NUMERICAL METHODS FOR PDE IN FINANCE - M2MO - U. PARIS CITÉ

American options

O. Bokanowski

Plan of work : follow sections 1-5 and write a short report about your results.

We look for a numerical approximation of the american put option $v = v(t,s)$, $t \in (0,T)$ and $s \in \Omega := (S_{min}, S_{max})$, solution of the following Partial Differential Equation:

$$\min(\partial_t v + \mathcal{A}v,\ v - \varphi) = 0, \quad (t,s) \in (0,T) \times \Omega, \tag{1a}$$ {eq:1a}

$$v(t, S_{min}) = v_\ell(t), \quad t \in (0,T), \tag{1b}$$ {eq:1b_g}

$$v(t, S_{max}) = v_r(t) \equiv 0, \quad t \in (0,T), \tag{1c}$$ {eq:1b_d}

$$v(0,s) = \varphi(s), \quad s \in \Omega \tag{1d}$$ {eq:1c}

with

$$\mathcal{A}v := -\frac{\sigma^2}{2} s^2\ \partial_{s,s} v - rs\ \partial_s v + rv,$$

and σ, r, K are strictly positive constants. A logical choice for the left boundary condition $v_\ell(t)$ is

$$v_\ell(t) :\equiv \varphi(S_{min}) = K - S_{min}.$$

For the numerical tests we will chose the following financial parameters:

$$K = 100.0,\ T = 1,\ \underline{\sigma = 0.3},\ \text{and } r = 0.1 \tag{2}$$

and we set the computational domain $\Omega = (S_{min}, S_{max})$ as follows:

$$S_{min} = 50,\ S_{max} = 250. \tag{3}$$

We will consider mainly the following payoff function $\varphi = \varphi_1$ for the model of the american put option: (`payoff=1`):

$$\varphi_1(s) := (K - s)_+, \quad \text{together with } v_\ell(t) := K - S_{min}.$$

Note that looking for $v(t,s) \simeq a(t)s + b(t)$ for $s \simeq 0$ in (1a) leads to $v(t,s) = K - s$, hence the above definition of the left boundary condition for v_ℓ. (An other barrier payoff function[1] will be also used for testing the Brennan-Schwartz algorithm.)

Finally, we aim to compute the value $\bar{v} := v(T, S_{val})$ at

$$S_{val} = 90.0. \tag{4}$$

The exact value of $\bar{v}$ is not known.[2]

[1] Barrier payoff function - `"payoff=2"`:

$$\varphi_2(s) := \left\{ \begin{array}{ll} K & \text{for } \frac{K}{2} \le s \le K \\ 0 & \text{otherwise} \end{array} \right\}, \quad \text{with } v_\ell(t) = v_r(t) = 0.$$

[2] Using a BDF scheme of second order, with centered approximation, with parameters $(I, N) = (5000, 500)$, gives the following approximation: $\bar{v} \simeq 13.12055$

1 Explicit Euler Scheme (or "Euler Forward Scheme")

Notations. We adopt the usual notations : mesh $s_j = S_{min} + jh$, $j = 1, \dots, I$, $h = (S_{max} - S_{min})/(I+1)$ (so that $s_0 = S_{min}$ and $s_{I+1} = S_{max}$), and $t_n = n\Delta t$, $0 \le n \le N$, $\Delta t = T/N$. We then look for U_j^n, an approximation of $v(t_n, s_j)$. We choose to work with the unknown vector of $\mathbb{R}^I$:

$$U^n := \begin{pmatrix} U_1^n \\ \vdots \\ U_I^n \end{pmatrix}.$$

Euler Forward scheme, or "Explicit Euler" (EE) scheme, using the centered approximation, is the following scheme :

$$\begin{cases} \min\Big(\dfrac{U_j^{n+1} - U_j^n}{\Delta t} + \dfrac{\sigma^2}{2} s_j^2 \dfrac{-U_{j-1}^n + 2U_j^n - U_{j+1}^n}{h^2} - r s_j \dfrac{U_{j+1}^n - U_{j-1}^n}{2h} + rU_j^n, \\ \qquad\qquad U_j^{n+1} - \varphi(s_j)\Big) = 0, \qquad\qquad 1 \le j \le I, \\ U_0^n = v_\ell(t_n), \\ U_{I+1}^n = v_r(t_n), \end{cases}$$

for $n = 0, \dots, N-1$. The scheme is initialized with $U_j^0 = \varphi(s_j)$. We denote by A the discretization matrix associated to the operator $\mathcal{A}$, of size I, and $q(t) \in \mathbb{R}^I$, such that

$$\begin{aligned} (AU + q(t))_j \quad &:= \quad +\frac{\sigma^2}{2} s_j^2 \frac{-U_{j-1} + 2U_j - U_{j+1}}{h^2} - r s_j \frac{U_{j+1} - U_{j-1}}{2h} + rU_j, \quad 1 \le j \le I. \\ &= \quad -(\alpha_j - \beta_j)U_{j-1} + (2\alpha_j + r)U_j - (\alpha_j + \beta_j)U_{j+1}, \quad 1 \le j \le I. \end{aligned}$$

with $\alpha_j = \frac{\sigma^2}{2}\frac{s_j^2}{h^2}$ and $\beta_j = \frac{r s_j}{2h}$. We recall that A is the tridiagonal matrix

$$tridiag\big(-(\alpha_j - \beta_j),\ 2\alpha_j + r,\ -(\alpha_j + \beta_j)\big),$$

and

$$q(t) \quad := \quad \begin{pmatrix} (-\alpha_1 + \beta_1) v_\ell(t) \\ 0 \\ \vdots \\ 0 \\ (-\alpha_I - \beta_I) v_r(t) \end{pmatrix}.$$

This matrix A and vector $q(t)$ are the same as the one used for European options.

Let also g be the vector of $\mathbb{R}^I$ with components $g_j := \varphi(s_j)$. We finaly obtain the following equivalent form of the scheme (EE) in $\mathbb{R}^I$:

$$\begin{aligned} &\min\Big(\frac{U^{n+1} - U^n}{\Delta t} + AU^n + q(t_n),\ U^{n+1} - g\Big) = 0, \quad n = 0, \dots, N-1, \qquad (5) \quad \{3a\} \\ &U^0 = g. \end{aligned}$$

(where the "min" must be understood component-wise). One can check that the main iteration can also be written

$$U_i^{n+1} = \max(U_i^n - \Delta t (AU^n + q(t_n))_i,\ g_i), \quad 1 \le i \le I,$$

or, in vector form,

$$U^{n+1} = \max(U^n - \Delta t(AU^n + q(t_n)),\ g).$$

• Program the corresponding Euler Forward scheme. [3]
• Check that the program does give a stable solution with the parameters `I=20` and `N=20`.
• Check that there is an unstable behavior with other parameters (such as `I=50` and `N=20`).
• Using for instance $N \simeq 2 * I^2/10$, with $I + 1 = 20, 40, \cdots$, give an approximation of $\bar{v}$ (value at $T = 1$, and $s = S_{val}$). Typical results [4]

```
I=    19, N=    80, v:=  12.947098,   err=  0.000000,  ord= 0.00 [tcpu=  0.027]
I=    39, N=   320, v:=  13.064717,   err=  0.263003,  ord= 0.00 [tcpu=  0.209]
I=    79, N=  1280, v:=  13.109572,   err=  0.070922,  ord= 0.95 [tcpu=  1.078]
I=   159, N=  5120, v:=  13.117805,   err=  0.009205,  ord= 1.47 [tcpu= 12.508]
I=   319, N= 20480, v:=  13.119987,   err=  0.001726,  ord= 1.21 [tcpu=127.680]
```

2 A first implicit scheme: the splitting scheme

For stability reasons, we now focus on implicit schemes. We propose first to use an implicit splitting scheme.[5] Although it might be less precise than exactly solving the implicit scheme (see next section), it is much simplier to program. The scheme is as follows:

$$(i) \text{ compute } U^{n+1,(1)} \text{ s.t. } \quad \frac{U^{n+1,(1)} - U^n}{\Delta t} + AU^{n+1,(1)} + q(t_{n+1}) = 0, \tag{6}$$

$$(ii) \text{ compute } U^{n+1} \text{ s.t. } \quad U^{n+1} = \max(U^{n+1,(1)}, g). \tag{7}$$

• Program this method (for instance in the case `SCHEME='EI-AMER-SPLIT'`). The advantage of this method is to be free of a CFL condition for stability, and it is also simple to implement.
• Propose a variant of the previous scheme, of Crank-Nicolson type ($\theta = \frac{1}{2}$ scheme). [6]
• For both methods, compute the corresponding convergence tables for $(I + 1, N) = (20, 20) * 2^k$, $k = 0, 1, 2, 3, 4, \ldots$.

Notice that this splitting - Crank-Nicolson type method is not second order consistent in time with respect to the PDE (1a).[7]

[3] For instance, when parameter `SCHEME` has value `SCHEME='EE-AMER'`

[4] Errors e_k estimated by taking the differences $|v_k - v_{k-1}|$, "order" (in time) estimated as $\beta_k := \log(e_{k-1}/e_k)/\log(\Delta t_{k-1}/\Delta t_k)$.

[5] For a convergence proof of this method, we refer to Barles, Daher and Romano (1994).

[6] Solution: $U^{n+1} = \max(U^{n+1,(1)}, g)$ where $U^{n+1,1}$ solution of the Crank-Nicolson scheme, that is:

$$\frac{U^{n+1,(1)} - U^n}{\Delta t} + \frac{1}{2}(AU^{n+1,(1)} + q(t_{n+1})) + \frac{1}{2}(AU^n + q(t_n)) = 0.$$

[7] A second order method consistent is proposed in Osterlee (2003) - see also Section 4. For a precise discussion and analysis, see Bokanowski and Debrabant (2020) on arXiv.

3 Implicit Euler Scheme

For stability reasons, we now turn on the time-implicit Euler Scheme for the american option, which takes the following form:

$$\begin{aligned} &\min\Big(\frac{U^{n+1}-U^n}{\Delta t} + AU^{n+1} + q(t_{n+1}),\ U^{n+1} - g\Big) = 0, \quad n = 0, \ldots, N-1, \\ &U^0 = g. \end{aligned} \tag{8}$$ {eq:3a}

(U^n is known and we look for a solution U^{n+1}). Let us define

$$B := I_d + \Delta t A, \quad \text{and} \quad b := U^n - \Delta t\ q(t_{n+1}).$$

For each n, one must solve a solution $x \in \mathbb{R}^I$ of the following non-linear system

$$\min(Bx - b, x - g) = 0, \quad \text{in } \mathbb{R}^I. \tag{9}$$ {eq:4}

Then, we will take $U^{n+1} = x$ as the solution of the scheme (8). There exists several algorithms for solving (9). This problem is also referred as an *obstacle problem*.

3.1 PSOR Algorithm (PSOR = "Projected Successive Over Relaxation")

This is an iterative method based on the decomposition $B = L + U$ where L is the lower triangular part of B and U is the strict upper triangular part. [8]

We recall that the solution x of $min(Lx - b, x - g) = 0$ can be solved explicitly.

• Check that the solution $x = x^{k+1}$ of $\min(Lx-(b-Ux^k), x-g) = 0$ can be programmed using the following pseudo-algorithm

```
for i=1 .. n
  x(i) =( b(i)- sum_{j=1,..,J, j!=i} (B(i,j) x(j)) ) / B(i,i)
  x(i) =max(x(i),g(i))
end
```

Hence the PSOR algorithm take the form

```
# Data: matrix B=L+U, vector g, vector x0 (initial guess)
# Data: threshold eta, integer kmax
x=x0, k=0
while (k<kmax):
  xold = x
  for i=1 .. n
    x(i) =( b(i)- sum_{j=1,..,J, j!=i} (B(i,j) x(j)) ) / B(i,i)
    x(i) =max(x(i),g(i))
  end
```

[8] For a given starting vector $x^0 \in \mathbb{R}^I$, we define x^{k+1} as the solution of the system

$$min(Lx^{k+1} - (b - Ux^k), x^{k+1} - g) = 0.$$

Assuming that $L_{i,i} := B_{i,i} > 0$, the system can be solved explicitly, using that L is also lower triangular. By a fixed point argument the method can be shown to be convergent as soon as B is a strictly diagonal dominant matrix with $B_{i,i} > 0\ \forall i$.

```
k = k+1

# PRINTS FOR DEBUG PURPOSE / CONVERGENCE ANALYSIS vs. k:
# formatted print [k, norm(x-xold), norm(min(Bx-b,x-g))], such as:
err1=lng.norm(x-xold)
err2=lng.norm(np.minimum(B*x-b,x-g))
print("k=%3i, |x-xold|=%10.6f, |min(Bx-b,x-g)|=%10.6f\n" % (k,err1,err2))

if norm(x-xold)<= eta:
   STOP
if (k=kmax):
   WARNING MESSAGE
```

• Complete the iterative method in a fonction `PSOR`
• Branch on this method in the code : `SCHEMA='EI-AMER-PSOR'`.
• Observe that the method slows down for larger I values (for instance, test with $\sigma = 0.3$, $N = 10$, $I + 1 = 100$, and observe a more important number of PSOR iterations at each time iteration).
• (Optionnal) Observe that the method can be accelerated by using a relaxation method based on the following decomposition (instead of $B = L + U$):

$$B = L_w + U_w$$

where $L_w = (\frac{1}{w} - 1)D + L$, $D = diag(A)$, $U_w = B - L_w$. The parameter w can be tested with values in $(1, 2)$ - for instance $w = 1.5$.

3.2 Semi-smooth Newton's method

The following proposed method will work whatever the form of the data and payoff functions.[9]

We now want to apply a Newton type algorithm for solving $F(x) = 0$ with

$$F(x) := \min(Bx - b, x - g).$$

We consider the following algorithm: iterate over $k \geq 0$ (for a given x^0 starting point of $\mathbb{R}^I$, to be choosen)

$$x^{k+1} = x^k - F'(x^k)^{-1}F(x^k),$$

until $F(x^k) = 0$ (or, that $x^{k+1} = x^k$). We will take the following definition for $F'(x^k)$ (row by row derivative)

$$F'(x^k)_{i,j} := \begin{cases} B_{i,j} & \text{if } (Bx^k - b)_i \leq (x^k - g)_i, \\ \delta_{i,j} & \text{otherwise.} \end{cases}$$

(Note the specific choice $F'(x^k)_{i,j} = B_{i,j}$ even in the case when $(Bx^k - b)_i = (x^k - g)_i$. The other choice $F'(x^k)_{i,j} = \delta_{i,j}$ if $(Bx^k - b)_i = (x^k - g)_i$ works also but may be less efficient : more iterations might be needed.)

[9] Assuming for instance that B is an M-matrix in the sense $B_{ii} \geq 0$, $B_{ij} \leq 0$, and $B_{ii} > \sum_{j \neq i} |B_{ij}|$ for all i. An analysis of the scheme can been found in Bokanowski, Maroso, Zidani (2009). This type of algorithm goes back to Howard's algorithm, 1957.

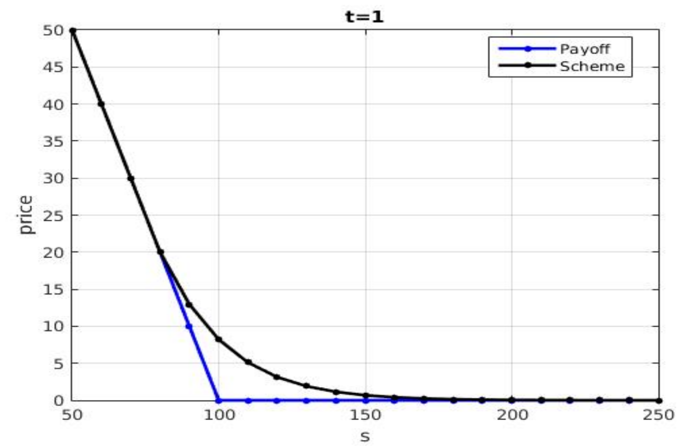

Figure 1: American put option. Evaluated at time $t = 0$ for terminal time $T = 1$ ($\sigma = 0.3$, $r = 0.1$; $N = 20$, $I = 20$).

• Program Newton's method in a function `newton`
• Program the algorithm, using Newton's method. [10]
• Test the method with $N = 20$, $I = 50$ and the classical payoff function φ_1.
• Draw errors tables: with $N = I$ and with $N = I/10$. Compare with the EI/CN splitting schemes.

Remark: With the particular payoff function φ_2, one can check that the method works also well, whereas the Brennan and Schwartz algorithm (appendix) would introduce an error when solving (9).

Remark: there are (roughly) equivalent methods for the obstacle problems, known as "Primal-Dual" method, the "policy iteration algorithm", or "Howard's algorithm".

3.3 Brennan and Schwartz algorithm (in second lecture)

There exists a direct method for solving $\min(Bx - b, x - g) = 0$, when the solution x has a particular "shape". [11] The idea is to write a decomposition of the form $B = UL$ (L: lower triangular matrix, and U: upper triangular matrix, with $U_{ii} = 1$, $\forall i$), and to use the equivalence, in some cases :

$$\min(ULx - b, x - g) = 0 \Leftrightarrow \min(Lx - U^{-1}b, x - g) = 0. \tag{10}$$

{bs1}

Then, the right-hand-side of (10) has a simple explicit solution given by

(i) solve $c = U^{-1}b$: **upwind** algorithm.
(ii) solve $\min(Lx - c, x - g) = 0$: **downwind** algorithm.

[10] For instance `SCHEME='EI-AMER-NEWTON'`
[11] In the case of the american put with one asset, and for a finite element approach, see Jaillet, Lamberton and Lapeyere (1990). The algorithm has initially been introduced by Brennan and Schwartz.

Therefore this method can be seen as a "projected" UL algorithm.

• For instance set `SCHEME='EI-AMER-UL'` in the main working file, in order to branch on this scheme.

• Program the $B = UL$ decomposition of a tridiagonal matrix B in a function of the form `[U,L]=uldecomp(B)`.

First check that the decomposition $B = UL$ is working on the specific matrix $B := I_d + \Delta t A$, in the case $I = 10$.

To do so, one can introduce in the main loop a test that is only performed at iteration `n=0`, as follows:

```
if SCHEME=='EI-AMER-UL':
    if n==0:
        # Here decompose B=UL and test the decomposition
        B= ...
        U,L = uldecomp(B);
        # Here test that the norm of B-UL is zero or close to zero:
        print('norme de B-UL: ',lng.norm(B-U@L,np.inf));

    # Here american option scheme
    ...
```

• Program the projected downwind algorithm (complete the function `descente_p`), in order to find the solution x of $min(Lx - b, x - g) = 0$.

• Program the scheme by using the upwind algorithm (which is given) and the projected downwind algorithm. Test the method with $N = 20$, $I + 1 = 50$ (and with the classical payoff function). Check that we do solve correctly the equation $\min(Bx - b, x - g) = 0$ at each time iteration. To this end one can print the norm $\|\min(Bx - b, x - g)\|$ after each new computation of the vector `U` in the main loop

```
Pold=P;
P=...  % scheme definition

err=lng.norm(min(B*U-Uold,U-payoff(s)),np.inf);
fprintf('Check:  |min(B x- b, x-g)|= %15.10f\n', err);
```

• Run the program again with the particular payoff φ_2 instead of φ_1. Check that in that case $\min(Bx - b, x - g) \neq 0$ (as soon as $n = 0$).

4 Higher order schemes

sec:BDF {sec:BDF}

Here we aim to program and test and compare three different schemes:

(i) Implicit Euler (previous section)

(ii) Crank-Nicolson: initialize with $U^0 = g$, and, for $n \geq 0$:

$$\min\Big(\frac{U^{n+1} - U^n}{\Delta t} + \frac{1}{2}(AU^{n+1} + q(t_{n+1})) + \frac{1}{2}(AU^n + q(t_n)),\ U^{n+1} - g\Big) = 0 \tag{11}$$

{eq:CN}

(iii) BDF scheme (see below)

and to draw errors tables, with $N = I$ and with $N = I/10$. (the implicit schemes can be solved by using Newton's algorithm.)

The BDF scheme (or Backward Difference Formula scheme) is as follows.[12] Initialize $U^0 = g$. Compute U^1 with the EI scheme. Then, for $n = 1, \dots, N-1$, compute U^{n+1} such that :

$$\min\Big(\frac{3U^{n+1} - 4U^n + U^{n-1}}{2\Delta t} + AU^{n+1} + q(t_{n+1}),\ U^{n+1} - g\Big) = 0 \tag{12}$$

{eq:4a}

(U^{n-1}, U^n are known and we look for a solution U^{n+1}).

• Check that the scheme is mathematically consistent of order two (check the consistency at time t^{n+1}) in time and space.

• Draw errors tables: with $N = I$ and with $N = I/10$, compare.

[12] Osterlee (2003). For an analysis, see Bokanowski and Debrabant IMA J. of Numerical Analysis, 41(2):900-934 (2021) or arXiv