École Centrale Casablanca

Résolution de l'équation Black-Scholes par la méthode des différences finies

Auteur :
Saad TAZROUTE
Akram IBN EL AHMAR

Enseignant :
Anass Bouchnita
Aissam Jebrane
Bouchra BENSIALI

3 avril 2020

Table des matières

1 Description du problème : 2
1.1 Définitions : . . . 2
1.2 L'équation de Black-Scholes : . . . 2
1.3 Définition des variables : . . . 2

2 Méthode des différences finies 2
2.1 Les conditions aux limites : . . . 3
2.2 Schéma explicite . . . 4
2.2.1 Discretisation . . . 4
2.2.2 Matrice associée . . . 4
2.3 Schéma implicite . . . 5
2.3.1 Discretisation . . . 5
2.3.2 Matrice associée . . . 5
2.3.3 Decomposition LU . . . 5

3 Résultats numériques : 6

4 Discussion des résultats : 9

5 Conclusion : 9

1 Description du problème :

1.1 Définitions :

Une option est un contrat à terme conditionnel, qui confère à son acheteur, moyennant le paiement d'une prime, le droit - et non l'obligation - d'acheter ou de vendre une quantité d'un actif : action, obligation, devise, ... à un prix déterminé lors de la négociation du contrat : prix d'exercice ou Strike et à une date ou jusqu'à une date qui limite la durée de vie de l'option : échéance ou maturité.
Si l'option ne peut être exercée qu'à échéance, on parle d'option européenne. Si l'option peut être exercée tout au long de sa durée de vie, on parle d'option Américaine.
On parle de call pour une option d'achat et de put pour une option de vente. En d'autres termes, pour un Call je vends ou j'achète le droit de vendre mon produit à un prix déjà fixé, pour un Put je vends ou j'achète le droit qu'on me vende un produit à un prix déjà fixé.

1.2 L'équation de Black-Scholes :

L'équation qui caractérise le prix d'un dérivé d'un primitif, est l'équation Black-Scholes.

$$\frac{\partial V}{\partial t} = rS\frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} - rV \tag{1}$$

1.3 Définition des variables :

- $S(t)$: le prix du sous-jacent en fonction du temps, le prix en t de ce que l'on souhaite vendre.
- $V(S,t)$: le prix de l'option qui dépend dans notre approche du prix du sous-jacent et en fonction du temps.
- r : le taux d'intérêt annuel sans risque, le taux de rendement d'un investissement hypothétique sans risque de perte financière, sur une période de temps donnée.
- σ : L'écart type est souvent utilisé par les investisseurs pour mesurer le risque d'une action ou d'un portefeuille d'actions. L'idée de base est que l'écart-type est une mesure de la volatilité : plus le rendement d'un titre diffère du rendement moyen du titre, plus le titre est volatil.

2 Méthode des différences finies

Dans cette section, nous nous focaliserons sur la méthode des différences finies pour la résolution de l'équation. Pour résoudre une équation à dérivées partielles numériquement, on se base sur une approximation des dérivées, selon chaque schéma utilisé on obtient un système linéaire d'équations qu'on doit résoudre par la suite. Les deux schémas envisagés dans la suite de ce document, le schéma explicite et le schéma implicite.
Cette section comprend l'approximation des dérivées partielles, la discrétisation de l'équation et les conditions aux limites associées au problème.

2.1 Les conditions aux limites :

Avant de se lancer dans le fond de la résolution numérique, nous discutrons le choix des conditions limites, un choix qui reste primordial dans le processus de fixation du prix (Pricing). Il existe plusieurs manières de caractériser les conditions aux limites, dans notre cas nous nous focaliserons sur le cas d' European put.

$$\begin{aligned} \frac{\partial V}{\partial t} &= rS\frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} - rV \\ V(0,t) &= 0,\ V(S_{\max},t) = S_{\max} - Ke^{-r(T-t)}, \quad 0 \le t \le T \\ V(S,T) &= \max(S-K,0), \quad 0 \le S \le S_{\max} \end{aligned}$$

L'équation limite de gauche $V(0,t) = 0$ indique que dans la limite de gauche, pour tous t jusqu'à la maturité, le prix de l'option sera égale à 0.
L'équation de droite $V(S_{\max},t) = S_{\max} - Ke^{-r(T-t)}$pour la frontière de droite indique que, pour tous les temps jusqu'à l'expiration, la valeur de l'appel sera égale à la fonction de remboursement, bien que légèrement ajustée pour tenir compte de l'actualisation due au taux sans risque, ce qui résulte de la parité Put-Call. Ces deux conditions limites sont du type Dirichlet.

La condition initiale (puisqu'on évolue inversement)$V(S,T) = \max(S-K,0)$, qui décrit comment la solution doit se comporter au début de la procédure de recherche de temps. Comme nous marchons à reculons, il s'agit en fait du paiement final de l'option à l'échéance, ce qui est l'expression familière pour un paiement d'option d'achat.

La condition de type Neumann spécifie la dérivée partielle de l'option à la frontière. La discrétisation de la dérivée seconde à la frontière par la différence centrale nous donne une équation plus simple pour le $V_{i,j}$.
En outre, la condition de Neumann est plus précise pour les mêmes limites que la condition de Dirichlet, car les dérivées secondes diminuent plus vite que le prix. Et la condition de Neumann est plus universelle, la même pour l'appel et l'option de vente, et pour les deux extrémités.

Aux deux extremités

$$\frac{\partial^2 V}{\partial S^2}(\delta S, t) = 0$$

$$\frac{\partial^2 V}{\partial S^2}((N-1)\delta S, t) = 0$$

Ce qui nous donne deux équations :

$$\begin{aligned} &\Rightarrow V_{0,j} - 2V_{1,j} + V_{2,j} = 0 \\ &\Rightarrow V_{M-2,j} - 2V_{M-1,j} + V_{M,j} = 0 \end{aligned}$$

2.2 Schéma explicite

2.2.1 Discretisation

En discretisant l'équation de Black-Scholes suivant le schéma explicite, on obtient l'équation suivante :

$$\frac{V_{i,j} - V_{i,j-1}}{dt} + ridS\frac{V_{i+1,j} - V_{i-1,j}}{2dS} + \frac{1}{2}\sigma^2(idS)^2\frac{V_{i+1,j} - 2V_{i,j} + V_{i-1,j}}{(dS)^2} = rV_{i,j} \qquad \text{(XXX)}$$

On obtient alors l'équation discrétisée suivante :

$$\alpha_i V_{i-1,j} + (1 - \beta_i)V_{i,j} + \gamma_i V_{i+1,j} = V_{i,j-1}$$

Avec α β γ sont des constantes définies par :

$$\alpha_i = \frac{1}{2}idt(i\sigma^2 - r)$$

$$\beta_i = -dt(i^2\sigma^2 + r)$$

$$\gamma_i = \frac{1}{2}idt(i\sigma^2 + r)$$

A partir de notre maillage, des conditions aux limites et l'équation qui remplace numériquement notre équation, on peut passer à la formulation matricielle pour résoudre le problème.

2.2.2 Matrice associée

L'équation obtenue peut être mise sous la forme matricielle, la matrice A sera de dimension XXXX :

$$\begin{pmatrix} 1+\beta_1 & \gamma_1 & 0 & \cdots & 0 \\ \alpha_2 & 1+\beta_2 & \gamma_2 & \cdots & 0 \\ 0 & \alpha_3 & 1+\beta_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \gamma_{M-2} \\ 0 & 0 & 0 & \alpha_{M-1} & 1+\beta_{M-1} \end{pmatrix} \begin{pmatrix} V_{1,j} \\ V_{2,j} \\ V_{3,j} \\ \vdots \\ V_{M-1,j} \end{pmatrix} + \begin{pmatrix} \alpha_1 V_{0,j} \\ 0 \\ 0 \\ \vdots \\ \gamma_{M-1}V_{M,j} \end{pmatrix} = \begin{pmatrix} V_{1,j-1} \\ V_{2,j-1} \\ V_{3,j-1} \\ \vdots \\ V_{M-1,j-1} \end{pmatrix}$$

On obtient alors en utilisant les conditions aux limites de type Neumann, 2.1, 2.1, la formulation matricielle suivante :

$$\begin{pmatrix} 2\alpha_1 + 1 + \beta_1 & \gamma_1 - \alpha_1 & 0 & \cdots & 0 \\ \alpha_2 & 1+\beta_2 & \gamma_2 & \cdots & 0 \\ 0 & \alpha_3 & 1+\beta_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \gamma_{M-2} \\ 0 & 0 & 0 & \alpha_{M-1} - \gamma_{M-1} & 2\gamma_{M-1} + 1 + \beta_{M-1} \end{pmatrix} \begin{pmatrix} V_{1,j} \\ V_{2,j} \\ V_{3,j} \\ \vdots \\ V_{M-1,j} \end{pmatrix} = \begin{pmatrix} V_{1,j-1} \\ V_{2,j-1} \\ V_{3,j-1} \\ \vdots \\ V_{M-1,j-1} \end{pmatrix}$$

On arrive alors à l'équation matricielle qui nous permet de calculer toutes les valeurs de $V_{i,j}$ en connaissant la valeur $V_{N,j} = max(K - j\delta S, 0)$

2.3 Schéma implicite

2.3.1 Discretisation

En discretisant l'équation de Black-Scholes suivant le schéma implicite, on obtient l'équation suivante :

$$\frac{V_{i,j+1} - V_{i,j}}{dt} + ridS\frac{V_{i+1,j} - V_{i-1,j}}{2dS} + \frac{1}{2}\sigma^2(idS)^2\frac{V_{i+1,j} - 2V_{i,j} + V_{i-1,j}}{(dS)^2} = rV_{i,j}$$

$$\alpha_i V_{i-1,j} + (1+\beta_i)V_{i,j} + \gamma_i V_{i+1,j} = V_{i,j+1}$$
$$\beta_i = dt(r + i^2\sigma^2)$$
$$\gamma_i = -\frac{1}{2}idt(r + i\sigma^2)$$

2.3.2 Matrice associée

On obtient la forme matricielle du problème :

$$\begin{pmatrix} 2\alpha_1 + 1 + \beta_1 & \gamma_1 - \alpha_1 & 0 & \cdots & 0 \\ \alpha_2 & 1+\beta_2 & \gamma_2 & \cdots & 0 \\ 0 & \alpha_3 & 1+\beta_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \gamma_{M-2} \\ 0 & 0 & 0 & \alpha_{M-1} - \gamma_{M-1} & 2\gamma_{M-1} + 1 + \beta_{M-1} \end{pmatrix} \begin{pmatrix} V_{1,j} \\ V_{2,j} \\ V_{3,j} \\ \vdots \\ V_{M-1,j} \end{pmatrix} = \begin{pmatrix} V_{1,j+1} \\ V_{2,j+1} \\ V_{3,j+1} \\ \vdots \\ V_{M-1,j+1} \end{pmatrix}$$

La résolution du schéma à chaque pas de temps revient donc à calculer la solution d'un système linéaire de la forme : $AX^{m+1} = b^m$ L'inconvénient de la méthode d'Euler implicite par rapport à la méthode explicite est l'apparente difficulté supplémentaire pour la résolution de ces systèmes linéaires afin de calculer la solution.

Une manière a priori simple de résoudre les systèmes linéaires consiste à inverser la matrice A, en remarquant qu'elle est constante au fil des pas de temps. Mais l'inverse d'une matrice tri diagonale est pleine, ce qui représente un cout de stockage important, au delà du cout de calcul de la matrice inverse. En temps normal, l'inverse d'une matrice de taille N nécessite $O(N^4)$ opérations. Le stockage de l'inverse de la matrice représente N^2 éléments, à comparer avec seulement $3N$ éléments dans la matrice A. [1]

2.3.3 Decomposition LU

La résolution d'un système linéaire peut se faire avec la traditionnelle méthode du pivot de Gauss, qui consiste 'à itérer les opérations sur les lignes ou colonnes de la matrice pour rendre le système triangulaire. La résolution d'un système linéaire triangulaire est alors t'es rapide. La matrice étant ici tri diagonale, il est plus intéressant d'utiliser une décomposition LU.

L'idée de la méthode LU consiste à dcomposer la matrice A sous la forme d'un produit $A = LU$, ou L est une matrice triangulaire inferieure (Lower), et U une matrice

1. https ://math.unice.fr/ auroux/EPU/MNPO.pdf

triangulaire supérieure (UPPER). Cette décomposition n'existe que sous conditions de rang et de non nullité des mineurs principaux de la matrice. Dans le cas de la matrice A présente, cette décomposition existe, et les matrices L et U n'ont que deux diagonales non nulles.https ://math.unice.fr/ auroux/EPU/MNPO.pdf

Nous utiliserons dans notre approche un module prédifini sur Python pour pouvoir obtenir la factorisation LU de la matrice A, le module s'appelle *scipy.linalg.lu* et permet rendre A sous la forme suivante : $A = PLU$ où P est une matrice de permutation, L triangulaire inférieur avec des éléments diagonaux unitaires et U triangulaire supérieur.

3 Résultats numériques :

Nous présenterons ce qu'on a eu comme résultat de résolution numérique de cette équation, en se basant sur deux schémas, l'explicite et l'implicite et nous essayerons de visualiser l'erreur commis pour chaque méthode en variant les paramètres de l'équation.

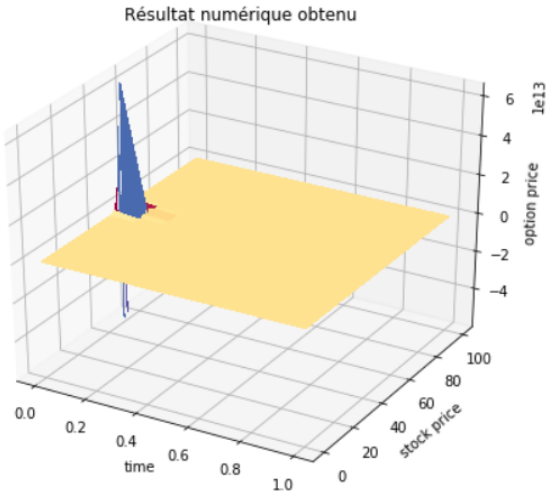

FIGURE 1 – Schéma explicite avec N=10

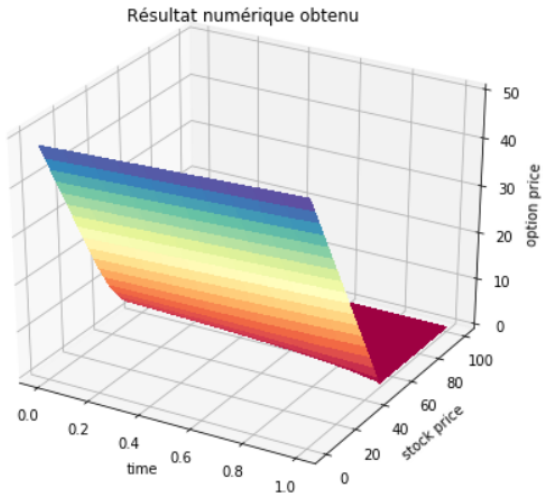

FIGURE 2 – Schéma implicite avec N=10

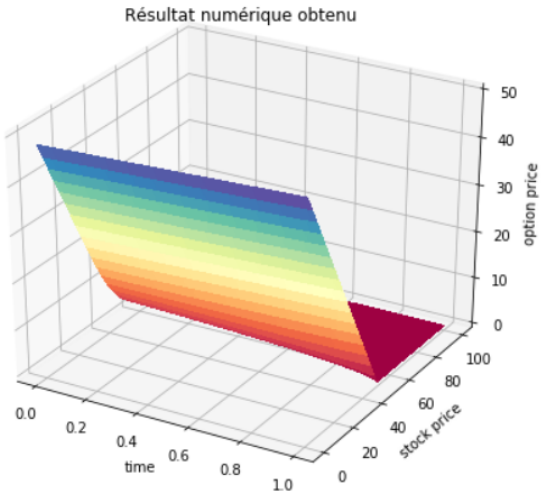

FIGURE 3 – Schéma explicite avec N=10000

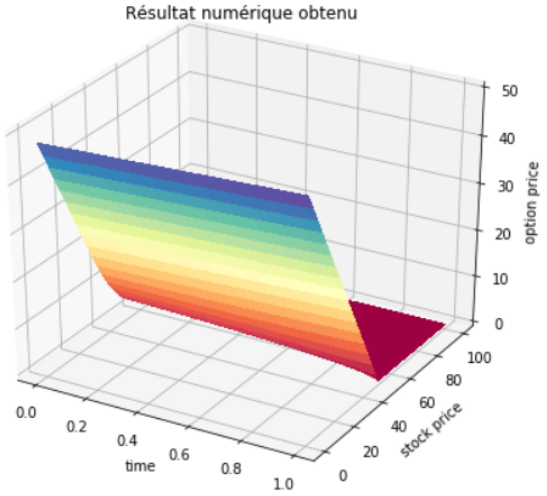

FIGURE 4 – Schéma implicite avec N=10000

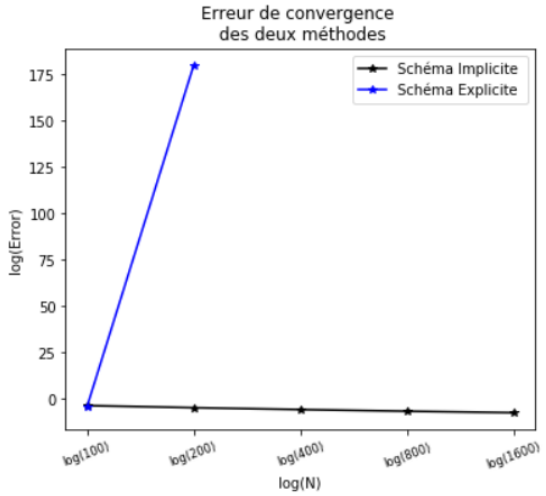

FIGURE 5 – Comparaison des deux méthodes en termes d'erreur en fct de N

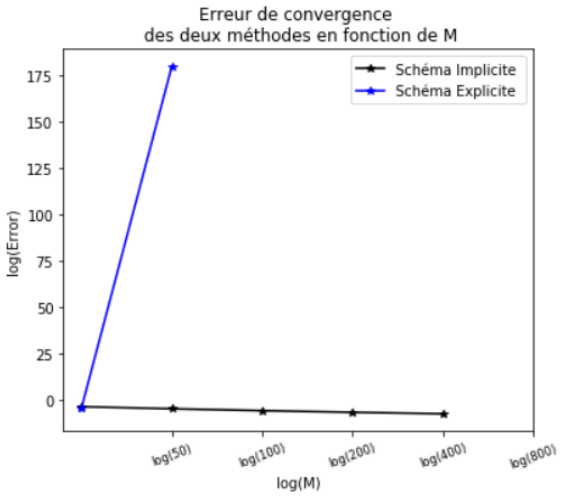

FIGURE 6 – Comparaison des deux méthodes en termes d'erreur en fct de M

4 Discussion des résultats :

La partie implémentation est réalisée sur Jupyter Notebook sur un ordinateur mené d'un processeur i7 7700 HQ à titre d'information. Le choix du schéma numérique le plus performant se fonde naturellement sur la facilité d'implémentation de la méthode numérique, stabilité, consistance,et convergence.

Le schéma explicite reste extrêmement facile à implémenter sur machine. Toutefois, la question de stabilité de ce schéma remet en question sur sa performance, puisqu'on constate que le schéma devient instables à N=10 et oscille d'une manière aléatoire.

Le schéma implicite reste un peu plus délicat à implémenter et à résoudre (passage par la factorisation LU). La stabilité de ce schéma n'a pas de lien avec le nombre de noeuds du maillage.

5 Conclusion :

Cette simulation nous a permis en premier temps de voir à l'oeil nue l'importance de la résolution numérique des équations dans les differents domaines. Parmi les autres points qu'on a pas pu mettre en évidence, c'est les inputs de l'équation qui découlent de la statistique. La loi normale ne décrit pas exactement le comportement de l'évolution des prix d'options, et ne caractérise pas l'imprévu. D'autres méthodes plus puissantes sont envisageables dans ce domaine et qui permettront d'avoir des résultats plus fiables et rationnels.

Liens utiles :

- https://fr.wikipedia.org/wiki/Modèle_Black-Scholes
- https://www.quantstart.com/articles/C-Explicit-Euler-Finite-Difference-Method-f
- https://github.com/NeilyWitches/Black-Scholes-PDE
- https://www.investopedia.com/terms/b/boundary-conditions.asp

In [2]:
```
# import des bibliothèques nécessaires
import numpy as np
from scipy.stats import norm # nous donne la fct de répartition qu'on a
ura besoin pour la sol analytique.
import scipy.linalg as linalg #décomposition LU
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D #Tracé 3D
```

In [7]:
```
# Phase de parametrage
S0 = 50
K = 50
r = 0.1
T = 5/12
sigma = 0.4
Smax = 100
M = 100  # S
N = 1000 # t
M, N = int(M), int(N)  # S'assurer que M et N se sont bien des entiers.
dS = Smax / float(M)
dt = T / float(N)
iValues = np.arange(1, M)
jValues = np.arange(N)
price = np.zeros(shape=(M+1, N+1)) # le prix de l'option sera une matri
ce de taille(M+1,N+1)
SValues = np.linspace(0, Smax, M+1)
price.shape
```

Out[7]:
```
(101, 1001)
```

In [8]:
```
jValues.shape
```

Out[8]:
```
(1000,)
```

In [10]:
```
#les coefficients de la matrice A
alpha = 0.5*dt * (sigma**2 * iValues**2 - r * iValues)
```

```
beta  = - dt * (sigma**2 * iValues**2 + r)
gamma = 0.5*dt * (sigma**2 * iValues**2 + r * iValues)
beta.shape
```

Out[10]: (99,)

In [12]:
```
# Contruction de la matrice tri diagonale associé au schéma explicite:
A= np.diag(alpha[1:], -1) + np.diag(1 + beta) + np.diag(gamma[:-1], 1)
A.shape
```

Out[12]: (99, 99)

In [38]:
```
#Call or put ?
#Si c'est un call:
price[:, -1] = np.maximum(SValues - K, 0)
#si c'est un put :
 #   price[:, -1] = np.maximum(K - SValues, 0)
```

In [ ]:
```
```

In [61]:
```
#Conditions aux limites type Neumann nous conduisent à ces équations:

A[0,   0] += 2*alpha[0]
A[0,   1] -= alpha[0]
A[-1, -1] += 2*gamma[-1]
A[-1, -2] -= gamma[-1]
```

In [15]:
```
# schéma explicite
def eulerexplicite(S0, K, r, T, sigma, Smax, m, n):
    alpha = 0.5*dt * (sigma**2 * iValues**2 - r * iValues)
    beta  = - dt * (sigma**2 * iValues**2 + r)
    gamma = 0.5*dt * (sigma**2 * iValues**2 + r * iValues)
    A = np.diag(alpha[1:], -1) + np.diag(1 + beta) + np.diag(gamma[:-1
], 1)
    A[0,   0] += 2*alpha[0]
    A[0,   1] -= alpha[0]
    A[-1, -1] += 2*gamma[-1]
```

```
    A[-1, -2] -= gamma[-1]
    #Résolution du sys
    for j in reversed(jValues):
        price[1:-1, j] = np.dot(A, price[1:-1, j+1])
        price[0, j] = 2 * price[1, j] - price[2, j]
        price[-1, j] = 2 * price[-2, j] - price[-3, j]
    return price
```

In [16]:
```
#schéma implicite et utilisation de la décompostion LU par linalg.lu
P, L, U = linalg.lu(A)
def eulerimplicite(S0, K, r, T, sigma, Smax, m, n):
    for j in reversed(jValues):
        Ux = linalg.solve(L, price[1:-1, j+1])
        price[1:-1, j] = linalg.solve(U, Ux)
        price[0, j] = 2 * price[1, j] - price[2, j]
        price[-1, j] = 2 * price[-2, j] - price[-3, j]
    return price
```

In [17]:
```
#Solution analytique dans le cas d'un call, on utilise la fct de répart
ition voire la solution analytique sur le doc
d1 = ((r + 0.5 * sigma**2) * T - np.log(K / S0)) / (sigma * np.sqrt(T))
d2 = ((r - 0.5 * sigma**2) * T - np.log(K / S0)) / (sigma * np.sqrt(T))
pcallana = S0 * norm.cdf(d1) - np.exp(-r * T) * K * norm.cdf(d2)
pcallana
```

Out[17]:
```
6.116508129330871
```

In [18]:
```
#Solution analytique dans le cas d'un put:
d1 = ((r + 0.5 * sigma**2) * T - np.log(K / S0)) / (sigma * np.sqrt(T))
d2 = ((r - 0.5 * sigma**2) * T - np.log(K / S0)) / (sigma * np.sqrt(T))
p = np.exp(-r * T) * K * norm.cdf(-d2) - S0 * norm.cdf(-d1)
p
```

Out[18]:
```
4.075980984787783
```

In [20]:
```
# Erreur de convergence d'un schéma par rapport à la solution analytiqu
e.
def convergeError(S0, K, r, T, sigma, Smax, m, n):
    p = np.array([])
    for M, N in zip(m, n):
        option=eulerexplicite(S0, K, r, T, sigma, Smax, m, n)
        p = np.append(p, option)
    pcallana = S0 * norm.cdf(d1) - np.exp(-r * T) * K * norm.cdf(d2)
    error = np.abs(pcallana-option)
    return error, p, pcallana
m = np.array([50, 100, 200, 400, 800])
n = np.array([100, 200, 400, 800, 1600])
```

In [ ]:
```
convergeError(S0, K, r, T, sigma, Smax, m, n)
```

In [ ]:
```
#tracé de la matrice price pbtenu par le schéma explicite
def plotpriceexp( S0, K, r, T, sigma, Smax, M, N):
    fig = plt.figure(figsize=(6,5))
    ax = Axes3D(fig)
    t, S = np.meshgrid(np.linspace(0, 1, N+1), np.linspace(0, Smax, M+1
))
    p = eulerexplicite(S0, K, r, T, sigma, Smax, m,n)

    ax.plot_surface(t, S, price, cmap='Spectral', linewidth=0, antialia
sed=False)
    ax.set_xlabel('time')
    ax.set_ylabel('stock price')
    ax.set_zlabel('option price')
    plt.show()

plotpriceexp( S0, K, r, T, sigma, Smax, M, N)
```

In [24]:
```
#tracé de la matrice price pbtenu par le schéma explicite
def plotpriceimp( S0, K, r, T, sigma, Smax, M, N):
    fig = plt.figure(figsize=(6,5))
    ax = Axes3D(fig)
    t, S = np.meshgrid(np.linspace(0, 1, N+1), np.linspace(0, Smax, M+1
```

```
))
    p = eulerimplicite(S0, K, r, T, sigma, Smax, m,n)
    ax.plot_surface(t, S, price, cmap='spectral', linewidth=0, antialia
sed=False)
    ax.set_xlabel('time')
    ax.set_ylabel('stock price')
    ax.set_zlabel('option price')
    plt.show()
```

In [ ]:

```
# tracé de l'erreur
fig = plt.figure(figsize=(6,5))
ax = fig.gca()
ax.plot(np.log(n), np.log(errorImp), '*-', c='black')
ax.plot(np.log(n), np.log(errorExplicitEu), '*-', c='blue')
ticks = ax.set_xticks(np.log(n))
labels = ax.set_xticklabels(['log(100)', 'log(200)',
                             'log(400)', 'log(800)',
                             'log(1600)'], rotation=20, fontsize='sm
all')
ax.set_xlabel('log(N)')
ax.set_ylabel('log(Error)')
ax.set_title('Erreur de convergence \n des deux méthodes')
plt.legend(['Schéma Implicite ', 'Schéma Explicite'])
plt.show()

#
```