Finite difference method for a European option

Stochastic Optimal Control Problems & Application in finance
Hasnaa.Zidani@insa-rouen.fr February 7th, 2023

In the sequel, some hints for the implementation in Python are given. However, you can use other programming languages (`c`*,* `c++`*,* `matlab`*,* `octave`*,* `scilab`*).*

Consider the European put function $v = v(t,s)$, $t \in [0,T]$, $s \in [0, X_{\max}]$. It satisfies the Black and Scholes backward PDE on the truncated domain $\Omega = [X_{\min}, X_{\max}]$:

$$\begin{cases} \frac{\partial}{\partial t}v - \frac{\sigma^2}{2}s^2\ \frac{\partial^2}{\partial s^2}v - rs\ \frac{\partial}{\partial s}v + rv = 0, \quad t \in (0,T),\ s \in (X_{\min}, X_{\max}) \\ v(t, X_{\min}) = v_\ell(t) \equiv Ke^{-rt} - X_{\min}, \quad t \in (0,T) \\ v(t, X_{\max}) = v_r(t) \equiv 0, \quad t \in (0,T) \\ v(0,s) = \varphi(s) := (K-s)_+, \quad s \in (X_{\min}, X_{\max}). \end{cases} \tag{1}$$

In the numerical simulations, we will consider the following parameters:

$$K = 100, \quad X_{\min} = 0,\ X_{\max} = 200, \quad T = 1, \quad \sigma = 0.2, \quad r = 0.1$$

The aim of this session is to compute an approximation of $v(t,s)$ at final time $t = T$.

1 The Euler Forward (or Explicit Euler) scheme

Let $h := \frac{X_{\max} - X_{\min}}{I+1}$ and $\Delta t := \frac{T}{N}$ be the time step and spatial mesh step, and

$$\begin{aligned} &s_j := X_{\min} + jh, \quad j = 0, \ldots, I+1 \text{ (mesh points)} \\ &t_n = n\Delta t, \quad n = 0, \ldots, N \text{ (time mesh)} \end{aligned}$$

We are interested in computing an approximation U_j^n of $v(t_n, s_j)$. for $n = 1, \cdots N$ and $j = 1, \cdots I$.

The "Euler Forward scheme" (or Explicit Euler scheme), hereafter denoted "EE", is given as follows:

$$\begin{aligned} &\frac{U_j^{n+1} - U_j^n}{\Delta t} + \frac{\sigma^2}{2}s_j^2\ \frac{-U_{j-1}^n + 2U_j^n - U_{j+1}^n}{h^2} - rs_j\frac{U_{j+1}^n - U_{j-1}^n}{2h} + rU_j^n = 0 \quad \begin{matrix} n = 0, \ldots, N-1, \\ j = 1, \ldots, I \end{matrix} \\ &U_0^n = v_\ell(t_n) \equiv Ke^{-rt_n} - X_{\min}, \quad n = 0, \ldots, N \\ &U_{I+1}^n = v_r(t_n) \equiv 0, \quad n = 0, \ldots, N \\ &U_j^0 = \varphi(s_j) \equiv (K - s_j)_+, \quad j = 1, \ldots, I \end{aligned} \tag{2}$$

Notice that the indices j vary between 1 and I . For $j = 1$, the scheme uses the value $U_0^n := v_\ell(t_n)$ (left boundary value), and for $j = I$, the scheme uses the value $U_{I+1}^n := v_r(t_n)$ (right boundary value).

1.1 Implementation of Euler Explicit (EE). Some preliminaries.

Consider the vector $U^n = \left(U_1^n \cdots U_I^n\right)^\mathsf{T}$. With this notation, the scheme (2) can be re-written as

$$\frac{U^{n+1} - U^n}{\Delta t} + QU^n + q(t_n) = 0, \quad n = 0, \ldots, N-1 \tag{3}$$

$$U^0 = (\varphi(s_i))_{1 \le i \le I} \tag{4}$$

where Q is a square matrix of dimension I and $q(t_n)$ is a column vector of size I.
Let us denote

$$\alpha_j := \frac{\sigma^2}{2}\frac{s_j^2}{h^2}, \quad \beta_j := r\frac{s_j}{2h}.$$

From (2), one can check that Q is a tridiagonal matrix given by

$$Q := \begin{bmatrix} 2\alpha_1 + r & -\alpha_1 - \beta_1 & & & & 0 \\ -\alpha_2 + \beta_2 & 2\alpha_2 + r & -\alpha_2 - \beta_2 & & & \\ & \ddots & \ddots & \ddots & & \\ & & -\alpha_i + \beta_i & 2\alpha_i + r & -\alpha_i - \beta_i & \\ & & \ddots & \ddots & \ddots & \\ 0 & & & & -\alpha_I + \beta_I & 2\alpha_I + r \end{bmatrix}$$

and $q(t_n)$ contains the boundary values $U_0^n = v_\ell(t_n)$ and $U_{I+1}^n = v_r(t_n)$ as follows

$$q(t_n) := \begin{pmatrix} (-\alpha_1 + \beta_1)U_0^n \\ 0 \\ \vdots \\ 0 \\ (-\alpha_I - \beta_I)U_{I+1}^n \end{pmatrix} \equiv \begin{pmatrix} (-\alpha_1 + \beta_1)v_\ell(t_n) \\ 0 \\ \vdots \\ 0 \\ (-\alpha_I - \beta_I)v_r(t_n) \end{pmatrix}.$$

Then the EE scheme can be implemented with the initialization $U^0 = (\varphi(s_i))$ and the following recursion

$$U^{n+1} = (Id - \Delta t Q)U^n - \Delta t q(t_n), \quad n = 0, \ldots, N-1.$$

1.2 Notations. Some hints for Python

Financial parameters should be denoted `r,sigma,K,T` and should be defined at the beginning of the program along with the following functions and parameters

- payoff function `u0` (for φ) as a function of s.
- functions `uleft` (for v_ℓ) and `uright` (for v_r), as functions of the time
- parameters `Smin`, `Smax` for the domain boundary

- parameter `Sval` (a specific value `Sval`$= \bar{s}$ where we want to evaluate $v(T, \bar{s})$).

We advice to print a part of the data to check the simulation's setting:

```
print('N=%3i' % N,  'I=%3i' % I,  'SCHEME=%s' % SCHEME)
```

Vectors of $\mathbb{R}^I$ can be represented as matrices of size $I \times 1$ (i.e. , elements of type `np.array`). The mesh grid is given by

```
s=Xmin + h*np.arange(1:I+1);  # s[0],...,s[I-1] encodes s_1,...,s_I
```

The payoff function `phi` can be implemented by the formula

```
def phi(s):
  return np.maximum(K-s,0).reshape(I,1)
```

To implement he matrix `Q` ($I \times I$) and the function `q(t)`, it is strongly recommended to use the following types with module `numpy`:

```
import numpy as np
Q=np.zeros((I,I))
# fill Q correctly
# ...

def q(t):
  y=np.zeros((I,1))
  # fill y correctly
  # ...
  return y
```

The initialization and recursion steps can be implemented as follows

```
  # init
  U=phi(s)

  # main loop
  for n in range(0:N):
    t=n*dt
    U = (Id-dt*Q) @ U - dt* q(t)
    # ... plots, prints, etc.
```

Once the scheme is well implemented, you should be able to obtain the following typical result as shown in Figure 1.

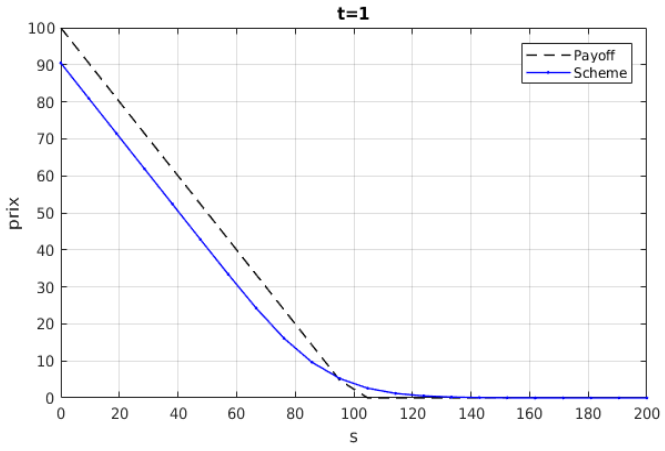

Figure 1: European put option. $T = 1$, $N = I = 20$.

1.3 First numerical tests

a) Test the (EE) scheme for $N = 10$ and $I = 10, 20, 50, \ldots$, and for $N = I = 10, 20, 50, 100$. Observe that:

- the scheme is not always numerically stable (the norm of U^n may explode after a finite number of iterations)
- it does not always give a positive solution (i.e. we do not always have $U_j^n \geq 0 \ \forall n, j$)

b) In order to understand the origin of the oscillations or explosion when they occur, for instance, fix $N = 10$ and $I = 50$, and consider the "amplification" matrix defined as

$$B := I_d - \Delta t Q.$$

Check that the coefficients of B are not all positive [1] and that they may have a modulus greater than 1. Compute also the norms $\|B\|_\infty$ and $\|B\|_2$ (see at the end of the document for python commands) and check that they are larger than one.

On the contrary, check that for $N = I = 10$, coefficients of B are (almost) all positive, and smaller that 1.

c) For the same previous values $(N, I) = (10, 10)$ or $(10, 50)$, compute the CFL number

$$\mu := \frac{\Delta t}{h^2} \, \sigma^2 X_{\max}^2$$

and print it. Check that when μ is sufficiently small, the scheme is stable.

[1] For a scheme that would be simply written $U^{n+1} = BU^n$ (ie with zero boundary conditions), positivity of the matrix B matrix ($B \geq 0$ compentwise) ensures that if $U^0 \geq 0$ (componentwise) then $U^n \geq 0$ for all n.

d) Compute the P1-interpolated value at $\bar{s} = 90$ ($\bar{s}$ may not be a grid point !). if $\bar{s} \in [s_i, s_{i+1}]$, then the $P1$-interpolated is given by $U(\bar{s}) \simeq \dfrac{s_{i+1} - \bar{s}}{h} U_i + \dfrac{\bar{s} - s_i}{h} U_{i+1}$.

e) Numerical order of the scheme *(The exact value is given in Exercise 2)*

First consider the following values of I and of N: $N = I = 10, 20, 40, 80, 160, 320$, and complete the corresponding Table 1. Then consider $I = 10, 20, 40, 80, ...$, and $N = I^2/10$ and complete the corresponding error columns in Table 2.

The CFL condition requires N to be or order I^2, which makes the (EE) scheme very costly in terms of number of operations. In the sequel, we will turn our attention to the implicit scheme.

2 Implicit Euler (IE) scheme

The Implicit Euler scheme ("IE" scheme), with centered difference approximation for the first spatial derivative, is

$$\begin{aligned}
&\frac{U_j^{n+1} - U_j^n}{\Delta t} + \frac{\sigma^2}{2} s_j^2 \, \frac{-U_{j-1}^{n+1} + 2U_j^{n+1} - U_{j+1}^{n+1}}{h^2} - r s_j \frac{U_{j+1}^{n+1} - U_{j-1}^{n+1}}{2h} + r U_j^{n+1} = 0 \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad n = 0, \ldots, N-1, \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad j = 1, \ldots, I \\
&U_0^{n+1} = v_\ell(t_{n+1}) \equiv K e^{-r t_{n+1}} - X_{\min}, \quad n = 0, \ldots, N-1 \qquad (5) \\
&U_{I+1}^{n+1} = v_r(t_{n+1}) \equiv 0, \quad n = 0, \ldots, N-1 \\
&U_j^0 = (K - s_j)_-, \quad j = 1, \ldots, I
\end{aligned}$$

Check that the scheme, in vector form, can be written as

$$\frac{U^{n+1} - U^n}{\Delta t} + Q U^{n+1} + q(t_{n+1}) = 0, \quad n = 0, \ldots, N-1$$

with $U^0 = (\varphi(s_i))_{1 \le i \le I}$.

Implement (IE) scheme. By setting the parameter `SCHEMA='IE'` at the begining of the **same** program, the code should switch to the IE method.

Linear systems in the form $Ax = b$ may be solved by `x=nlp.solve(A,b)`

a) Check that with the IE scheme there is no stability issues.

b) Complete Table 3&4 with $N = I$ and with $N = I/10$.

3 Crank-Nicolson scheme

The Crank Nicholson scheme ("CN" scheme), with centered difference approximation for the first spatial derivative, is:

$$\frac{U_j^{n+1}-U_j^n}{\Delta t}+\frac{1}{2}\left(-\frac{\sigma^2}{2}s_j^2\,\frac{U_{j-1}^{n+1}-2U_j^{n+1}+U_{j+1}^{n+1}}{h^2}-rs_j\frac{U_{j+1}^{n+1}-U_{j-1}^{n+1}}{2h}+rU_j^{n+1}\right)$$
$$+\frac{1}{2}\left(-\frac{\sigma^2}{2}s_j^2\,\frac{U_{j-1}^{n}-2U_j^{n}+U_{j+1}^{n}}{h^2}-rs_j\frac{U_{j+1}^{n}-U_{j-1}^{n}}{2h}+rU_j^{n}\right)=0 \qquad (6)$$
$$n=0,\dots,N-1,$$
$$j=1,\dots,I$$

with adequate boundary conditions and initial conditions, such as:

$$\begin{aligned} &U_0^{n+1}=v_\ell(t_{n+1})\equiv Ke^{-rt_{n+1}}-X_{\min},\quad n=0,\dots,N-1 \qquad (7)\\ &U_{I+1}^{n+1}=v_r(t_{n+1})\equiv 0,\quad n=0,\dots,N-1\\ &U_j^0=(K-s_j)_-,\quad j=1,\dots,I \end{aligned}$$

a) Implement the Crank-Nicolson scheme (CN) (`SCHEME='CN'`). First, write the CN scheme in vector form.

b) Complete the corresponding table with $N=I$ and $N=I/10$.

4 Exercices

Exercice. 1 Improved efficiency using sparse matrices. *The matrix Q has only a few non zero elements (about $3I$ non zero elements). It is possible to code only the non-zero elements of Q by using a sparse type matrix. Typical modules:*

```
from scipy.sparse import csr_matrix as sparse
from scipy.sparse.linalg import spsolve
```

Warning: after the use of a command such as `x=spsolve(B, b)` *for solving* `B*x=b`*, it is recommended to reshape the result (*`x=x.reshape(I,1)`*).*

$\Rightarrow$ *Modify slightly your code in order to program EE, IE and CN with sparse matrices. Compare the speed of the new code with respect to the full matrix approach. (Execution time is in general improved for large N, I).*

Exercice. 2 ("Black and Scholes formula") *Program the Black and Scholes formula for the put option. A formula corresponding to $v(t,S)$ is*

$$v(t,S):=Ke^{-rt}N(-d_-)-SN(-d_+)$$

with

$$d_\pm:=\frac{log(S/K)+(r\pm\frac{1}{2}\sigma^2)t}{\sqrt{\sigma^2 t}}\quad \textit{and}\quad N(y):=\int_{-\infty}^{y}e^{-u^2/2}\frac{du}{\sqrt{2\pi}} \qquad (8)$$

5 Useful modules and commands

```
import numpy as np                # array
import numpy.linalg as lng        # linear algebra
import matplotlib.pyplot as plt   # plot functions
import time
import sys                        # command sys.exit()
```

To obtain the cputime for a sequence of instructions, use:

```
t0=time.time()
# instructions ...
t1=time.time(); print('tcpu=%5.2f' % (t1-t0))
```

Miscellaneous : for a matrix A of type `np.array`:

`print(A)`	Nice print of an nd.array `A`
`print(np.round(A,decimals=3))`	
`lng.norm(A,np.inf)`	$\|A\|_\infty$ $(= \max_{x\neq 0} \|Ax\|_\infty / \|x\|_\infty)$
`lng.norm(A,2)`	$\|A\|_2$ (largest singular value)

For computing the cumulative normal distribution function of a scalar or np.array `x`:

```
import scipy.stats as stats
stats.norm.cdf(x, 0.0, 1.0)
```

6 Tables

I	N	$U(\bar{s})$	e_k	order α_k
10	10			–
20	20			
40	40			
80	80			
160	160			

Table 1: EE scheme with $N = I$

I	N	$U(\bar{s})$	e_k	order α_k
10	10			–
20	40			
40	160			
80	640			
160	2540			
320	10240			

Table 2: EE scheme with $N = I^2/10$

I	N	$U(\bar{s})$	e_k	order α_k
10	10			–
20	20			
40	40			
80	80			
160	160			
320	320			

Table 3: Implicit Euler (IE) scheme with $N = I$

I	N	$U(\bar{s})$	e_k	order α_k
10	1			–
20	2			
40	4			
80	8			
160	16			
320	32			

Table 4: Implicit Euler scheme with $N = I/10$

I	N	$U(\bar{s})$	e_k	order α_k
10	10			–
20	20			
40	40			
80	80			
160	160			
320	320			

Table 5: Crank-Nicolson (CN) scheme with $N = I$

I	N	$U(\bar{s})$	e_k	order α_k
10	1			–
20	2			
40	4			
80	8			
160	16			
320	32			

Table 6: Crank-Nicolson (CN) scheme with $N = I/10$