```
import numpy as np
import matplotlib.pyplot as plt
from scipy import sparse
from scipy.stats import norm
```

Consider parabolic partial differential equation (PDE) with the following form:

$$\frac{\partial V(s,t)}{\partial t} + a(s,t)\frac{\partial^2 V(s,t)}{\partial s^2} + b(s,t)\frac{\partial V(s,t)}{\partial s} + c(s,t)V(s,t) = 0$$

Let $\Delta s = \frac{s_{max}}{M}$ and $\Delta t = \frac{T}{N}$ be the sizes of space step and time step. Define grid point $V_m^n := V(s = m\Delta s, t = T - n\Delta t), \forall (m,n) \in \{0,1,2,\dots,M\} \times \{0,1,2,\dots,N\}$.

Explicit method

Apply the explicit finite-difference method to the PDE:

$$-\frac{V_m^{n+1} - V_m^n}{\Delta t} + a_m^n \frac{V_{m+1}^n - 2V_m^n + V_{m-1}^n}{(\Delta s)^2} + b_m^n \frac{V_{m+1}^n - V_{m-1}^n}{2\Delta s} + c_m^n V_m^n = 0$$

Rearrange the terms:

$$\begin{aligned} V_m^{n+1} &= \left(a_m^n \frac{\Delta t}{(\Delta s)^2} - b_m^n \frac{\Delta t}{2\Delta s}\right) V_{m-1}^n + \left(1 - 2a_m^n \frac{\Delta t}{(\Delta s)^2} + c_m^n \Delta t\right) V_m^n + \left(a_m^n \frac{\Delta t}{(\Delta s)^2} + b_m^n \frac{\Delta t}{2\Delta s}\right) V_{m+1}^n \\ &= \alpha_m^n V_{m-1}^n + \beta_m^n V_m^n + \gamma_m^n V_{m+1}^n \end{aligned}$$

Represent the linear system in matrix form:

$$\begin{pmatrix} V_1^{n+1} \\ V_2^{n+1} \\ \vdots \\ V_{M-2}^{n+1} \\ V_{M-1}^{n+1} \end{pmatrix} = \begin{pmatrix} \beta_1^n & \gamma_1^n & 0 & \cdots & 0 \\ \alpha_2^n & \beta_2^n & \gamma_2^n & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & 0 & \alpha_{M-2}^n & \beta_{M-2}^n & \gamma_{M-2}^n \\ 0 & 0 & 0 & \alpha_{M-1}^n & \beta_{M-1}^n \end{pmatrix} \begin{pmatrix} V_1^n \\ V_2^n \\ \vdots \\ V_{M-2}^n \\ V_{M-1}^n \end{pmatrix} + \begin{pmatrix} \alpha_1^n V_0^n \\ 0 \\ \vdots \\ 0 \\ \gamma_{M-1}^n V_M^n \end{pmatrix}$$

The explicit finite-difference method suffers from restrictions in the size of the grid step Δs. We assume $\Delta t \leqslant \frac{\Delta s}{2a}$ to ensure the convergence and numerical stability of the method.

Example: European call option under the Black-Scholes economy

The Black-Scholes PDE:

$$\frac{\partial V(s,t)}{\partial t} + \frac{1}{2}\sigma^2 s^2 \frac{\partial^2 V(s,t)}{\partial s^2} + rs\frac{\partial V(s,t)}{\partial s} - rV(s,t) = 0$$

Terminal and boundary conditions for European call option with strike price K:

$$\begin{aligned} V(s,T) &= \max(S - K, 0) \\ \lim_{s \to 0^-} V(s,t) &= 0 \\ \lim_{s \to \infty} V(s,t) &= s - Ke^{-r(T-t)} \end{aligned}$$

```
# parameters
stock_price = 100.0
strike_price = 110.0
volatility = 0.2
risk_free_rate = 0.05
time_to_maturity = 1.0


# BS formula
def black_scholes(S, K, r, T, sigma, option_type):
    d1 = (np.log(S/K) + (r + 0.5*sigma**2)*T) / (sigma*T**0.5)
    d2 = d1 - sigma*T**0.5
    Q1 = norm.cdf(d1)
    Q2 = norm.cdf(d2)

    if option_type == 'call':
        option_price = S*Q1 - K*np.exp(-r * T)*Q2
    elif option_type == 'put':
        option_price = K*np.exp(-r * T)*(1-Q2) - S*(1-Q1)

    return option_price


call_BS = black_scholes(stock_price, strike_price, risk_free_rate,
                                time_to_maturity, volatility, 'call')

print(f'Call option price (BS formula):{call_BS:.4f}')
```

```
Call option price (BS formula):6.0401
```

```
# BS PDE
a = lambda s : 0.5 * s**2 * volatility**2
b = lambda s : risk_free_rate * s
c = lambda s : -risk_free_rate


# Grid Parameters
s_max = 4 * strike_price                                    # space upper boundary
M = int(s_max) + 1                                          # the number of space steps
N = 10000                                                   # the number of time steps


# Grid Construction
arr_s, ds = np.linspace(0, s_max, M, retstep=True)          # space discretization
arr_t, dt = np.linspace(0, time_to_maturity, N, retstep=True)  # time discretization
if 2*a(s_max)*dt > ds:
    print(f'the condition fails')
V = np.zeros((M, N))                                        # grid initialization
V[:,0] = np.maximum(0, arr_s-strike_price)                  # European call option payoff (terminal condition)
```

```
V[0,:] = 0                                                          # boundary condition
V[-1,:] = s_max - strike_price * np.exp(-risk_free_rate*arr_t)      # boundary condition


# Tri-diagonal Matrix Construction
alpha = lambda s : a(s)*dt/ds**2 - b(s)*dt/2/ds
beta = lambda s : 1 - 2*a(s)*dt/ds**2 + c(s)*dt
gamma = lambda s : a(s)*dt/ds**2 + b(s)*dt/2/ds

D = np.zeros(shape = (M-2,M-2))
D[0,0], D[0,1], D[-1,-2], D[-1,-1] = beta(arr_s[1]), gamma(arr_s[1]), alpha(arr_s[-2]), beta(arr_s[-2])
for m in range(1, M-3):
    D[m,m-1], D[m,m], D[m,m+1] = alpha(arr_s[m+1]), beta(arr_s[m+1]), gamma(arr_s[m+1])


# Grid Computation
rem = np.zeros(shape = (M-2,))
for n in range(1,N):
    rem[0], rem[-1] = alpha(arr_s[1])*V[0,n-1], gamma(arr_s[-2])*V[-1,n-1]
    V[1:-1,n] =  D @ V[1:-1, n-1] + rem


print(f'Call option price (explicit method): {V[int(stock_price/ds), -1]:.4f}')
```

```
Call option price (explicit method): 6.0381
```

Implicit method

Apply the implicit finite-difference method to the PDE:

$$-\frac{V_m^{n+1}-V_m^n}{\Delta t}+a_m^{n+1}\frac{V_{m+1}^{n+1}-2V_m^{n+1}+V_{m-1}^{n+1}}{(\Delta s)^2}+b_m^{n+1}\frac{V_{m+1}^{n+1}-V_{m-1}^{n+1}}{2\Delta s}+c_m^{n+1}V_m^{n+1}=0$$

Rearrange the terms:

$$\begin{aligned}V_m^n &= \left(-a_m^{n+1}\frac{\Delta t}{(\Delta s)^2}+b_m^{n+1}\frac{\Delta t}{2\Delta s}\right)V_{m-1}^{n+1}+\left(1+2a_m^{n+1}\frac{\Delta t}{(\Delta s)^2}-c_m^{n+1}\Delta t\right)V_m^{n+1}+\left(-a_m^{n+1}\frac{\Delta t}{(\Delta s)^2}-b_m^{n+1}\frac{\Delta t}{2\Delta s}\right)V_{m+1}^{n+1}\\ &= \alpha_m^{n+1}V_{m-1}^{n+1}+\beta_m^{n+1}V_m^{n+1}+\gamma_m^{n+1}V_{m+1}^{n+1}\end{aligned}$$

Represent the linear system in matrix form:

$$\begin{pmatrix}V_1^n\\V_2^n\\\vdots\\V_{M-2}^n\\V_{M-1}^n\end{pmatrix}=\begin{pmatrix}\beta_1^{n+1}&\gamma_1^{n+1}&0&\cdots&0\\\alpha_2^{n+1}&\beta_2^{n+1}&\gamma_2^{n+1}&0&0\\0&\ddots&\ddots&\ddots&0\\\vdots&0&\alpha_{M-2}^{n+1}&\beta_{M-2}^{n+1}&\gamma_{M-2}^{n+1}\\0&0&0&\alpha_{M-1}^{n+1}&\beta_{M-1}^{n+1}\end{pmatrix}\begin{pmatrix}V_1^{n+1}\\V_2^{n+1}\\\vdots\\V_{M-2}^{n+1}\\V_{M-1}^{n+1}\end{pmatrix}+\begin{pmatrix}\alpha_1^{n+1}V_0^{n+1}\\0\\\vdots\\0\\\gamma_{M-1}^{n+1}V_M^{n+1}\end{pmatrix}$$

```
# Tri-diagonal Matrix Construction
alpha = lambda s : -a(s)*dt/ds**2 + b(s)*dt/2/ds
beta = lambda s : 1 + 2*a(s)*dt/ds**2 - c(s)*dt
gamma = lambda s : -a(s)*dt/ds**2 - b(s)*dt/2/ds

D = np.zeros(shape = (M-2,M-2))
D[0,0], D[0,1], D[-1,-2], D[-1,-1] = beta(arr_s[1]), gamma(arr_s[1]), alpha(arr_s[-2]), beta(arr_s[-2])
for m in range(1, M-3):
    D[m,m-1], D[m,m], D[m,m+1] = alpha(arr_s[m+1]), beta(arr_s[m+1]), gamma(arr_s[m+1])
D = sparse.csr_matrix(D)


# Grid Computation
rem = np.zeros(shape = (M-2,))
for n in range(1,N):
    rem[0], rem[-1] = alpha(arr_s[1])*V[0,n-1], gamma(arr_s[-2])*V[-1,n-1]
    V[1:-1,n] =  sparse.linalg.spsolve(D, V[1:-1, n-1] - rem)

print(f'Call option price (implicit method): {V[int(stock_price/ds), -1]:.4f}')
```

```
Call option price (implicit method): 6.0380
```