In put option
print(f'Analytical solution for Continuous Down and In put :{down_and_in_put
```

```
Analytical solution for Continuous Down and In put :3.0188
```

Comment on the result:

This result is confirming the result that we obtain in Task C, which as well confirms the validity of using Explicit PDE grid to price Continuous Down and In Put Option.

Description or Math Process for this task

The Python code calculates the price of a Down-and-In European put option using an analytical solution. A Down-and-In put option becomes active, i.e., a standard put option, if the price of the underlying asset hits a pre-specified barrier level before the option's expiry. If the barrier is never breached, the option expires worthless.

The formula for a Down-and-In European put option is given by:

$$\begin{aligned} P(S, K, B, r, \sigma, T) = & -SN(-x_1) + Ke^{-rT}N(-x_1 + \sigma\sqrt{T}) \\ & + S\left(\frac{B}{S}\right)^{2\lambda}[N(y) - N(y1)] \\ & - Ke^{-rT}\left(\frac{B}{S}\right)^{2\lambda-2}\left[N(y - \sigma\sqrt{T}) - N(y1 - \sigma\sqrt{T})\right] \end{aligned}$$

where

$$\begin{aligned} \lambda &= \frac{r + \frac{\sigma^2}{2}}{\sigma^2}, \\ x_1 &= \frac{\ln\left(\frac{S}{B}\right) + \lambda\sigma\sqrt{T}}{\sigma\sqrt{T}}, \\ y &= \frac{\ln\left(\frac{B^2}{SK}\right) + \lambda\sigma\sqrt{T}}{\sigma\sqrt{T}}, \\ y1 &= \frac{\ln\left(\frac{B}{S}\right) + \lambda\sigma\sqrt{T}}{\sigma\sqrt{T}}, \end{aligned}$$

$N(\cdot)$ is the cumulative distribution function of the standard normal distribution,
S is the spot price of the underlying asset,
K is the strike price of the option,
B is the barrier level,
r is the risk-free interest rate,
σ is the volatility of the underlying asset,
T is the time to expiry of the option.

Reconciling B with Trinomial Tree Model

Code with Necessary Comments and Result

In [7]:
```
# 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

# Print the price of the American put option
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
```

Comment on the result:

This result is confirming the result that we obtain in Task CB, which as well confirms the validity of using Explicit PDE grid to price American Put Option.

Description or Math Process for this task

The trinomial tree is a discrete-time model for the dynamics of the underlying asset price. At each time step, the asset price can either move up by a factor of u, move down by a factor of d, or stay the same. These movements occur with respective probabilities p_u, p_d, and p_m.

The trinomial probabilities are calculated as follows:

$$p_u = \left(\frac{e^{\frac{r \cdot dt}{2}} - e^{-\sigma \cdot \sqrt{\frac{dt}{2}}}}{e^{\sigma \cdot \sqrt{\frac{dt}{2}}} - e^{-\sigma \cdot \sqrt{\frac{dt}{2}}}} \right)^2$$

$$p_d = \left(\frac{e^{\sigma \cdot \sqrt{\frac{dt}{2}}} - e^{\frac{r \cdot dt}{2}}}{e^{\sigma \cdot \sqrt{\frac{dt}{2}}} - e^{-\sigma \cdot \sqrt{\frac{dt}{2}}}} \right)^2$$

$$p_m = 1 - p_u - p_d$$

The price of the option at each node in the trinomial tree is calculated as the maximum of the intrinsic value and the discounted expected value of the option at the next time step:

$$V_{i,n} = \max\left(K - S0 \times u^{(n-i)}, e^{-r \cdot dt} \times (p_u \cdot V_{i,n+1} + p_m \cdot V_{i+1,n+1} + p_d \cdot V_{i+2,n+1})\right)$$

The price of the American put option is then given by the value of the option at the root of the trinomial tree.

Task E

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

Code with Necessary Comments and Result (Totally 4 pts)

In [8]:
```
# 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

# Set the parameters for the Monte Carlo simulation
num_simulations = 100000

# Generate standard normal random variables
np.random.seed(42)
Z = np.random.randn(num_simulations)

# Simulate asset prices at expiry
S_T = S0 * np.exp((r - 0.5 * sigma ** 2) * T + sigma * np.sqrt(T) * Z)

# Calculate the payoffs for each simulation, discounted to the present
payoffs = np.maximum(K - S_T, 0) * np.exp(-r * T)

# Calculate the Monte Carlo price as the average of the payoffs
```

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

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

Comment on the result:

This result is confirming the result that we obtain in Task A, which as well confirms the validity of using Explicit PDE grid to price European Put Option.

Description or Math Process for this task (4 pts)

This Python code calculates the price of a European put option using the Monte Carlo simulation method. The details of the method are explained below:

The Monte Carlo price for a European put option is given by the discounted expectation of the payoff under the risk-neutral measure:

$$P_{MC}(S, K, r, \sigma, T) = e^{-rT}\mathbb{E}_{\mathbb{Q}}[(K - S_T)^+] \tag{10}$$

where

- $(\cdot)^+ = \max(\cdot, 0)$ denotes the positive part,
- $S_T = Se^{(r-\frac{1}{2}\sigma^2)T+\sigma\sqrt{T}Z}$ is the asset price at expiry,
- Z is a standard normal random variable,
- S is the spot price of the underlying asset,
- K is the strike price of the option,
- r is the risk-free interest rate,
- σ is the volatility of the underlying asset, and
- T is the time to expiry of the option.

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.


ILLINOIS