Projets Maths

Paul Mazeau, Victorien Leconte, Théo Le Moal

1. Sujet

L'objectif est de modéliser un marché financier et de déterminer le prix et la couverture d'options européennes. Le marché financier est composé de deux actifs que l'on peut échanger à un prix fixé par le marché :

- un actif que l'on appelle sans risque connu dès l'instant initial 0 (ce prix est une variable aléatoire déterministe),
- un actif risqué dont le prix St à l'instant t est une variable aléatoire (typiquement une action), dont on modélisera la loi par la suite.

Un investisseur souhaite acheter à l'instant initial une option européenne qui lui rapportera la valeur $f(S_T)$ à l'instant T , où la fonction $f : R+ \to R+$ est continue (c'est une fonction qui prend en argument la valeur de l'actif risqué à l'instant T). Afin de déterminer le prix que doit faire payer le vendeur à l'instant initial à cet investisseur nous utiliserons des outils probabilistes. Nous souhaitons également déterminer la couverture que doit mettre en place le vendeur pour couvrir ses risques. La couverture correspond à la façon dont le vendeur investit dynamiquement l'argent qu'il reçoit à l'instant initial dans le marché pour qu'à l'instant T il ait exactement $f(S_T)$ à donner à l'acheteur du contrat sachant qu'à l'instant initial il n'a que ce qu'il reçoit de la 1 de la part de l'investisseur et qu'il ne rajoute jamais d'argent de sa poche et qu'il ne prend jamais d'argent pour lui. En resumé on cherche la proportion d'argent investie dans l'actif risqué et la proportion d'argent investie dans l'actif sans risque à n'importe quelle date pour qu'à la date T , lorsqu'on vend tout pour récupérer du cash, le vendeur ait exactement la somme $f(S_T)$. Pour résoudre ce problème nous modéliserons dans un premier temps l'évolution des prix de manière discrète avec une progression par arbre en suivant le modèle de Cox-Ross-Rubinstein, puis dans un second temps nous modéliserons l'évolution des prix de manière continue avec le modèle de Black-Scholes. Dans un troisième temps nous étudierons la convergence du prix donné par le modèle de Cox-Ross-Rubinstein vers celui donné par le modèle de Black-Scholes.

2. Modèle de Cox-Ross-Rubinstein (binomial)

Question 1

On recherche Q la probabilité risque-neutre tel que $E_Q(T_1^{(N)}) = 1 + r_N$. Pour cela on va développer l'ésperance, sachant que les variables aléatoires T_i^N prennent la valeur $1 + h_N$ avec la probabilité q_N et la valeur $1 + b_N$ avec la probabilité $1 - q_N$
On a donc :

$$E_Q(T_1^{(N)} = (1+h_N)q_N + (1+b_N)(1-q_N)$$
$$= 1 + r_N$$

On isole ensuite q_N et on trouve alors :

$$q_N = \frac{r_N - b_N}{h_N - b_N}$$

Question 2

D'après la définition, on a : $Prix_{Bin}^{(N)} = \frac{1}{(1+r_N)^N}\mathbb{E}_{\mathbb{Q}}[f(S_{t_N}^{(N)})]$
or on sait que $\forall i \in [0,N]$,$S_{t_i}^{(N)} = T^{(N)}S_{t_{i-1}}$ ainsi par récurrence on a : $S_{t_N}^{(N)} = T_{t_N}...T_{t_1}s$.
De plus $T_i = 1 + h_n$ ou bien $T_i = 1 + b_n$ ainsi $S_{t_i}^{(N)}$ peux prendre les valeurs: $s(1+h_n)^k(1+b_n)^{N-k}$ avec $k \in [0,N]$.
En posant $X_i = 1_{T_i = 1+h_N}, \forall i \in [0,N]$ tous les X_i suivent une loi de bernouilli de paramètre q_N.
De ce fait $\sum_{i=0}^N X_i$ suit une loi binomiale de paramètre (N, q_N).
L'esperance de $f(S_{t_N}^{(N)})$ est donc : $\mathbb{E}_{\mathbb{Q}}[f(S_{t_N}^{(N)})] = \sum_{i=0}^N f(x_i)\binom{N}{i}q_N^i(1-q_N)^{N-i}$ en posant $x_i = S(1+h_n)^i(1+b_n)^{N-i}$.
Le résultat finale est donc :$Prix_{Bin}^{(N)} = \frac{1}{(1+r_N)^N}\sum_{i=0}^N f(x_i)\binom{N}{i}q_N^i(1-q_N)^{N-i}$ avec $x_i = s(1+h_n)^i(1+b_n)^{N-i}$.

Premier pricer

Question 3

On doit créer une fonction *pricer1* : pour cela, on doit d'abord créer une fonction *coeff_binomiale* qui va nous permettre de calculer facilement le coefficient binomiale de N par i.
la fonction *pricer1* renvoie alors le prix de l'option en fonction des différents paramètres renseignés, determiné grâce au modèle binomiale.

In [9]:
```
import math as mt

def coeff_binomiale(n,p):
    a = mt.factorial(n)
    b = mt.factorial(p)
    c = mt.factorial(n-p)
    return a/(b*c)


def pricer1(N,rn,hn,bn,s,f):
    q = (rn-bn)/(hn-bn) # calcul de qn
    C = 1/pow((1+rn),N) # calcul de la constante devant la somme
    prix = 0
    for i in range(0,N+1):
        coef = coeff_binomiale(N,i) #calcul du coefficient binomial
        x_i= s*((1+hn)**i)*((1+bn)**(N-i))  #calcul de x_i
        a = q**i
        b = (1-q)**(N-i)
        prix += f(x_i)*coef*a*b
    prix*=C
    return prix
```

Question 4

In [10]:
```
def f(x):
    return max(x-110,0)

N=20
rn=0.02
hn=0.05
bn=-0.05
s=100
f=f

x=pricer1(N,rn,hn,bn,s,f)
print("le prix de l'option est : ", round(x,4))
```

```
le prix de l'option est :  26.6169
```

On teste notre premier pricer avec $f(x) = max(x-110,0)$, s = 100, h_N = 0.05, b_N = -0.05,
r_N = 0.02 et N = 20. La fonction nous renvoie le prix de l'option, qui ici est de **26,6169€**.

Deuxième pricer

Question 5

Pour comprendre comment implémenter le pricer 2, il faut dans un premier temps expliciter la formule de récurence associé au prix de l'option à la date t_k, c'est à dire la formule suivante :

$$v_k(S_{t_k}^N) = \frac{1}{1+r_N}E_Q(v_{k+1}(S_{t_{k+1}}^N)|S_{t_k}^N)$$

On sait que $S_{t_{k+1}}^N = T_{tk+1}S_{t_k}^N$ On va donc pouvoir d´evelopper l'espérance en posant (pour simplifier la lecture des calculs) $Y_k = S_{t_k}^N$. Ainsi on a donc:

$$E_Q(v_{k+1}(S_{t_{k+1}}^N)|S_{t_k}^N) = E_Q(Y_{k+1} = T_{t_{k+1}}Y_k|Y_k = y_k).$$

Détaillons maintenant les calculs:

$$E_Q(v_{k+1}(Y_{k+1} = T_{t_{k+1}}Y_k|Y_k = y_k)$$

$$= v_{k+1}((1+h_N)y_k)Q(Y_{k+1} = (1+h_N)y_k|Y_k = y_k)$$
$$+((1+b_N)y_k)Q(Y_{k+1} = (1+b_N)y_k|Y_k = y_k)$$

$$= v_{k+1}((1+h_N)y_k)Q(T_{k+1}^{(N)} = (1+h_N)y_k|Y_k = y_k)$$
$$+((1+b_N)y_k)Q(T_{k+1}^{(N)} = (1+b_N)y_k|Y_k = y_k)$$

$$= v_{k+1}((1+h_N)y_k)q_N + v_{k+1}((1+b_N)y_k)(1-q_N) \text{ par indépendance de } T_{k+1}^{(N)} \text{ et } Y_k$$

On va donc a chaque itération avoir un vecteur V_k dont les éléments seront les $v_k((1+h_N)^i(1+b_N)^{k-i}$.

L'initialisation sera alors le vecteur V_N, puis on va décrémenter l'indice jusqu'a 0, pour obtenir le prix de $p_{(N)}$.

Voici les deux vecteurs que l'on aura: $V_N = \begin{bmatrix} f(s(1+h_N)^N) \\ f(s(1+b_N)(1+h_N)^{N-1}) \\ ... \\ f(s(1+b_N)^N) \end{bmatrix}, V_K = \begin{bmatrix} v_k(s(1+h_N)^k) \\ v_k(s(1+b_N)(1+h_N)^{k-1}) \\ ... \\ v_k(s(1+b_N)^k) \end{bmatrix}$

On a donc comme formule vectorielle: $V_k\,[\,i\,] = \frac{1}{1+r_N}(q_N V_{k+1}\,[\,i+1\,] + (1-q_N)V_{k+1}\,[\,i\,])$

In [15]:
```
import numpy as np
def pricer2(N,rn,hn,bn,s,f):

    #initialisation des variables sous la forme d'un tableau
    tableau = np.zeros((N+1,N+1))
    for i in range(0,N+1,1):
        tableau[i][0] = f(s*((1+bn)**i)*((1+hn)**(N-i)))

    #calculs des vecteurs successifs
    const = 1/(1+rn)
    q = (rn-bn)/(hn-bn)
    for j in range(1,N+1,1):
        for i in range(j,N+1,1):
            tableau[i][j] = const*(q*tableau[i-1][j-1]+(1-q)*tableau[i][j-1])
    res = tableau[N][N]
    return res
```

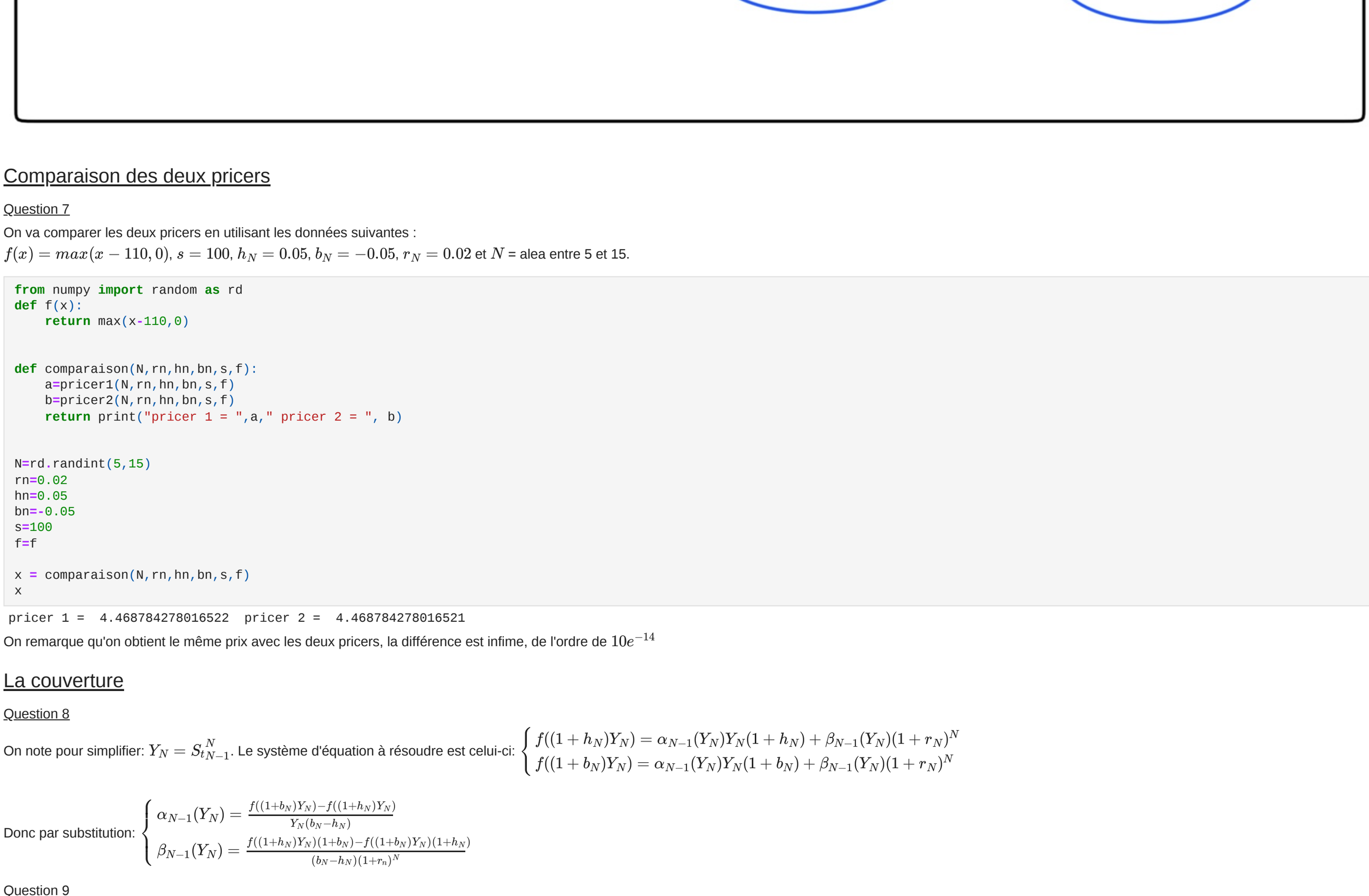
Question 6

In [12]:
```
def f(x):
    return max(x-100,0)

N=3
rn=0.02
hn=0.05
bn=-0.05
s=100
f=f

x=pricer2(N,rn,hn,bn,s,f)
print("le prix de l'option est : ", round(x,4))
```

```
le prix de l'option est :  7.0634
```

On teste notre deuxième pricer avec $f(x) = max(x-100,0)$, s = 100, h_N = 0.05, b_N = -0.05,
r_N = 0.02 et N = 3. La fonction nous renvoie le prix de l'option, qui ici est de **7,0634€**.

L'arbre des valeurs correspondant est celui-ci :



Comparaison des deux pricers

Question 7

On va comparer les deux pricers en utilisant les données suivantes :
$f(x) = max(x-110,0)$, $s = 100$, $h_N = 0.05$, $b_N = -0.05$, $r_N = 0.02$ et N = alea entre 5 et 15.

In [13]:
```
from numpy import random as rd
def f(x):
    return max(x-110,0)


def comparaison(N,rn,hn,bn,s,f):
    a=pricer1(N,rn,hn,bn,s,f)
    b=pricer2(N,rn,hn,bn,s,f)
    return print("pricer 1 = ",a," pricer 2 = ", b)


N=rd.randint(5,15)
rn=0.02
hn=0.05
bn=-0.05
s=100
f=f

x = comparaison(N,rn,hn,bn,s,f)
x
```

```
pricer 1 =  4.468784278016522  pricer 2 =  4.468784278016521
```

On remarque qu'on obtient le même prix avec les deux pricers, la différence est infime, de l'ordre de $10e^{-14}$

La couverture

Question 8

On note pour simplifier: $Y_N = S_{t_{N-1}}^N$. Le système d'équation à résoudre est celui-ci: $\begin{cases} f((1+h_N)Y_N) = \alpha_{N-1}(Y_N)Y_N(1+h_N) + \beta_{N-1}(Y_N)(1+r_N)^N \\ f((1+b_N)Y_N) = \alpha_{N-1}(Y_N)Y_N(1+b_N) + \beta_{N-1}(Y_N)(1+r_N)^N \end{cases}$

Donc par substitution: $\begin{cases} \alpha_{N-1}(Y_N) = \frac{f((1+b_N)Y_N) - f((1+h_N)Y_N)}{Y_N(b_N - h_N)} \\ \beta_{N-1}(Y_N) = \frac{f((1+h_N)Y_N)(1+b_N) - f((1+b_N)Y_N)(1+h_N)}{(b_N-h_N)(1+r_n)^N} \end{cases}$

Question 9

On note cette fois ci $Y_k = S_{t_{k-1}}^N$. Par le même principe on obtient:

$\begin{cases} \alpha_{N-1}(Y_K) = \frac{v_k((1+b_N)Y_k) - v_k((1+h_N)Y_K)}{Y_k(b_N - h_N)} \\ \beta_{N-1}(Y_K) = \frac{v_k((1+h_N)-Y_k)(1+b_N) - v_k((1+b_N)Y_k)(1+h_N)}{(b_N-h_N)(1+r_n)^N} \end{cases}$

Question 10

In [14]:
```
def f(x):
    return max(x-100,0)

hn=0.05
bn=-0.05
rn = 0.03
s=100
a = 1+hn
b = 1+bn
c=1/(1+rn)
qN = (rn-bn)/(hn-bn)

v1_hn = c*(f(s*(a**2))*qN+f(s*a*b)*(1-qN))
v1_bn = c*(f(s*a*b)*qN+f(s*(b**2))*(1-qN))

alpha0 = (v1_bn-v1_hn)/(s*(bn-hn))
beta0  = (v1_hn*(1+bn)-v1_bn*(1+hn))/((1+rn)*(bn-hn))
print("alpha0 = ",alpha0," et beta0 = ",beta0)

#Pour s(1+hn)
alpha1 = (f(s*(a**2))-f(s*a*b))/(s*a*(hn-bn))
beta1 = (a*f(s*b*a)-b*f(s*(a**2)))/(((1+rn)**2)*(hn-bn))
print("Pour St1 = (1+hn) on a alpha1 = ",alpha1," et beta1 = ",beta1)

#Pour s(1+bn)
alpha1_bis = (f(s*a*b)-f(s*(b**2)))/(s*b*(hn-bn))
beta1_bis = (a*f(s*(b**2))-b*f(s*a*b))/(((1+rn)**2)*(hn-bn))
print("Pour St1 = (1+bn) on a alpha1 = ",alpha1_bis," et beta1 = ",beta1_bis)
```

```
alpha0 =  0.7961165048543688  et beta0 =  -73.42822132151944
Pour St1 = (1+hn) on a alpha1 =  0.9761904761904762  et beta1 =  -91.78527665189932
Pour St1 = (1+bn) on a alpha1 =  0.0  et beta1 =  0.0
```

Concernant la couverture, avec $N = 2$, $s = 100$, $r_N = 0.03$, $h_N = 0.05$, $b_N = -0.05$, $f(x) = max(x-100,0)$, on obtient les résultats suivants :
A la date T_0 :

$$\alpha_0 = 0.7961 \text{ et } \beta_0 = -73.4282$$

A la date T_1 :

$$\alpha_1 = 0.9761 \text{ et } \beta_1 = -91.7852 \text{ pour } S_{t1} = (1+h_n)$$

$$\alpha_1 = 0.0 \text{ et } \beta_1 = 0.0 \text{ pour } S_{t1} = (1+b_n)$$

3. Modèle de Black-Scholes

Le modèle

Question 11

On a d $S_t = S_t(rdt + \sigma dB_t(1)$ et $dg(S_t) = g'(S_t)dS_t + \frac{|\sigma S_t|^2}{2}g''(S_t)dt(2)$.
On va appliquer (2) à $ln(S_t)$:

$$dg(ln(S_t)) = \frac{dS_t}{S_t} + \frac{|\sigma S_t|^2}{2}(-\frac{1}{S_t^2})dt$$
$$= \frac{dS_t}{S_t} + \frac{|\sigma^2 d_t|^2}{2}$$
$$= rdt + \sigma dB_t - \frac{|\sigma^2 d_t|^2}{2}$$
$$\frac{dS_t}{dt} = S_t(r - \frac{\sigma^2}{2} + \sigma\frac{dB_t}{ft})$$

On résout maintenant une équation différentielle du premier ordre sachant que $S_0(0) = s$. La solution est donc $S_t(t) = sexp((r - \frac{\sigma^2}{2})t + \sigma Bt)$.

Le pricer par la méthode de Monte-Carlo

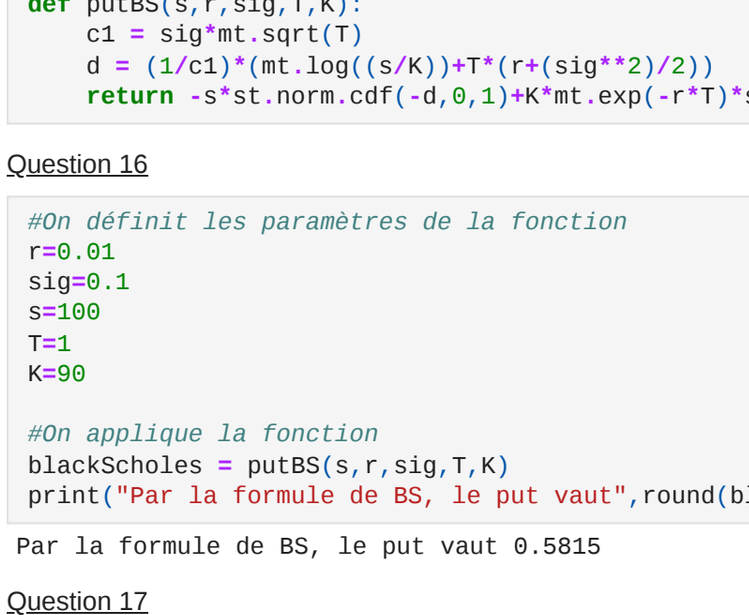
Question 12

Pour coder le pricer_MC on génère les variables aléatoires de loi normale N(0,1) avec la bibliothéque scipy.stats.

In [2]:
```
import numpy as np
import math as mt
import scipy.stats as st
import matplotlib.pyplot as plt

def pricer_MC(n,s,r,sig,T,f):
    xi = st.norm.rvs(0,1,n)
    c1 = mt.exp(-r*T)
    c2 = mt.exp(T*((r-(sig**2)/2)))
    c3 = sig*mt.sqrt(T)
    somme = 0
    for i in range(0,n,1):
        c4 = mt.exp(c3*xi[i])
        c = s*(c2*c4)
        somme += f(c)
    return (c1*somme)/n
```

Question 13

In [119…
```
#paramètre du test de la fonction
def f(x):
    return max(x-100,0)
r=0.01
sig=0.1
s=100
T=1
res = np.zeros(10)
n = (10**5)*np.arange(1,11,1)

#on teste la fonction avec les paramètres définis ci-dessus
res = [pricer_MC(i,s,r,sig,T,f) for i in n]

#On affiche un graphique du résultat obtenu par la fonction Pricer_MC
plt.plot(n,res)
plt.xlabel('n')
plt.ylabel('Prix')
plt.show()
```



Question 14

Pour montrer la convergence presque sûr vers p on va utiliser la loi forte des grands nombres (LFGD). Notons:

$$Z_i = exp(-rT)f(s * exp(T(r - \frac{\sigma**2}{2}) + \sigma\sqrt{T}\xi_i)).$$

Montrons l'indépendance des Z_i. Pour cela on va noter $\psi(\xi_i) = Z_i$ où ψ est une fonction croissante positive. Ainsi, on a:

$$P(\xi_i < x et \xi_j < y) = P(\psi(\xi_i) < \psi(x) et \psi(\xi_j) < \psi(y))$$

$$= P(\xi_i < x)P(\xi_j < y) \text{ par indépendance des } \xi$$

On a donc en composant par ψ:

$$P(\psi(\xi_i) < \psi(x) et \psi(\xi_j) < \psi(y)) = P(Z_i < \psi(x) et Z_j < \psi(y)) = P(\psi(\xi_i) < \psi(x))P(\psi(\xi_j < \psi(y))$$
$$= Z_i < \psi(x)P(Z_j < \psi(y)$$

Ainsi, les X_i sont bien indépendants et identiquement distribués (iid), et on peut appliquer LFGD, ce qui donne:

$$\frac{1}{n}\sum_{i=1}^n X_i \Rightarrow E(Z_i)$$

Or on sait que B_t et $\sqrt{T}\xi_i$ suivent la loi normale (O,T), donc avec la question 11, on a bien $f(s * exp(T(r - \frac{\sigma**2}{2}) + \sigma\sqrt{T}\xi_i))$ qui possède la même loi que f(S_T).
Ainsi, on a prouvé le résultat souhaité, soit la convergence presque sûr de p(n) vers p.

Le pricer par formule fermée

Question 15

In [4]:
```
def putBS(s,r,sig,T,K):
    c1 = sig*mt.sqrt(T)
    d = (1/c1)*(mt.log((s/K))+T*(r+(sig**2)/2))
    return -s*st.norm.cdf(-d,0,1)+K*mt.exp(-r*T)*st.norm.cdf(-d+c1,0,1)
```

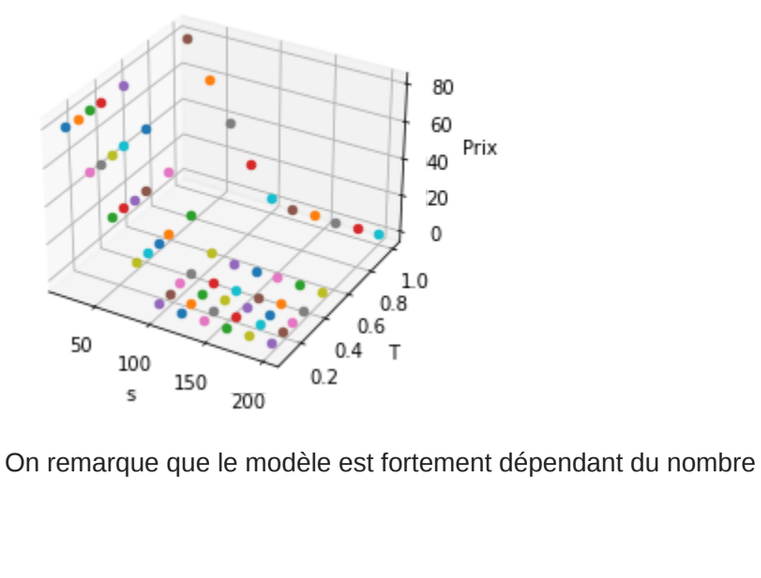
Question 16

In [12]:
```
#On définit les paramètres de la fonction
r=0.01
sig=0.1
s=100
T=1
K=90

#On applique la fonction
blackScholes = putBS(s,r,sig,T,K)
print("Par la formule de BS, le put vaut",round(blackScholes,4))
```

```
Par la formule de BS, le put vaut 0.5815
```

Question 17

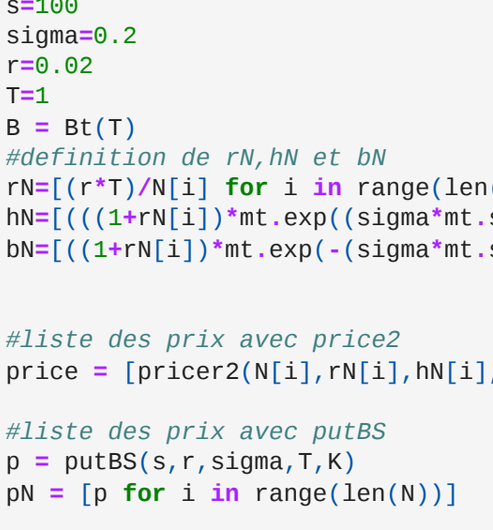
In [106…
```
def f(x):
    return max(90-x,0)
r=0.01
sig=0.1
s=100
T=1
res = np.zeros(10)
n = (10**5)*np.arange(1,11,1)

res = [pricer_MC(i,s,r,sig,T,f) for i in n]
bs =[blackScholes for i in range(0,10)]

#graphe de comparaison

plt.plot(n,res)
plt.plot(n,bs)
plt.xlabel('n')
plt.ylabel('Prix')
plt.show()
```



On remarque que le prix fournir par le pricer Monte Carlo converge vers le prix donné par Black-Scholes.

Question 18

In [36]:
```
from mpl_toolkits.mplot3d import Axes3D
from pylab import *
s = [20*i for i in range(1,11,1)]
T = [1/12,1/6,1/4,1/3,1/2,1]
r=0.01
sig=0.1
K=100

a=[]
#graphe 3D
fig = figure()
ax = fig.gca(projection='3d')
for i in range(len(s)):
    for j in range(len(T)):
        ax.scatter(s[i],T[j],putBS(s[i],r,sig,T[j],K))

ax.set_xlabel('s')
ax.set_ylabel('T')
ax.set_zlabel('Prix')
plt.show()
```

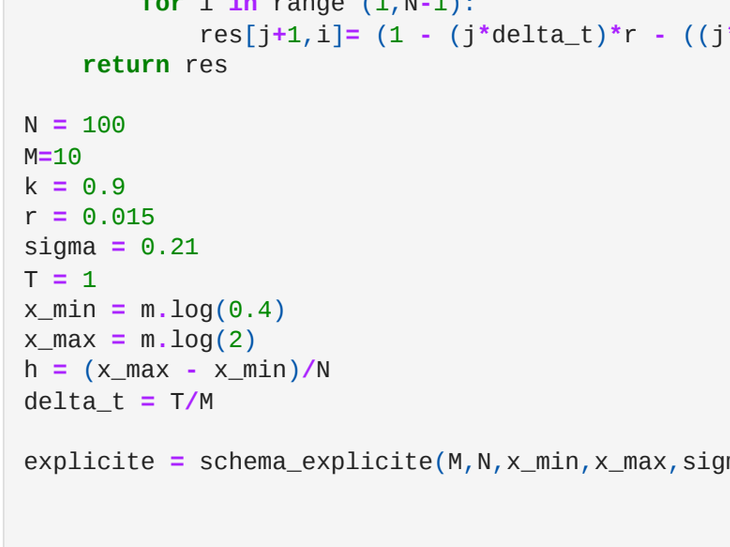


On remarque que le modèle est fortement dépendant du nombre d'itérations.

4. Convergence des prix

Question 19

In [120…
```
import random as rd
#mouvement brownien
def Bt(T):
    return rd.gauss(0,T)


def S(s,r,sig,t,Bt):
    c1 = (r-(sig**2)/2)*t
    c2 = sig*Bt
    return s*mt.exp(c1+c2)

#fonction max(100-sT,0)
def sT(t):
    T=1
    s=100
    r=0.02
    sig=0.2
    B=Bt(T)
    return max(100-S(s,r,sig,t,B),0)


N = [10*k for k in range(1,100,1)]
s=100
sigma=0.2
r=0.02
T=1
B = Bt(T)
#definition de rN,hN et bN
rN=[(r*T)/N[i] for i in range(len(N))]
hN=[(((1+rN[i])*mt.exp((sigma*mt.sqrt(T))/mt.sqrt(N[i])))-1) for i in range(len(N))]
bN=[((1+rN[i])*mt.exp(-(sigma*mt.sqrt(T))/mt.sqrt(N[i]))-1) for i in range(len(N))]


#liste des prix avec price2
price = [pricer2(N[i],rN[i],hN[i],bN[i],s,sT) for i in range(len(N))]

#liste des prix avec putBS
p = putBS(s,r,sigma,T,K)
pN = [p for i in range(len(N))]


plt.plot(N,price)
plt.plot(N,pN)
plt.xlabel('N')
plt.ylabel('Prix')
plt.show()
```



Notre code ne marche pas, on a un problème d'ordre de grandeur entre les deux prix

5. EDP de Black-Scholes

Question 20

explicite.png

implicite.png

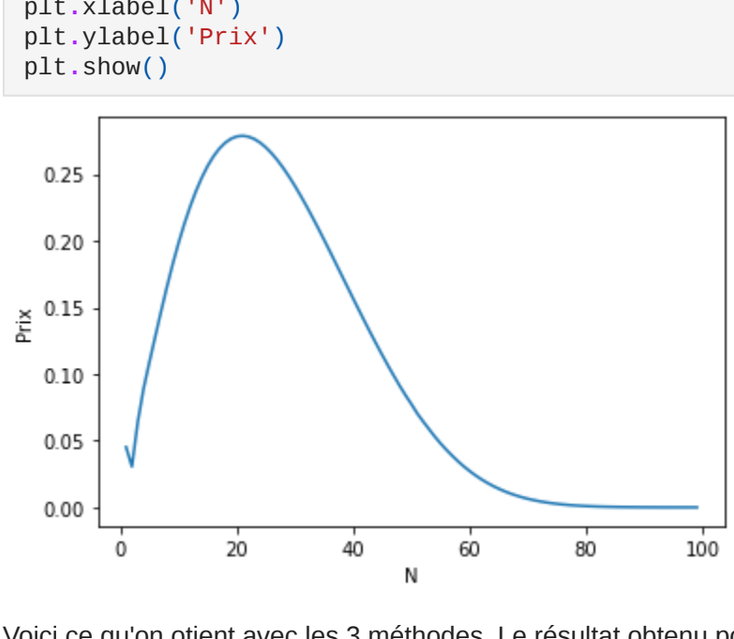
CN.png

In [111…
```
import math as mt
import numpy as np

def schema_explicite(M,N,x_min,x_max,sigma,r,k,T):
    res = np.zeros((M,N))
    h = (x_max - x_min)/N
    delta_t = T/M
    for i in range(0,M):
        res[i,0]= k * mt.exp(-r*(i*delta_t)) - mt.exp(x_min)
        res[i,N-1]= 0
    for i in range(0,N):
        res[0,i] = max(k - mt.exp(x_min+(i*h)),0)

    for j in range(0,M-1):
        for i in range (1,N-1):
            res[j+1,i]= (1 - (j*delta_t)*r - ((j*delta_t)*(sigma**2))/(h**2)) * res[j,i] + (((j*delta_t)*sigma**2)/(2*h**2) + (r*(j*delta_t))/(2*h) - ((j*delta_t)*(sigma**2))/(4*h)
    return res

N = 100
M=10
k = 0.9
r = 0.015
sigma = 0.21
T = 1
x_min = m.log(0.4)
x_max = m.log(2)
h = (x_max - x_min)/N
delta_t = T/M

explicite = schema_explicite(M,N,x_min,x_max,sigma,r,k,T)
```

In [112…
```
##methode implicite
def init_res(K,x_min,x_max,N):
    h = (x_max - x_min) / N
    P=np.zeros(N-1)
    for i in range(N-1):
        x=x_min+i*h
        P[i]=max(K - mt.exp(x),0)
    return P

def schema_implicite(M,N,x_min,x_max,sigma,r,k,T):
    delta_t = T/M
    h = (x_max - x_min) / N
    A=((sigma**2)*delta_t)/(4*h) - (r*delta_t)/(2*h) - ((sigma**2)*delta_t)/(2*(h**2))
    B=1+((sigma**2)*delta_t)/(h**2)
    C=(r*delta_t)/(2*h) - ((sigma**2)*delta_t)/(4*h)  - ((sigma**2)*delta_t)/(2*h**2)
    l=(1-r*delta_t)

    Amat=np.array([A for k in range (N-2)])
    Bmat= np.array([B for k in range (N-1)])
    Cmat=np.array([C for k in range (N-2)])

    Mmat= np.diag(Bmat) + np.diag(Amat,1) + np.diag(Cmat,-1)
    Mmat= np.linalg.inv(Mmat)

    res= init_res(k,x_min,x_max,N)
    i=1
    while (i<M+1):
        t=i*delta_t
        D=np.zeros(N-1)
        D[0]=C* (K * mt.exp(-r*(t+delta_t)) - mt.exp(x_min))
        D[N-2]=0
        res= l*(np.matmul(Mmat,np.add(res,-D)))
        i+=1

    return res

N = 100
M=100
k = 0.9
r = 0.015
sigma = 0.21
T = 1
x_min = m.log(0.4)
x_max = m.log(2)

implicite = schema_implicite(M,N,x_min,x_max,sigma,r,k,T)
```

In [113…
```
import math as mt
def schema_CN(M,N,x_min,x_max,sigma,r,k,T):
    delta_t = T/M
    h = (x_max - x_min) / N

    B =((sigma**2*delta_t)/(8*h) - (r*delta_t/(4*h)) - (sigma**2*delta_t)/(4*h**2))
    A =(1 + (sigma**2*delta_t)/(2*h**2))
    C =((r*delta_t)/(4*h) - (sigma**2*delta_t/(8*h)) - (sigma**2*delta_t)/(4*h**2))
    D =(1 - (r*delta_t) - (sigma**2*delta_t)/(2*h**2))

    A_tab = np.array([A for k in range(N-1)])
    B_tab =np.array([B for k in range(N-2)])
    C_tab =np.array([C for k in range(N-2)])

    B_tab_neg =np.array([-B for k in range(N-2)])
    C_tab_neg =np.array([-C for k in range(N-2)])
    D_tab =np.array([D for k in range(N-1)])

    F = np.diag(A_tab) + np.diag(B_tab,1) + np.diag(C_tab,-1)
    G = np.diag(D_tab) + np.diag(B_tab_neg,1) + np.diag(C_tab_neg,-1)
    F = np.linalg.inv(F)
    T = np.matmul(F,G)

    res = init_res(k,x_min,x_max,N)

    for m in range(1,M+1):
        t_m = delta_t * m
        V = np.zeros(N-1)
        V[0]= -C * (k* mt.exp(-r*t_m) - mt.exp(x_min))
        V[N-2]=0
        U = np.zeros(N-1)
        U[0]= C * (k*mt.exp(-r*(t_m+delta_t)) - mt.exp(x_min))
        U[N-2]=0
        res = np.matmul(T, np.add(res, np.add(-V, -U)))

    return res


N = 100
M=10
k = 0.9
r = 0.015
sigma = 0.21
T = 1
CN = schema_CN(M,N,x_min,x_max,sigma,r,k,T)
```

In [70]:
```
expl=explicite[9,]
Num=[n for n in range(0,100)]
plt.plot(Num,expl)
plt.xlabel('N')
plt.ylabel('Prix')
plt.show()
```



In [92]:
```
Num=[n for n in range(1,100)]
plt.plot(Num,implicite)
plt.xlabel('N')
plt.ylabel('Prix')
plt.show()
```


In [90]:
```
Num=[n for n in range(1,100)]
plt.plot(Num,CN)
plt.xlabel('N')
plt.ylabel('Prix')
plt.show()
```


Voici ce qu'on otient avec les 3 méthodes. Le résultat obtenu pour le schéma explicite nous amène qu'il y a une erreur, de stabilité ou alors de code. Les deux autres méthodes semblent très bien fonctionner.