ENSIIE

Rapport de projet Mathématique

# Projet de Mathématique en groupe (MPM2)

Romain Pereira
Cyril Piquet
Etienne Taillefer de Laportalière

*Enseignants :*
M. Rasamoely
M. Pulido
M. Torri
M. Lim

03/04/2018

# MPM2 Projet Mathématique 2017-2018

Etienne TAILLEFER DE LAPORTALIERE
Romain PEREIRA
Cyril PIQUET

13/02/2018

## Table des matières



## Préambule

Ce projet est réalisé dans le cadre de nos études à l'ENSIIE. L'objectif est de modéliser un marché financier et de déterminer les prix et la couverture d'option européenne.

# 1 Modèle de Cox-Ross-Rubinstein

On a $\boxed{l = 2^N}$ trajectoire possible pour l'évolution du prix de l'actif risqué, entre l'instant initial 1 et l'instant final N.
On a $\boxed{N+1}$ valeurs possibles pour $S_{t_N}^N$

**1.** $\mathbb{Q}$ est la probabilité risque-neutre, vérifiant : $\mathbb{E}_{\mathbb{Q}}[T_1^{(N)}] = 1 + r_N$
On note : $q_n = \mathbb{Q}(T_1^{(N)} = 1 + h_N)$ Donc :

$$\begin{aligned}
\mathbb{E}_{\mathbb{Q}}[T_1^{(N)}] &= \sum_{t \in T_1^{(N)}(\Omega)} x\mathbb{P}(T_1^{(N)} = t) \\
&= (1+h_n)q_N + (1+b_n)(1-q_n) \\
&= 1 + r_N \\
\Rightarrow & \quad \boxed{q_N = \frac{r_N - b_N}{h_N - b_N}}
\end{aligned} \tag{1}$$

**2.** Soit $p_(N) = \frac{1}{(1+r_N)^N}\mathbb{E}_{\mathbb{Q}}[f(S_{t_N}^{(N)})]$, le prix de l'option qui paye $f(S_{t_N}^{(N)})$.
On a :

- $p_{(0)} = f(s)$
- $$\begin{aligned}
p_{(1)} &= \frac{1}{(1+r_N)^1}\mathbb{E}_{\mathbb{Q}}[f(S_{t_1}^{(N)})] \\
&= \frac{1}{(1+r_N)^1}\mathbb{E}_{\mathbb{Q}}[f(T_1^{(N)}S_{t_0}^{(N)})] \\
&= \frac{1}{1+r_N}[q_N f[(1+h_N)s] + (1-q_N)f[(1+b_N)s)]] \\
&= [...] \text{ (en se basant sur l'arbre de la figure 1)}
\end{aligned} \tag{2}$$
- $$\boxed{p_{(N)} = \frac{1}{(1+r_N)^N}\sum_{k=0}^{N}\binom{N}{k}q_N^k(1-q_N)^{N-k}f((1+h_N)^k(1+b_N)^{N-k}s)}$$

## 1.1 Premier pricer

**Code python du 1er pricer.** Le code est disponible dans le fichier 'code/pricer.py' et 'code/Premier_pricer.py'

**3.** voir ligne 8 à 27 du fichier ‘code/Q3.py‘

**4.** voir ligne 6 du fichier ‘code/Q4.py‘

La fonction renvoie '18.5686591182' pour ces paramètres.

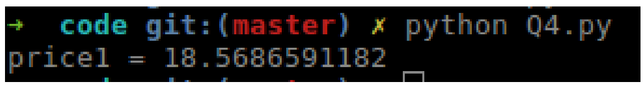


FIGURE 1 – *Résultat price1*

## 1.2 Deuxième pricer

Nous allons utiliser un algorithme de calcul récursif.

Ici on a :

$$\begin{aligned}
\forall k \in \mathbb{N}, \quad \mathbb{E}_{\mathbb{Q}}[v_{k+1}(S_{k+1}^{(N)})/S_k^{(N)}] &= \sum_{S \in (S_{k+1}^{(N)}/S_k^{(N)})(\Omega)} v_{k+1}(S)\mathbb{P}((S_{k+1}^{(N)}/S_k^{(N)}) = S) \\
&= \boxed{v_{k+1}((1+r_n)S_k^{(N)})q_n + v_{k+1}((1+b_n)S_k^{(N)})(1-q_n)} \\
v_k(S_{t_k}^N) &= \frac{E_{\mathbb{Q}}[v_{k+1}(S_{k+1}^{(N)})/S_k^{(N)}]}{1+r_N} \\
v_N(S_{t_N}^N) &= f(S_{t_k}^N)
\end{aligned}$$

Donc,

$$\boxed{v_k(S_{t_k}^N) = \frac{v_{k+1}((1+r_n)S_k^{(N)})q_n + v_{k+1}((1+b_n)S_k^{(N)})(1-q_n)}{1+r_N}}$$

On peut alors remonter avec une récursion montante (connaissant l'etat final).

**Code python du 2eme pricer.**

**5.** voir ligne 32 à 39 du fichier ‘code/Q5.py‘

**6.** voir ligne 6 du fichier ‘code/Q6.py‘

La fonction renvoie '10.7573397487' pour ces paramètres.

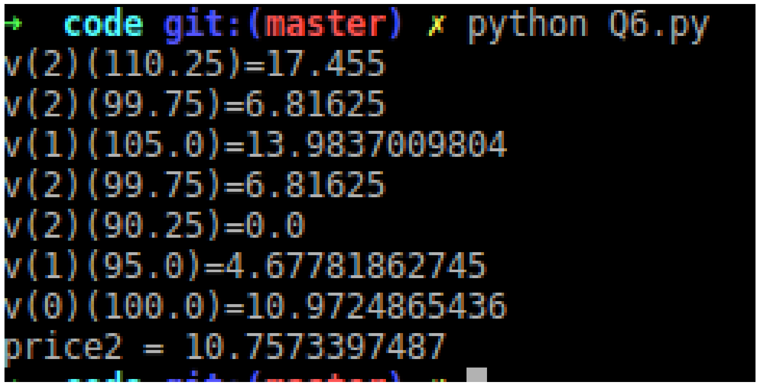


FIGURE 2 – *Résultat price2*

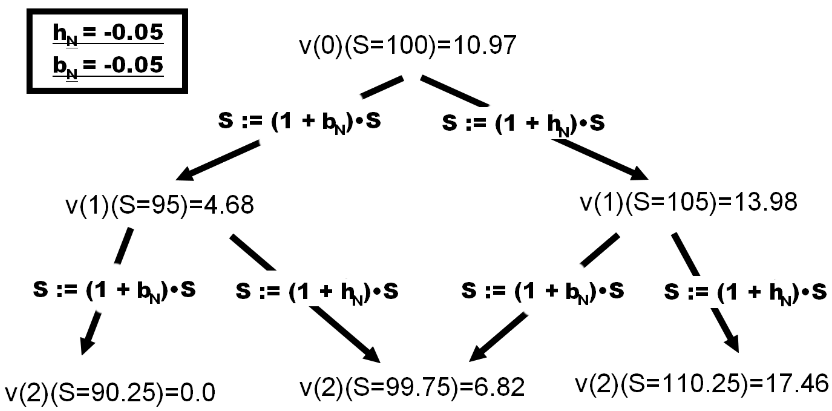


FIGURE 3 – *Résultat price1*

## 1.3 Comparaison

**7.** Pour ces entrées, (et pour toutes valeurs de N dans $[5, 15]$ les fonctions *price1* et *price2* renvoient les mêmes prix (à $10^{-11}$ prêt).

Voici un résultat obtenu, pour $N = 11$

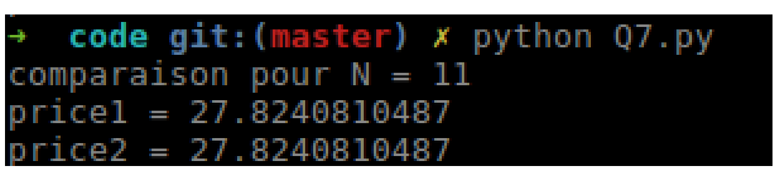


FIGURE 4 – *Comparaison price1 et price2*

## 1.4 La couverture

**8.**

$$\begin{cases}\alpha_{N-1}(S^{(N)}_{t_{N-1}})*(1+h_N)*S^{(N)}_{t_{N-1}}+\beta_{N-1}(S^{(N)}_{t_{N-1}})*S^{(0)}_{t_N}=f((1+h_N)*S^{(N)}_{t_{N-1}}) & (L_1)\\ \alpha_{N-1}(S^{(N)}_{t_{N-1}})*(1+b_N)*S^{(N)}_{t_{N-1}}+\beta_{N-1}(S^{(N)}_{t_{N-1}})*S^{(0)}_{t_N}=f((1+b_N)*S^{(N)}_{t_{N-1}}) & (L_2)\end{cases}$$

$$(L_2)-(L_1):\alpha_{N-1}(S^{(N)}_{t_{N-1}})*S^{(N)}_{t_{N-1}}*(h_N-b_N)=f((1+h_N)*S^{(N)}_{t_{N-1}})-f((1+b_N)*S^{(N)}_{t_{N-1}})$$

Donc :

$$\boxed{\alpha_{N-1}(S^{(N)}_{t_{N-1}})=\frac{f((1+h_N)*S^{(N)}_{t_{N-1}})-f((1+b_N)*S^{(N)}_{t_{N-1}})}{S^{(N)}_{t_{N-1}}*(h_N-b_N)}}$$

On injecte le résultat précédent dans $(L_1)$ :

$$\frac{f((1+h_N)*S^{(N)}_{t_{N-1}})-f((1+b_N)*S^{(N)}_{t_{N-1}})}{(h_N-b_N)}*(1+h_N)+\beta_{N-1}(S^{(N)}_{t_{N-1}})*S^{(0)}_{t_N} \quad (3)$$

$$=f((1+h_N)*S^{(N)}_{t_{N-1}}) \quad (4)$$

Donc :

$$\beta_{N-1}(S^{(N)}_{t_{N-1}})*(1+r_N)^N=\frac{(h_N-b_N)*f((1+h_N)*S^{(N)}_{t_{N-1}})-f((1+h_N)*S^{(N)}_{t_{N-1}})+f((1+b_N)*S^{(N)}_{t_{N-1}})}{(h_N-b_N)}$$

D'où :

$$\boxed{\beta_{N-1}(S^{(N)}_{t_{N-1}})=\frac{(1+h_N)*f((1+b_N)*S^{(N)}_{t_{N-1}})-(1+b_n)*f((1+h_N)*S^{(N)}_{t_{N-1}})}{(h_N-b_N)*(1+r_N)^N}}$$

**9.**

$$\begin{cases}\alpha_{k-1}(S^{(N)}_{t_{k-1}})*(1+h_N)*S^{(N)}_{t_{k-1}}+\beta_{k-1}(S^{(N)}_{t_{k-1}})*S^{(0)}_{t_k}=v_k((1+h_N)*S^{(N)}_{t_{k-1}}) & (L_1)\\ \alpha_{k-1}(S^{(N)}_{t_{k-1}})*(1+b_N)*S^{(N)}_{t_{k-1}}+\beta_{k-1}(S^{(N)}_{t_{k-1}})*S^{(0)}_{t_k}=v_k((1+b_N)*S^{(N)}_{t_{k-1}}) & (L_2)\end{cases}$$

De la même manière on obtient :

$$\boxed{\alpha_{k-1}(S^{(N)}_{t_{k-1}})=\frac{v_k((1+h_N)*S^{(N)}_{t_{k-1}})-v_k((1+b_N)*S^{(N)}_{t_{k-1}})}{S^{(N)}_{t_{k-1}}*(h_N-b_N)}}$$

$$\boxed{\beta_{k-1}(S^{(N)}_{t_{k-1}})=\frac{(1+h_N)*v_k((1+b_N)*S^{(N)}_{t_{k-1}})-(1+b_n)*v_k((1+h_N)*S^{(N)}_{t_{k-1}})}{(h_N-b_N)*(1+r_N)^N}}$$

**10.** voir code dans le fichier 'code/Q10.py'

Pour la date 0 la couverture a pour expression :

$$\boxed{\alpha_0(S_{t_0}^{(2)}) = \frac{v_1((1+h_N)*S_{t_0}^{(2)}) - v_1((1+b_N)*S_{t_0}^{(2)})}{S_{t_0}^{(2)}*(h_N - b_N)}}$$

$$\boxed{\beta_{k-1}(S_{t_{k-1}}^{(N)}) = \frac{(1+h_N)*v_k((1+b_N)*S_{t_{k-1}}^{(N)}) - (1+b_n)*v_k((1+h_N)*S_{t_{k-1}}^{(N)})}{(h_N - b_N)*(1+r_N)^N}}$$

avec ici $S_{t_0}^{(2)} = 100$ et $V_1(S_{t0}^{(2)}) = \frac{\mathbb{E}_{\mathbb{Q}}[f(S_{t1}^{(2)})/S_{t0}^{(2)}]}{1+r_N} = \frac{f((1+h_N)S_{t0}^{(2)})q_N + f((1+b_N)S_{t0}^{(2)})(1-q_N)}{1+r_N}$ Ainsi avec ces paramètres on trouve pour la couverture à la date t=0 : $\alpha_0(100) = 0.79$ et $\beta_0(100) = -78$

Pour la couverture à la date t1 il y a deux cas : Soit $S_{t1}^{(2)} = S_{t0}^{(2)}(1+h_N) = s(1+h_N) = 105$ soit $S_{t1}^{(2)} = S_{t0}^{(2)}(1+b_N) = s(1+b_N) = 95$ Il faut donc traiter ces deux cas : Dans les deux cas on a la couverture suivante à la date 1 :

$$\boxed{\alpha_1(S_{t_1}^{(2)}) = \frac{f((1+h_N)*S_{t_1}^{(2)}) - f((1+b_N)*S_{t_1}^{(2)})}{S_{t_1}^{(2)}*(h_N - b_N)}}$$

$$\boxed{\beta_1(S_{t_1}^{(2)}) = \frac{(1+h_N)*f((1+b_N)*S_{t_1}^{(2)}) - (1+b_N)*f((1+h_N)*S_{t_1}^{(2)})}{(h_N - b_N)*(1+r_N)^2}}$$

Ainsi avec ces paramètres on trouve pour la couverture à la date t=1 :

Si $S_{t1}^{(2)} = 105$ $\alpha_1(105) = 0.97$ et $\beta_1(105) = -91.35$

Si $S_{t1}^{(2)} = 95$ $\alpha_1(95) = 0$ et $\beta_1(95) = 0$

# 2 Modèle de Black-Scholes

## 2.1 Le modèle

**11.** On sait que pour toutes fonctions $g \in C^2$ on a la formule d'Ito suivante :

$$dg(S_t) = g'(S_t)dS_t + \frac{|\sigma S_t|^2}{2} g''(S_t)dt$$

On applique la formule d'Ito à à $ln(S_t)$

$$d\ln(S_t) = \frac{1}{S_t}dS_t + \frac{|\sigma S_t|^2}{2}\frac{-1}{S_t^2}dt \tag{5}$$

$$\Rightarrow \frac{d\ln(S_t)}{dt} = \frac{1}{S_t}\frac{dS_t}{dt} + \frac{|\sigma S_t|^2}{2}\frac{-1}{S_t^2} \tag{6}$$

Or on a $d(S_t) = S_t(rdt + \sigma d\beta_t)$

Donc :

$$\frac{d\ln(S_t)}{dt} = r + \frac{1}{S_t}\frac{\sigma d\beta_t}{dt} + \frac{|\sigma S_t|^2}{2}\frac{-1}{S_t^2}$$

On intègre alors la formule précédente (avec la constante en 0 : s) :

$$\ln(S_t) = \ln(s) + rt + \sigma\beta_t - \frac{\sigma^2}{2}t$$

On a donc :

$$S_t = s\exp^{rt+\sigma\beta_t-\frac{\sigma^2}{2}t}$$

$$\Rightarrow \boxed{S_t = s\exp^{\sigma\beta_t+(r-\frac{\sigma^2}{2})t}}$$

## 2.2 Le pricer par la méthode de Monte-Carlo

**12.** La formule est explicite dans le sujet, on utilise la fonction rnorm pour le $\epsilon_i$. Voir fichier ‘code/Q12.py‘

**13.** voir fichier ‘code/Q13.py‘

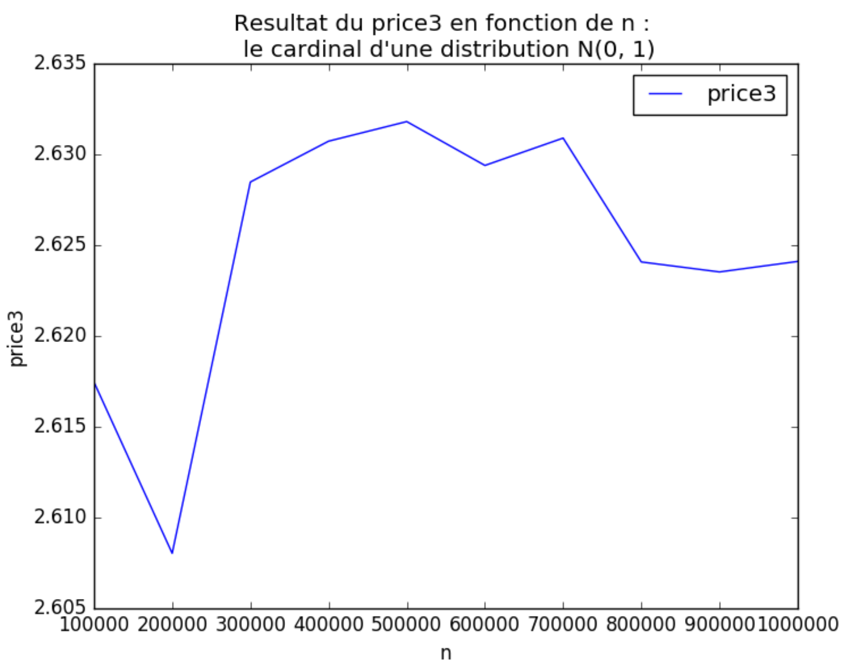


Figure 5 – *Graphique du pricer3 en fonction de 'n'*

**14.** Montrons que $(\hat{p}(n))$ converge presque sûrement vers p :

On a :

$$\hat{p}(n) = \frac{1}{n}\sum_{i=1}^{n} \exp^{-rT} f(s\exp^{(r-\frac{\sigma^2}{2})T+\sigma\sqrt{T}\xi_i})$$

Pour répondre à cette question, nous allons utiliser la loi des grands nombres.

**Loi des grands nombres** Soit $(X_i)$ une suite de variables intégrables indépendantes 2 à 2 identiquement distribuées, on a alors : $\bar{X_n}$ converge presque surement vers $\bar{\mu_n}$ avec $\bar{X_n} = \frac{1}{n}\sum_1^n X_i$ et $\bar{\mu_n} = E[X_1]$

On note $X_i = \exp^{-rT} f(s\exp^{(r-\frac{\sigma^2}{2})T+\sigma\sqrt{T}\xi_i})$ Or d'après le sujet les $(\delta_i)$ sont une suite de variables indépendantes et identiquement distribuées Donc les $X_i$ respectent les conditions de la loi des grands nombres.

Donc $\hat{p}(n)$ converge presque surement vers $E[X_1]$

Or $E[X_1] = E[exp^{-rT} f(s\exp^{(r-\frac{\sigma^2}{2})T+\sigma\sqrt{T}\xi_i})]$

De plus on sait que : $p = E[\exp^{-rt} f(S_t)]$

Or d'après la question 12 on a : $p = E[exp^{-rT} f(s\exp^{(r-\frac{\sigma^2}{2})T+\sigma\beta_T}]$ avec $\beta_T$ un mouvement Brownien donc $\forall s \in [0, T]$ on a donc $\beta_t - \beta_s \sim N(0, t-s)$. Donc pour s=0 on a : $\beta_T \sim N(0, T)$

Et donc $\frac{\beta_T}{\sqrt{T}} \sim N(0,1)$ donc on peut noter $\xi_1 = \beta_t\sqrt{t} \sim N(0,1)$

En faisant ce remplacement p devient :

$$p = E[exp^{-rT} f(s\exp^{(r-\frac{\sigma^2}{2})T+\sigma\sqrt{T}\xi_i})]$$

$$p = E[X_1]$$

Donc

$$\boxed{\hat{p}(n)\text{converge presque surement vers } p}$$

## 2.3 Le pricer par formule fermée

**15.** La formule est explicite dans le sujet, on utilise la fonction norm.cdf pour la fonction de répartition. Voir fichier ‘code/Q15.py‘

**16.** voir fichier ‘code/Q16.py‘


FIGURE 6 – *Price3 en fonction de ’n’, et ’put’*

**17.** voir fichier ‘code/Q17.py‘ On remarque que la fonction *price3* oscille autour du prix *put*, et semble converger vers le prix *put*.

Ce graphe confirme ainsi la théorie de la question 14, la suite $p_n$ converge presque surement vers p.


FIGURE 7 – *Price3 en fonction de ’n’, et ’put’*

**18.** voir fichier ‘code/Q18.py‘ On remarque que le prix du put ne dépend pas du temps mais que de s. Ainsi le prix du put est indépendant du temps.

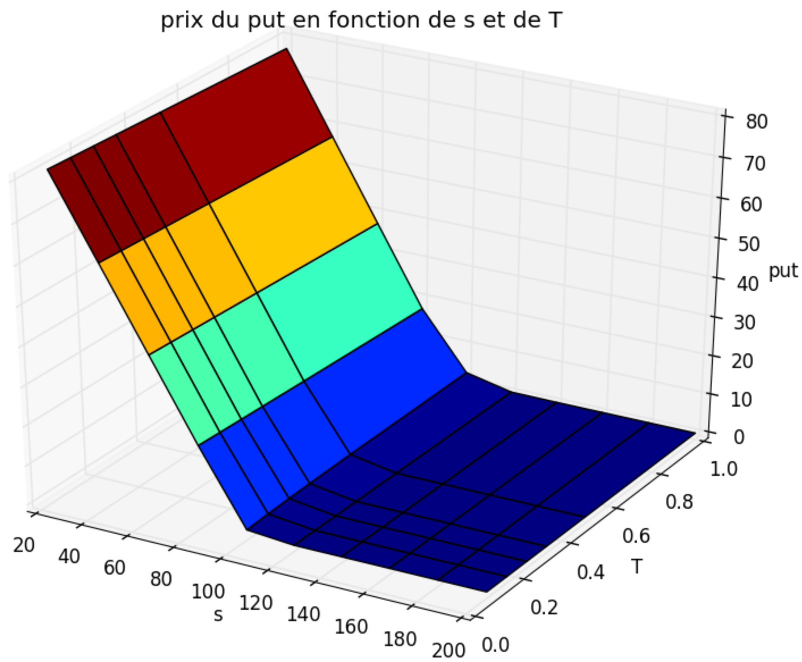


FIGURE 8 – *prix du put en fonction de s et T*

# 3 Convergence des prix

**19.** voir fichier ‘code/Q19.py‘

On remarque que le *price1* semble converger vers le prix du *put* pour $N >> 1$. Or à la question 17 nous avons vu que le pricer 3 convergait aussi vers le put. Donc ce graphe montre bien la convergence des prix dans le cas du modèle de Cox-Ross-Rubinstein(pricer1) vers les prix donnés dans le cas du modèle de Black-Scholes(pricer3).

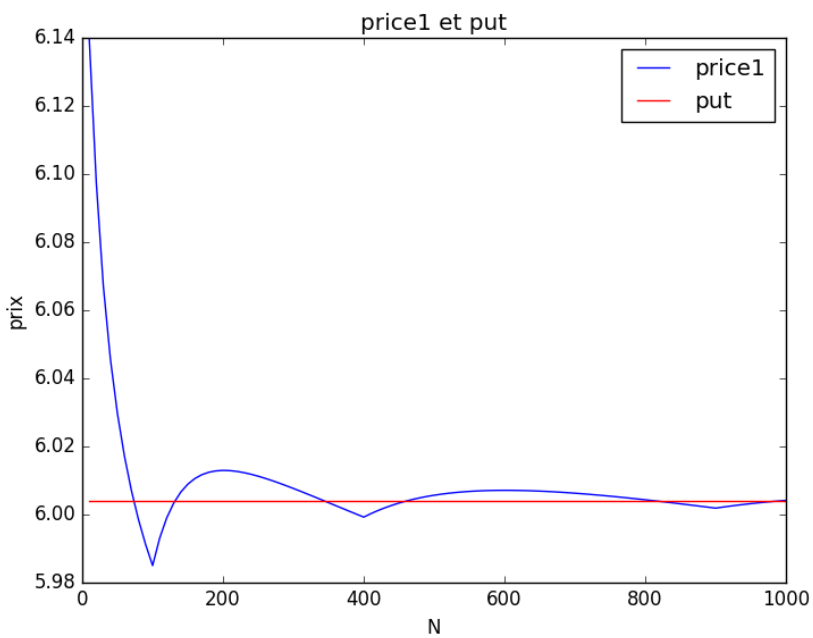


FIGURE 9 – *Price1 et put*

# 4 EDP de Black-Scholes

La fonction prix du put $P$ vérifie l'équation différentielle suivante $\forall t, S \in [0,T] \times [0,L]$ :

$$\frac{\partial P}{\partial t} + rS\frac{\partial P}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 P}{\partial S^2} = rP$$

Les conditions de bords nous donnent les valeurs de $P$ en :

$$(T, s) \quad t.q \quad s \in [0, L]$$

$$(t, 0) \quad t.q \quad t \in [0, T]$$

$$(t, L) \quad t.q \quad t \in [0, T]$$

Ce qui peut se représenter sur le plan suivant :

On cherche à intégrer numériquement, afin d'obtenir une estimation des valeurs $P(0, s), \forall s \in [0, L]$.

Pour cela, on discrétise le plan de la façon suivante. Soit $N, M \in \mathbb{N}^*$ :

$$\begin{cases} \Delta T = \frac{T}{N} \quad et \quad \forall i \in \{0, ..., N\}, \boxed{t_i = i\Delta T} \\ \Delta S = \frac{L}{M+1} \quad et \quad \forall j \in \{0, ..., M+1\}, \boxed{S_j = j\Delta S} \end{cases}$$

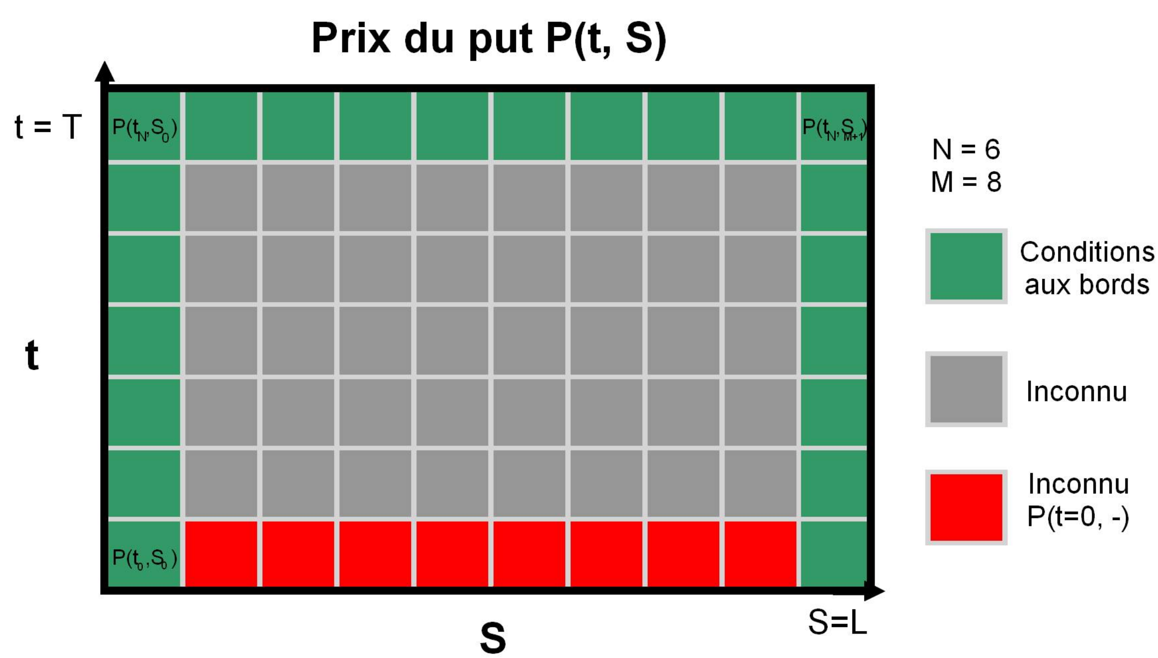


FIGURE 10 – *Grille schématisant la résolution*

De plus, on effectue les approximations suivantes : $\forall (i,j) \in \{0,...,N-1\} \times \{1,...,M\}$ et $\forall \theta \in [0,1]$ :

$$\boxed{\begin{cases} \frac{\partial P}{\partial t}(t_i,S_j) \simeq \frac{1}{\Delta T}\left(P(t_{i+1},s_j) - P(t_i,s_j)\right) \\ \frac{\partial P}{\partial S}(t_i,S_j) \simeq \theta * \frac{P(t_{i+1},s_{j+1})-P(t_{i+1},s_j)}{\Delta S} + (1-\theta) * \frac{P(t_i,s_{j+1})-P(t_i,s_j)}{\Delta S} \\ \frac{\partial^2 P}{\partial S^2}(t_i,S_j) \simeq \theta * \frac{P(t_{i+1},s_{j+1})-2P(t_{i+1},s_j)+P(t_{i+1},s_{j-1})}{\Delta S^2} + (1-\theta) * \frac{P(t_i,s_{j+1})-2P(t_i,s_j)+P(t_i,s_{j-1})}{\Delta S^2} \end{cases}}$$

**Remarque**

- Pour $\theta = 0$, on obtient un schéma d'Euler implicit
- Pour $\theta = 1$, on obtient un schéma d'Euler explicit
- Pour $\theta = \frac{1}{2}$, on obtient la méthode de Crank-Nicolson

En approximant l'équation différentielle $\forall (i,j) \in \{0,...,N-1\} \times \{1,...,M\}$ :

$$\boxed{\frac{\partial P}{\partial t}(t_i,S_j) + rS_j\frac{\partial P}{\partial S}(t_i,S_j) + \frac{1}{2}\sigma^2 S_j^2 \frac{\partial^2 P}{\partial S^2}(t_i,S_j) = rP(t_{i+1},S_j)}$$

après developpement et séparation des termes en $t_i$ et $t_{i+1}$, on obtient :

$$\begin{aligned}
&\left[\frac{1}{\Delta T} + \left(r\frac{S_j}{\Delta S} + \sigma^2\frac{S_j^2}{\Delta S^2}\right)(1-\theta)\right] P(t_i,S_j) && (7)\\
&-\left(r\frac{S_j}{\Delta S} + \frac{1}{2}\sigma^2\frac{S_j^2}{\Delta S^2}\right)(1-\theta)P(t_i,S_{j+1}) && (8)\\
&-\frac{1}{2}\sigma^2\frac{S_j^2}{\Delta S^2}(1-\theta)P(t_i,S_{j-1}) && (9)\\
=&\left[-r + \frac{1}{\Delta T} - \left(r\frac{S_j}{\Delta S} + \sigma^2\frac{S_j^2}{\Delta S^2}\right)\theta\right] P(t_{i+1},S_j) && (10)\\
&+\left(r\frac{S_j}{\Delta S} + \frac{1}{2}\sigma^2\frac{S_j^2}{\Delta S^2}\right)\theta P(t_{i+1},S_{j+1}) && (11)\\
&+\frac{1}{2}\sigma^2\frac{S_j^2}{\Delta S^2}\theta P(t_{i+1},S_{j-1}) && (12)
\end{aligned}$$

$$\Leftrightarrow$$

$$\begin{aligned}
&A_{i,j}P(t_i,S_j) + A_{i,j+1}P(t_i,S_{j+1}) + A_{i,j-1}P(t_i,S_{j-1}) && (13)\\
=\ &B_{i+1,j}P(t_{i+1},S_j) + B_{i+1,j+1}P(t_{i+1},S_{j+1}) + B_{i+1,j-1}P(t_{i+1},S_{j-1}) && (14)
\end{aligned}$$

Ce qui peut se représenter sous forme des systèmes linéaires suivant $\forall i \in \{0,...,N-1\}$

$$\boxed{AX_i = BX_{i+1} + C_i}$$

avec :

- $\forall i \in \{0, ..., N\}, X_i = \begin{pmatrix} P(t_i, S_0) \\ ... \\ P(t_i, S_{M+1}) \end{pmatrix}$ ($X_N$ est connu par les conditions initiales, on recherche à déterminer $X_0$)

- A (tridiagonale) $=$

$$\begin{array}{cc}
\begin{array}{cccccccccc} 0 & 1 & 2 & ... & j-1 & j & j+1 & ... & M & M+1 \end{array} & \\
\begin{pmatrix}
A_{0,0} & A_{0,1} & 0 & 0 & ... & ... & ... & ... & ... & 0 \\
A_{1,0} & A_{1,1} & A_{1,2} & 0 & ... & ... & ... & ... & ... & 0 \\
0 & A_{2,1} & ... & ... & ... & ... & ... & ... & ... & ... \\
0 & 0 & ... & ... & ... & ... & ... & ... & ... & ... \\
... & ... & ... & ... & ... & ... & ... & ... & ... & ... \\
... & ... & ... & ... & A_{j,j-1} & A_{j,j} & A_{j,j+1} & ... & ... & ... \\
... & ... & ... & ... & ... & ... & ... & ... & ... & ... \\
... & ... & ... & ... & ... & ... & ... & ... & ... & 0 \\
... & ... & ... & ... & ... & ... & ... & ... & ... & A_{M,M+1} \\
0 & 0 & ... & ... & ... & ... & ... & 0 & A_{M+1,M} & A_{M+1,M+1}
\end{pmatrix}
&
\begin{array}{c} 0 \\ 1 \\ 2 \\ ... \\ j-1 \\ j \\ j+1 \\ ... \\ M \\ M+1 \end{array}
\end{array}$$

  - $A_{0,0} = 1$
  - $A_{0,1} = 0$
  - $A_{M+1,M} = 0$
  - $A_{M+1,M+1} = 1$
  - $\forall j \in \{1, ..., M\} \begin{cases} A_{j,j-1} & = & -\frac{1}{2}\sigma^2 \frac{S_j^2}{\Delta S^2}(1-\theta) \\ A_{j,j} & = & \frac{1}{\Delta T} + \left(r\frac{S_j}{\Delta S} + \sigma^2 \frac{S_j^2}{\Delta S^2}\right)(1-\theta) \\ A_{j,j+1} & = & -\left(r\frac{S_j}{\Delta S} + \frac{1}{2}\sigma^2 \frac{S_j^2}{\Delta S^2}\right)(1-\theta) \end{cases}$
  - Tous les autres $A_{i,j} = 0$

- B (tridiagonale) $=$

$$\begin{array}{cc}
\begin{array}{cccccccccc} 0 & 1 & 2 & ... & j-1 & j & j+1 & ... & M & M+1 \end{array} & \\
\begin{pmatrix}
B_{0,0} & B_{0,1} & 0 & 0 & ... & ... & ... & ... & ... & 0 \\
B_{1,0} & B_{1,1} & B_{1,2} & 0 & ... & ... & ... & ... & ... & 0 \\
0 & B_{2,1} & ... & ... & ... & ... & ... & ... & ... & ... \\
0 & 0 & ... & ... & ... & ... & ... & ... & ... & ... \\
... & ... & ... & ... & ... & ... & ... & ... & ... & ... \\
... & ... & ... & ... & B_{j,j-1} & B_{j,j} & B_{j,j+1} & ... & ... & ... \\
... & ... & ... & ... & ... & ... & ... & ... & ... & ... \\
... & ... & ... & ... & ... & ... & ... & ... & ... & 0 \\
... & ... & ... & ... & ... & ... & ... & ... & ... & B_{M,M+1} \\
0 & 0 & ... & ... & ... & ... & ... & 0 & B_{M+1,M} & B_{M+1,M+1}
\end{pmatrix}
&
\begin{array}{c} 0 \\ 1 \\ 2 \\ ... \\ j-1 \\ j \\ j+1 \\ ... \\ M \\ M+1 \end{array}
\end{array}$$

  - $B_{0,0} = 0$
  - $B_{0,1} = 0$
  - $B_{M+1,M} = 0$
  - $B_{M+1,M+1} = 0$
  - $\forall j \in \{1, ..., M\} \begin{cases} B_{j,j-1} & = & \frac{1}{2}\sigma^2 \frac{S_j^2}{\Delta S^2}\theta \\ B_{j,j} & = & -r + \frac{1}{\Delta T} - \left(r\frac{S_j}{\Delta S} + \sigma^2 \frac{S_j^2}{\Delta S^2}\right)\theta \\ B_{j,j+1} & = & \left(r\frac{S_j}{\Delta S} + \frac{1}{2}\sigma^2 \frac{S_j^2}{\Delta S^2}\right)\theta \end{cases}$

- Tous les autres $B_{i,j} = 0$

- $C_i$ conditions de bords $= \begin{array}{c} \begin{pmatrix} P(t_i, S_0) \\ 0 \\ ... \\ 0 \\ P(t_i, S_{M+1}) \end{pmatrix} \end{array} \begin{array}{c} 0 \\ 1 \\ ... \\ M \\ M+1 \end{array}$

**Remarque** Les valeurs de $A_{0,0}, B_{0,0}, B_{0,1}, B_{M+1,M}, B_{M+1,M+1}, C_0, C_M$ ont été choisit de sorte à ce que l'équation $AX_i = BX_{i+1} + C_i$ soit vérifié pour les lignes 0 et $M+1$. Ce sont les *conditions initiales*.

$X_N$ est determiné par les conditions initiales, on peut donc en déduire $X_{N-1}$, puis par récurrence sur le système, $X_i$ devient connu $\forall i \in \{0, ..., N-1\}$.

On a donc $N-1$ systèmes linéaires : $\forall i \in \{0, ..., N-1\}, AX_i = b_i$, avec $b_i = BX_{i+1} + C_i$, qu'il nous reste à résoudre dans l'ordre $i$ décroissant, afin d'obtenir $X_0$ : approximation du prix du put $P(0, S)$ en $t = 0, S \in [0, L]$

**Optimisation** $A$ est indépendante de $i$, on a donc uniquement à la générer une fois au début de la résolution. Avec une résolution par la méthode de Gauss (triangulation + résolution), on aurait une complexité temporelle en $\frac{2}{3}M^3 + o(M^3)$ Or, $A$ est tridiagonale, on peut donc optimiser la résolution, et on atteinds une complexité temporelle en $8M + o(M)$ (factorisation LU + résolution) (voir 1 p.8-9)

De plus, on gagne aussi en performance mémoire : $A$ tridiagonale peut être stocker en $3M + o(M)$ (de même pour $B$)

**Résolution** La résolution de ces systèmes (en Python) est réalisé avec la bibliothèque *scipy.linalg* et *numpy*, qui propose une fonction optimisé de résolution de tels systèmes. (voir 2)

Le code qui a généré les résultats suivant est disponible dans les fichiers 'code/EDP.py'.

Voici l'algorithme en pseudo-code :

**Algorithm 1** Résolution de l'équation différentielle de Black-Scholes

**Require:** $M, N \in \mathbb{N}^*, K, R, \sigma \in \mathbb{R}, \theta \in [0, 1]$
  $\delta T \leftarrow \frac{T}{N}$
  $\delta S \leftarrow \frac{T}{M+1}$
  Initialiser les matrices $A$ et $B$ (avec $K, R, \sigma, \theta, \delta T$ et $\delta S$)
  Initialiser $X_N$ (conditions aux bords)
  **for** $i := N-1$ ; $i >= 0$ ; $i := i-1$ **do**
    Calculer $b_i \leftarrow B.X_{i+1} + C_i$
    Résoudre systeme triangulaire : $X_i := solve(A.X_i = b_i)$
  **end for**
  Renvoyer $X_0$

Pour les méthodes **implicit** et de **Crank Nicholson**, la méthode est toujours **stable** (voir 3) On obtient une erreur relative entre les 2 solutions :

```
erreurs relatives:
Entre 0.0 et 0.5 (implicit et Crank Nicholson): 0.0182623582044
```

FIGURE 11 – *Erreur relative entre les 2 solutions obtenus*

On obtient les solutions suivantes pour le prix du put à $t = 0$ :

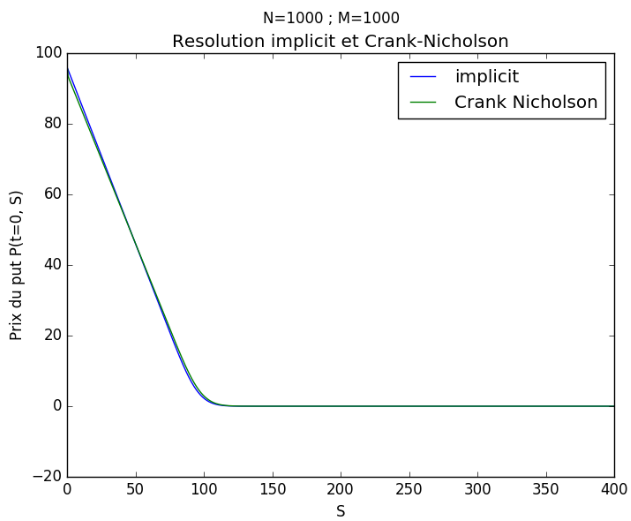


FIGURE 12 – *Solutions obtenus par méthode implicit et Crank-Nicholson*

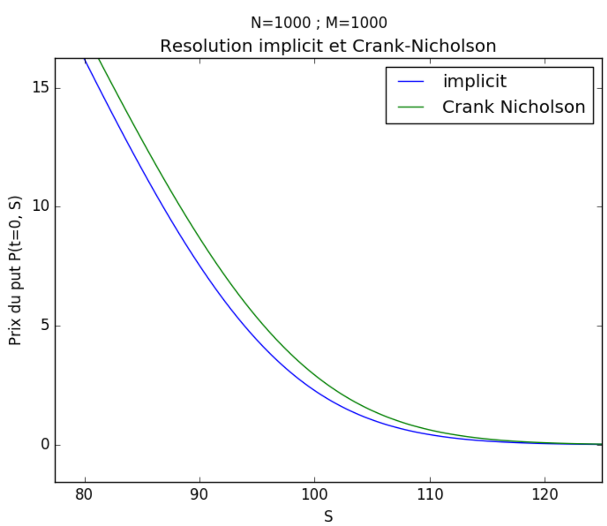


FIGURE 13 – *Solutions obtenus par méthode implicit et Crank-Nicholson (zoomées)*

Pour la méthode **explicit**, le schéma est stable si le pas de temps est de l'ordre du carré du pas d'espace. Autrement dit, si :

$$\Delta t \simeq \Delta S^2$$

On observe cette instabilité pour $M = N = 296$ (voir 'code/Q20/EDP_296_296.py') :

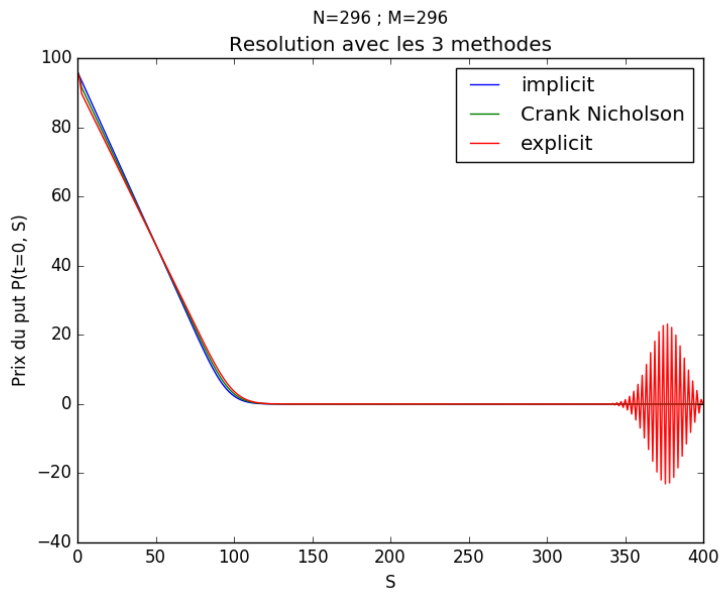


FIGURE 14 – *Solution avec les 3 méthodes*

```
erreurs relatives:
Entre 0.0 et 0.5 (implicit et Crank Nicholson): 1.52693458709e-05
Entre 0.0 et 1.0 (implicit et explicit)       : 0.156261138262
Entre 0.5 et 1.0 (explicit et Crank Nicholson): 0.156261215572
```

FIGURE 15 – *Erreur relative entre les 3 méthodes pour* $M = N = 296$

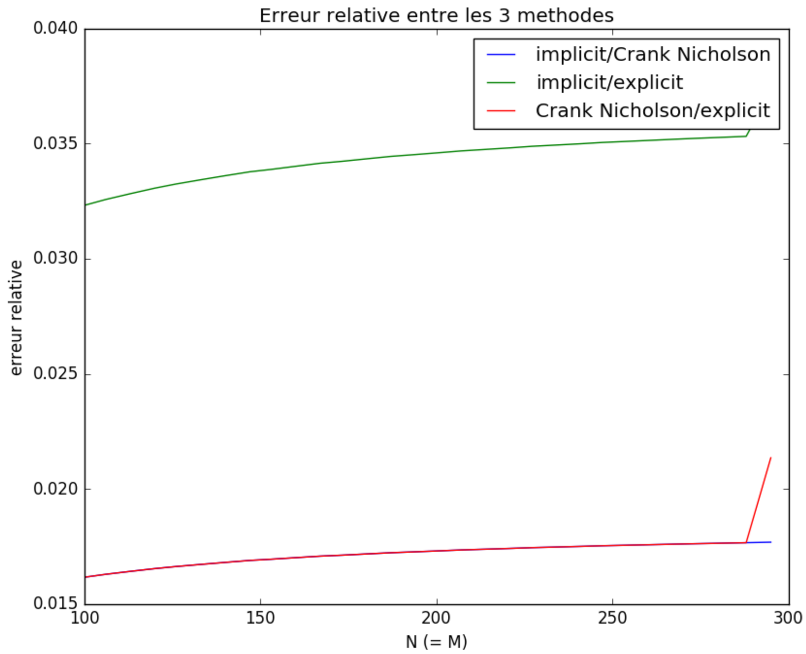


FIGURE 16 – *Erreur relative entre les 3 méthodes pour* $M = N \in [100, 296]$

On en déduit que lorsque les 3 méthodes convergent, elles donnent des résultats très proches. Pour $M$ plus grand que 296, la diverge de la méthode explicit fait exploser l'erreur relative.

Pour $M = 1000$ et $N = M^2$, la méthode explicit est stable et on obtient la solution suivante (voir 'code/Q20/EDP_1000_1000000.py') :

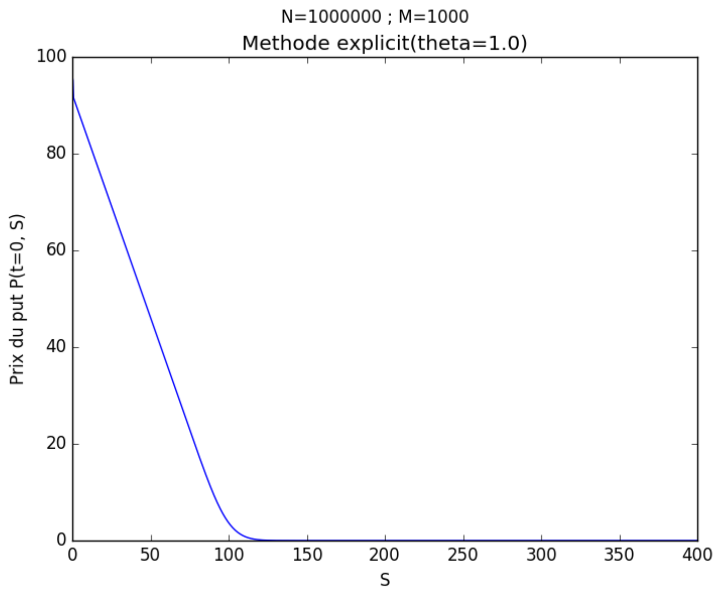


FIGURE 17 – *Solution explicit avec un rapport quadratique sur les pas*

```
python EDP.py  917,25s user 0,30s system 96% cpu 15:47,37 total
```

FIGURE 18 – *Temps d'execution de la résolution explicit pour $M = 1000$ et $N = M^2$ précèdente (15m47s)*

**Expression approchée** Finalement, en guise de **bonus**, on se propose d'obtenir une expression approchée du prix du put $P(t=0,S)$. Pour cela, et d'après l'allure des figures, on se propose d'approcher cette fonction par le modèle exponentielle suivant (voir 'code/Q20/EDP_Modele_exponentielle.py') :

$$\boxed{P(t=0,S) = ae^{-bS^3-cS+d} + e \quad t.q. \quad a,d,e \in \mathbb{R} \quad et \quad b,c \in \mathbb{R}_+^*}$$

Grâce aux modules 'numpy' et 'scipy' de Python, on a pu obtenir un modèle rapidement (voir curve_fit 4) :

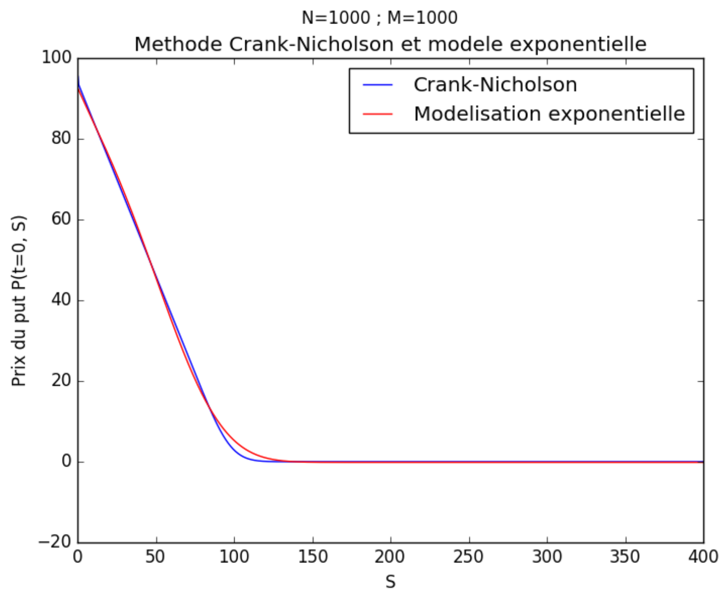


Figure 19 – *Modèle exponentielle obtenu*

```
erreur relative:
0.0278998216807
modele P(t=0, S) = a * exp(-b * S^3 - c * S + d) + e
avec: a=5.032027, b=0.000002, c=0.009344 d=2.915998 e=-0.156397
```

Figure 20 – *Erreur relative et paramètre optimal obtenu, entre le résultat obtenu par la méthode de Crank-Nicholson et le modèle exponentielle*

# 5 Références


[1] 'Analyse Numérique' - Paola GOATIN
*https://team.inria.fr/opale/files/2011/11/Anum.pdf*

[2] 'Résolution de système linéaires 'en bande' - SCIPY
*https://docs.scipy.org/doc/scipy/reference/linalg.html*

[3] 'Finite difference method' - Wikipédia
*https://en.wikipedia.org/wiki/Finite_difference_method*

[4] scipy.optimize.curve - Documentation scipy
*https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html*