EDP en finance - TP2 - Méthodes des différences finies pour une option américaine

ALOUADI Alexandre MEKKAOUI Samy

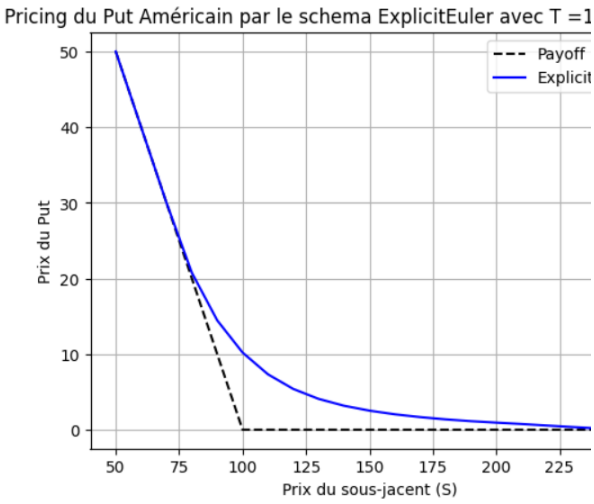
February 6, 2024

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

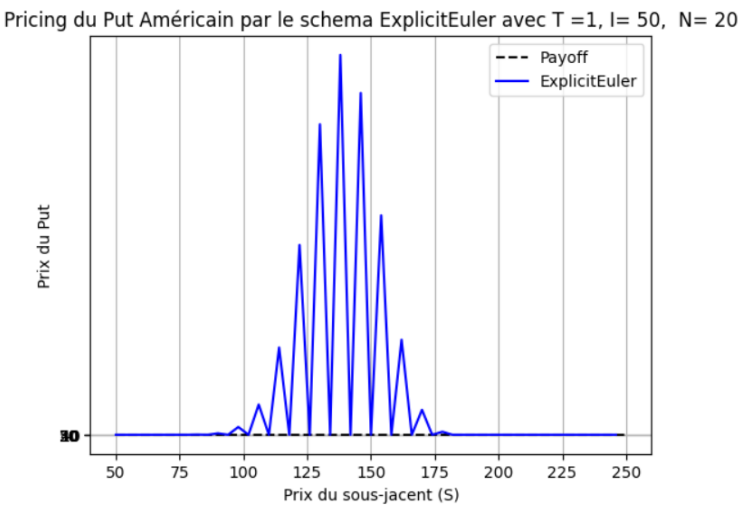
Question 1. Program the Euler Forward scheme. Check that the program does give a stable solution with the parameters $I = 20$ and $N = 20$. Check that there is an unstable behavior with other parameters (such as $I = 50$ and $N = 20$). Using for instance $N \simeq 2 * I^2/10$, with $I = 20, 40, ...$, give an approximation of $\overline{v}$ (value at $T = 1$ and $s = S_{\text{val}} = 90$).

Answer 1. Nous avons implémenté le schéma Euler Forward et nous avons pris ($I = N = 20$) et ($I = 50, N = 20$) dans les figures 1a et 1b. On remarque, comme attendu, que le schéma est stable pour $I = N = 20$ mais instable pour $I = 50$ et $N = 20$.

Une approximation de $\overline{v} := U(S_{\text{val}})$ est $\simeq 13.16$ d'après la table 1.



(a) payoff = φ_1, $N = 20$, $I = 20$



(b) payoff = φ_1, $N = 20$, $I = 50$

Figure 1: American Put Option, Explicit Euler scheme

Table 1: Convergence table: scheme EE-AMER, payoff = 1, s_val = 90

I	N	U(s_val)	err	ord	tcpu
20	80	13.637239	0.00000	0.00000	0.004891
40	320	13.447666	0.189573	0.00000	0.015272
80	1280	13.291158	0.156508	0.276515	0.058466
160	5120	13.209415	0.081742	0.937083	0.298492
320	20480	13.165633	0.043783	0.900718	1.400540

2 A first implicit scheme: the splitting scheme

Question 2. Program the method **"EI-AMER-SPLIT"** and propose a variant of the previous scheme, of Crank-Nicolson type. For both methods, compute the corresponding convergence tables for $(I, N) = (20, 20) * 2^k$ with $k = 0, 1, \ldots$

Answer 2. On a implémenté à la fois les schémas **"EI-AMER-SPLIT"** et **"CN-AMER-SPLIT"** et on donne ci-dessous les tables 2 et 3 qui permettent de répondre à la question.

Table 2: Table de convergence: Schéma EI-AMER-SPLIT, payoff = 1, s_val = 90

I	N	U(s_val)	err	ord	tcpu
20	20	13.433044	0.00000	0.00000	0.004012
40	40	13.323320	0.109725	0.00000	0.008903
80	80	13.237293	0.086026	0.351044	0.022153
160	160	13.180674	0.056619	0.603489	0.056847
320	320	13.151157	0.029517	0.939737	0.711306

Table 3: Table de convergence: Schéma CN-AMER-SPLIT, payoff = 1, s_val = 90

I	N	U(s_val)	err	ord	tcpu
20	20	13.554020	0.00000	0.00000	0.001507
40	40	13.398479	0.155541	0.00000	0.010043
80	80	13.272351	0.126129	0.302397	0.014735
160	160	13.198952	0.073399	0.781071	0.056145
320	320	13.160582	0.038370	0.935787	0.769756

3 Implicit Euler Scheme

3.1 PSOR Algorithm

Question 3. 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
```

Answer 3. Pour montrer que la solution peut être obtenue via le pseudo-code suivant, on remarque que $\min(Lx - (b - Ux^k), x - g) = 0$ est équivalent à :

$$\begin{cases} \min(L_{1,1}x_1 - (b_1 - \sum_{j=2}^{I} U_{1,j}x_j^k), x_1 - g_1) = 0 \\ \min(L_{2,1}x_1 + L_{2,2}x_2 - (b_2 - \sum_{j=3}^{I} U_{2,j}x_j^k), x_2 - g_2) = 0 \\ \vdots \\ \min(\sum_{j=1}^{I-1} L_{I-1,j}x_j - (b_{I-1} - U_{I-1,I}x_I^k), x_{I-1} - g_{I-1}) = 0 \\ \min(\sum_{j=1}^{I} L_{I,j}x_j - b_I, x_I - g_I) = 0 \end{cases} \tag{1}$$

Or, on sait que $B_{i,j} = U_{i,j}\mathbf{1}_{j>i} + L_{i,j}\mathbf{1}_{j\leq i}$ et donc le système précédent se réecrit comme suit :

$$\begin{cases} x_1 = \max\left(g_1, \frac{1}{B_{1,1}}\left(b_1 - \sum_{j=2}^{I} B_{1,j}x_j^k\right)\right) \\ x_2 = \max\left(g_2, \frac{1}{B_{2,2}}\left(b_1 - B_{2,1}x_1 - \sum_{j=3}^{I} B_{2,j}x_j^k\right)\right) \\ \vdots \\ x_{I-1} = \max\left(g_{I-1}, \frac{1}{B_{I-1,I-1}}\left(b_{I-1} - \sum_{j=1}^{I-2} B_{I-1,j}x_j - B_{I-1,I}x_I^k\right)\right) \\ x_I = \max\left(g_I, \frac{1}{B_{I,I}}\left(b_I - \sum_{j=1}^{I-1} B_{I,j}x_j\right)\right) \end{cases} \tag{2}$$

Ainsi, on vient bien de montrer que la solution du pseudo-algorithme est bien solution du problème initial.

Question 4. Complete the iterative method in a function `PSOR` and branch on this method in the code with **'EI-AMER-PSOR'**. Observe that the method slows down for larger I values (for instance, test with $\sigma = 0.3, N = 10, I = 100$, and observe a more important number of PSOR iterations at each time iteration).

Answer 4. Pour la fonction PSOR, on met $(w = 1, k_{\max} = 1000, \eta = 10^{-8})$. Ensuite, on calcule la table de convergence du schéma PSOR. Dans la fonction PSOR, on a mis $(w = 1, k_{\max} = 1000, \eta = 10^{-8})$ et on a effectivement observé que le code mettait plus de temps à être éxécuté et que la contrainte $\|x - x_{\text{old}}\|_2 \leq \eta$ n'a pas souvent été atteinte lors de l'éxécution.

3.2 Semi-smooth Newton's method

Question 6. Program Newton's method in a function `newton` and program the associated algorithm. 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.

Answer 6. Dans la figure 2, on a mis $I = 50$ et $N = 20$ et on observe bien la stabilité du schéma. On a ensuité obtenu les tableaux 4 et 5 afin de comparer le schéma de Newton par

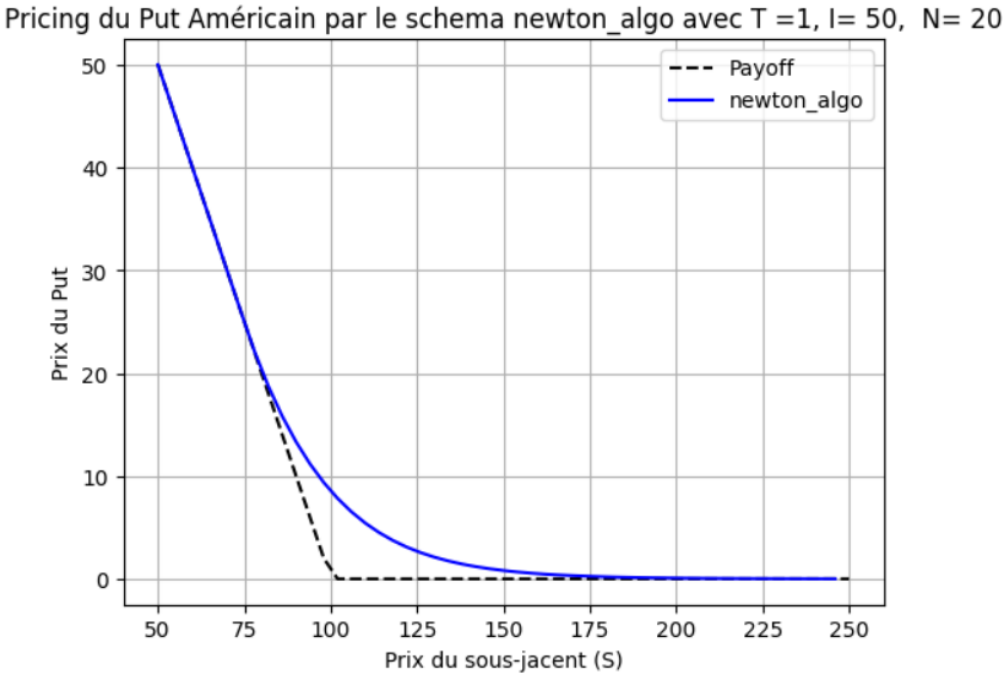


Figure 2: American Put Option, Implicit Euler scheme, Newton's method

rapport aux schémas de Splitting EI/CN. On remarque que le schéma de Newton présente une bien meilleure précision en regardant les erreurs.

Table 4: Table de convergence: Semi-Smooth Newton, payoff = 1, s_val = 90

I	N	U(s_val)	err	ord	tcpu
20	20	13.548691	0.00000	0.00000	0.005016
40	40	13.406152	0.142539	0.000000	0.018371
80	80	13.271841	0.134311	0.085778	0.041513
160	160	13.199803	0.072038	0.89875	0.298789
320	320	13.160831	0.038972	0.886325	3.424658

Table 5: Table de convergence: Semi-Smooth Newton, payoff = 1, s_val = 90

I	N	U(s_val)	err	ord	tcpu
20	2	13.037600	0.00000	0.00000	0.002012
40	4	13.119487	0.081887	0.000000	0.003008
80	8	13.126655	0.007167	3.514133	0.009722
160	16	13.124351	0.002304	1.637241	0.062544
320	32	13.122016	0.002335	-0.019237	0.854984

3.3 Brennan and Schwartz algorithm

Question 7. 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 $I = 10$.

Answer 7. Pour réaliser cette décomposition, nous faisons appel à un théorème fondamental de l'algèbre linéaire, nous permettant de décomposer une matrice carrée B en un produit PLU, où P est une matrice de permutation, L est une matrice triangulaire inférieure, et U est une matrice triangulaire supérieure, de telle sorte que $B = PLU$.

Dans notre implémentation, nous utilisons la fonction **scipy.linalg.lu** du module Python **scipy.linalg**, qui nous donne accès à des décompositions LU. Il est à noter que remplacer la décomposition UL par LU n'entraînera que des modifications mineures dans le code. Ainsi, nous privilégions l'utilisation de la décomposition LU déjà implémentée.

On remarque que le code a effectivement bien fonctionné pour $B = I_d + \Delta t A$:

```
#Test de la d composition

A = create_A(9)
dt = T/10
B = np.identity(9) + dt*A
check_decomp(B)
# Return :norme de B-UL:  5.551115123125783e-17
```

Question 8. Program the projected downwind algorithm in order to find the solution x of $\min(Lx - c, x - g) = 0$. Program the scheme by using the upwind algorithm and test the method with $N = 20, I = 50$. 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 $\| \min(Bx - b, x - g)\| = 0$ after each new computation of the vector U in the main loop. 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 $n = 0$).

Answer 8. On a utilisé la décomposition LU pour trouver la solution de $\min(Ux - c, x - g) = 0$ où $c = L^{-1}b$, le problème initial étant $\min(Bx - b, x - g) = 0$. On a programmé le schéma et on obtient la figure 3 pour $N = 50$ et $I = 20$.

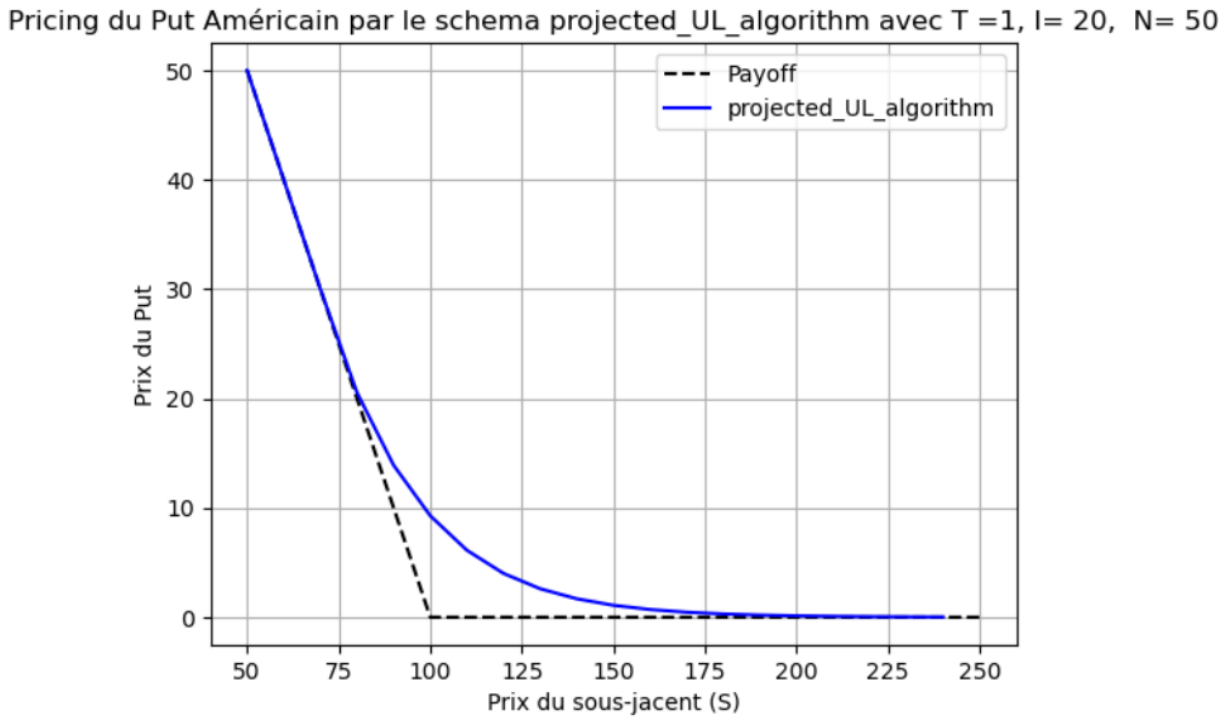


Figure 3: American Put Option, Implicit Euler scheme, Brennan & Schwartz's method

On donne ci-dessous les résultats obtenus pour vérifier si on arrive à trouver le x solution de $\min(Ux - c, x - g) = 0$.

```
def uldecomp(B):
  P, L, U = slng.lu(B) # scipy.linalg as slng
  return U,L    # B = LU, L triangulaire inf et U triangulaire inf


def check_decomp(B):
  U,L = uldecomp(B)
  print('norme de B-UL: ',lng.norm(B-L@U,np.inf));


def upwind(U, b):
    c = lng.solve(U,b)
    return c

def descente_p(L, c, g):
    x = np.zeros_like(c)
    for i in range(len(c)):
        x[i] = max((c[i] - L[i, :i] @ x[:i]) / L[i, i], g[i])
    return x

def projected_UL_algorithm(I, N, x0):
    g = create_U0(I)
    A = create_A(I)
```

```
    dt = T/N
    B = np.identity(I) + dt*A
    U, L = uldecomp(B)

    x = x0.copy()

    for n in range(N):
      xold = x.copy()
      b = x - dt * q((n+1)*dt, I)
      c = upwind(U, b)
      x = descente_p(L, c, g)
      if N - n <= 10: # on affiche juste les 10 dernieres normes
        err = np.linalg.norm(np.minimum(B @ x - xold, x - g), np.inf)
        print(f'Check: |min(B x- b, x-g)|_inf= {err}')

    return x
```

On obtient alors :

```
plot_scheme2(projected_UL_algorithm,100,1000)

## On remarque bien que quand N augmente, la norme d croit

Check: |min(B x- b, x-g)|_inf= 0.0004741063202864382
Check: |min(B x- b, x-g)|_inf= 0.00047375067854993347
Check: |min(B x- b, x-g)|_inf= 0.0004733959179237246
Check: |min(B x- b, x-g)|_inf= 0.0004730420608520802
Check: |min(B x- b, x-g)|_inf= 0.00047268912928011275
Check: |min(B x- b, x-g)|_inf= 0.0004725130171774339
```

Si on met `payoff=2`, obtient les résultats ci-dessous :

```
Check: |min(B x- b, x-g)|_inf= 0.010445244564194288
Check: |min(B x- b, x-g)|_inf= 0.010529368536754104
Check: |min(B x- b, x-g)|_inf= 0.01061380456372673
Check: |min(B x- b, x-g)|_inf= 0.010698551543116476
Check: |min(B x- b, x-g)|_inf= 0.010826252623699358
```

On voit donc que dans le cas du φ_2, l'algorithme n'arrive pas à trouver le x tel $\min(Bx - b, x - g) = 0$ à chaque itération.

On vérifie également que dans le cas du payoff $\phi = 2$, on a $\min(Bx - b, x - g) \neq 0$ dès que $n = 0$:

```
#On v rifie que min(Bx     b, x     g) != 0 d s que n = 0
#toujours pour le cas de phi2

def check(I,N,x0):
    g = create_U02(I)
    A = create_A(I)
    dt = T/N
    B = np.identity(I) + dt*A
    n = 0
    x = x0.copy()
```

```
    b = x0 - dt*q2((n+1)*dt,I)
    result = np.minimum(B@x-b,x-g) != np.zeros
    print(f"min( B x b , x g ) != 0 for n = {n}: ", np.all(result))

check(20,20,create_U02(20))

# Return : min( B x b , x g ) != 0 for n = 0:  True
```

Dans la figure 4 ci-dessous, on illustre l'implémentation du schéma "Projected-UL" :

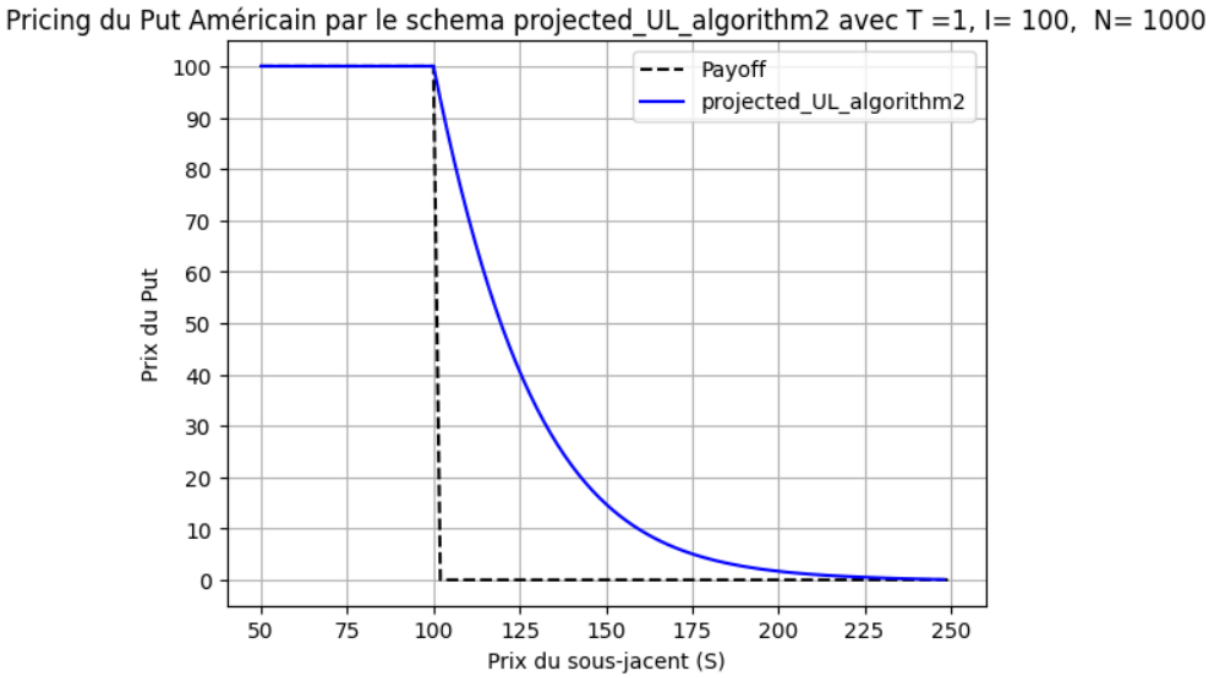


Figure 4: American Put Option,Brennan and Swchartz, ProjectedUL's method

4 Higher order schemes

Question 9. Check that the Backward Difference Formula (BDF) scheme is mathematically consistent of order two (check the consistency at time t_{n+1}) in time and space.

Answer 9. La Formule de BDF repose sur l'approximation suivante :

$$\partial_t u + \mathcal{A}u \simeq \frac{3U^{n+1} - 4U^n + U^{n-1}}{2\Delta t} + AU^{n+1} + q(t_{n+1}) \tag{3}$$

avec :

$$\mathcal{A}u := -\frac{\sigma^2}{2}s^2\partial_s^2 u - rs\partial_s u + ru \tag{4}$$

On va s'intéresser à l'approximation de $\partial_t u$ et prouver que le que schéma est consistent en temps à l'ordre 2.

$$\mathcal{E}(t) := u'(t) - \frac{3u(t) - 4u(t-\tau) + u(t-2\tau)}{2\tau} = O(\tau^2) \tag{5}$$

On va donc faire un DL de $u(t-\tau)$ eet $u(t-2\tau)$ et on sait qu'il existe $t_1, t_2 \in \mathbb{R}$ tel que :

$$\begin{cases} u(t-\tau) = u(t) - \tau u'(t) + \frac{1}{2}\tau^2 u''(t) - \frac{1}{6}\tau^3 u^{(3)}(t_1) \\ u(t-2\tau) = u(t) - 2\tau u'(t) + 2\tau^2 u''(t) - \frac{4}{3}\tau^3 u^{(3)}(t_2) \end{cases} \tag{6}$$

Donc :

$$\mathcal{E}(t) = u'(t) - \frac{2\tau u'(t) + \frac{2}{3}\tau^3 u^{(3)}(t_1) - \frac{4}{3}\tau^3 u^{(3)}(t_2)}{2\tau} \tag{7}$$

$$= \frac{1}{3}\tau^2 u^{(3)}(t_1) - \frac{2}{3}\tau^2 u^{(3)}(t_2) \tag{8}$$

Donc on a, $|\mathcal{E}(t)| \leq \tau^2 \|u^{(3)}\|_\infty$ et $\mathcal{E}(t)$ est bien d'ordre 2.

De cette analyse, on peut donc dire que le schéma BDF est bien d'ordre 2 en temps. Par ailleurs, on a vu dans le TP 1 que le terme AU^{n+1} est déjà d'ordre 2 en espace.

Question 10. Program the **'BDF-AMER'** scheme. Draw errors tables: with $N = I$ and with $N = I/10$, compare.

Answer 10. On a donc implémenté le schéma BDF et on donne les tableaux 6 et 7. On remarque que le schéma BDF est plus rapide en temps d'exécution que les schémas Euler Implicite et Crank-Nicholson dans le cas $N = I/10$ tout en étant aussi précis que le schéma de Newton.

Table 6: Table de convergence: Schéma BDF-AMER, payoff = 1, s_val = 90

I	N	U(s_val)	err	ord	tcpu
20	20	13.419939	0.00000	0.00000	0.004548
40	40	13.327285	0.092654	0.000000	0.003452
80	80	13.240180	0.087105	0.089092	0.011936
160	160	13.182809	0.07044	0.602448	0.059701
320	320	13.152550	0.030258	0.922986	0.731638

Table 7: Table de convergence: Schéma BDF-AMER, payoff = 1, s_val = 90

I	N	U(s_val)	err	ord	tcpu
20	2	10.564497	0.00000	0.00000	0.000000
40	4	12.177171	1.612673	0.00000	0.000000
80	8	12.755889	0.578718	1.478522	0.004014
160	16	12.949832	0.193944	1.577221	0.005813
320	32	13.036893	0.087061	1.155545	0.090785

Dans la figure 5 ci-dessous, on illustre l'implémentation du schéma BDF-AMER :

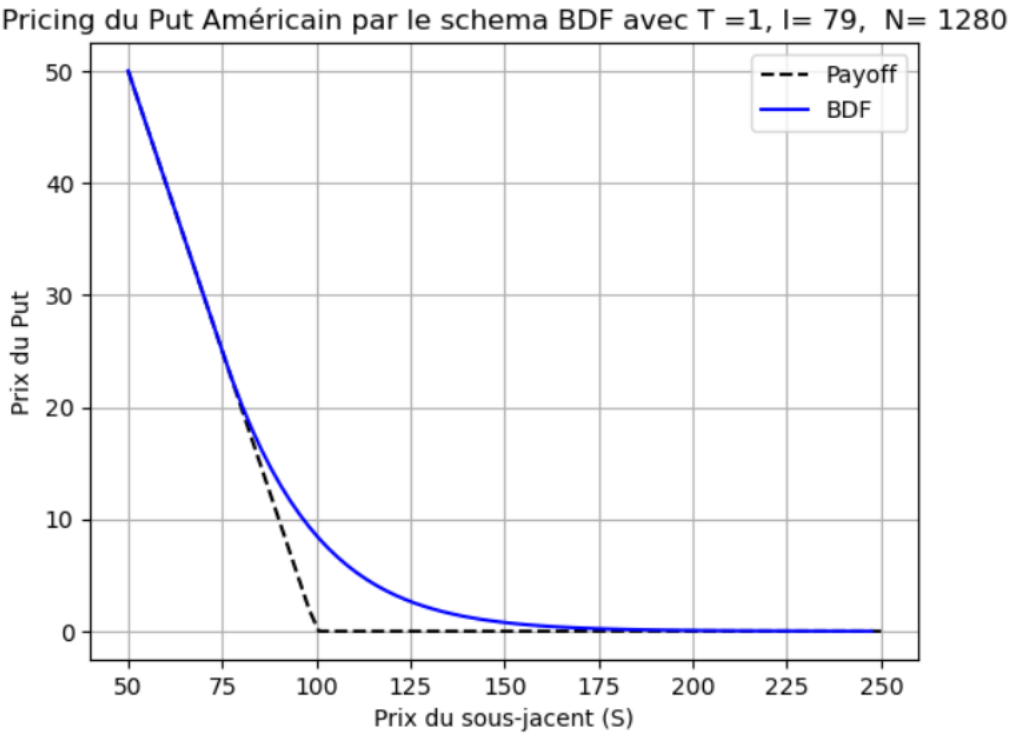


Figure 5: American Put Option,Higher Order scheme, BDF-AMER's method