ENSIIE

Promotion 2024

Rapport Projet Maths Info Semestre 2

Valentin Buchon, Eloïse Delhomelle, Yasmine Nait Kaci

Année 2021-2022

Table des matières

Partie I

Modèle de Cox-Ross-Rubinstein (binomial)

Sommaire

1 Préambule

Question 1

On sait que :

$$\mathbb{E}_{\mathbb{Q}} = (1+b_N)\mathbb{Q}(T_1^{(N)} = 1+b_N) + (1+h_N)\mathbb{Q}(T_1^{(N)} = 1+h_N) = 1+r_N$$

Or avec $b_N < r_N < h_N$ on pose également :

$$\begin{cases} q_N = \mathbb{Q}(T_1^{(N)} = 1+h_N) \\ 1-q_N = \mathbb{Q}(T_1^{(N)} = 1+b_N) \end{cases}$$

D'où avec l'égalité sur q_N et celle sur l'espérance :

$$q_N = \frac{(1+r_n) - (1+b_N)\mathbb{Q}(T_1^{(N)} = 1+b_N)}{1+h_N}$$

Ainsi en injectant la dernière égalité sur $1-q_N$ et en arrangeant :

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

Cette égalité fait sens avec les contraintes de définition, en particulier sur l'inégalité $b_N < r_N < h_N$.

Question 2

Le lemme de transfert nous donne :

$$\mathbb{E}\left[\phi(X)\right] = \sum_i \phi(x_i)\mathbb{P}(X = x_i)$$

Or d'après la définition de S_{t_i} on trouve :

$$f(S_{t_N}^{(N)}) = f(T_N^{(N)} S_{t_{N-1}}^{(N)}) = f(s \prod_{i=1}^{N} T_i^{(N)})$$

Ainsi en utilisant le lemme précédent :

$$\mathbb{E}_{\mathbb{Q}} \left[f(S_{t_N}^{(N)}) \right] = \sum_{k=0}^{N} f\left(s(1+h_N)^k (1+b_N)^{N-k}\right) \mathbb{Q}(S_{t_N} = s(1+h_N)^k (1+b_N)^{N-k})$$

Les événements étant idd. On a donc :

$$\mathbb{E}_{\mathbb{Q}} \left[f(S_{t_N}^{(N)}) \right] = \sum_{k=0}^{N} f\left(s(1+h_N)^k (1+b_N)^{N-k}\right) \binom{N}{k} q_N^k (1-q_N)^{N_k}$$

Nous avons finalement :

$$\text{Prix}_{\text{Bin}}^{(N)} = \frac{1}{(1+r_N)^N} \sum_{k=0}^{N} f\left(s(1+h_N)^k (1+b_N)^{N-k}\right) \binom{N}{k} q_N^k (1-q_N)^{N_k}$$

2 Premier Pricer

Question 3

Voici le pseudo-code :

Fonction Pricer1(N, r_N, h_N, b_N, s, f) :
  $q_N \leftarrow \dfrac{r_N - b_N}{h_N - b_N}$
  $T \leftarrow \text{TrianglePascal}(N)$
  $\Sigma \leftarrow 0$
  Pour $k = 0$ jusqu'à N, **faire** :
    $C_{N,k} \leftarrow T[N][k]$
    `nouveauF`$\leftarrow f(s(1+h_N)^k (1+b_N)^{N-k})$
    $\Sigma \leftarrow \Sigma +$`nouveauF`$\cdot C_{N,k} \cdot q_N^k \cdot (1-q_N)^{N-k}$
  Fin pour
Renvoyer $\dfrac{\Sigma}{(1+r_N)^N}$
Fin Fonction

Voici l'implémentation Python :

```python
def trianglePascal(n):
    T = [[0] * (n+1) for p in range(n+1)]
    for n in range(n+1):
        if n == 0:
            T[n][0] = 1
        else:
            for k in range(n+1):
                if k == 0:
                    T[n][0] = 1
                else:
                    T[n][k] = T[n-1][k-1] + T[n-1][k]
    return T



def pricer_1(N,rN,hN,bN,s,f):
    qN = (rN-bN)/(hN-bN)
    T = trianglePascal(N)
```

```
    somme = 0
    for k in range(0,N+1):
        binome = T[N][k]
        nouveauF = f(s*((1+hN)**k)*((1+bN)**(N-k)))
        somme += nouveauF*binome*(qN**k)*((1-qN)**(N-k))
    return (1/((1+rN)**N))*somme
```

Question 4

On trouve avec les valeurs données, Prix : 26.6169

3 Deuxième Pricer

Question 5

On a :

$$v_{N-1}(x) = \mathbb{E}_{\mathbb{Q}}\left[\frac{f(S_{t_N})}{1+r_N}|S_{t_{N-1}} = x\right]$$

D'où

$$v_{N-1}(x) = \mathbb{E}_{\mathbb{Q}}\left[\frac{f(xT_N)}{1+r_N}\right] = \frac{q_N f(x(1+h_N)) + (1-q_N)f(x(1+b_N))}{1+r_N}$$

Nous utiliserons donc l'algorithme suivant :

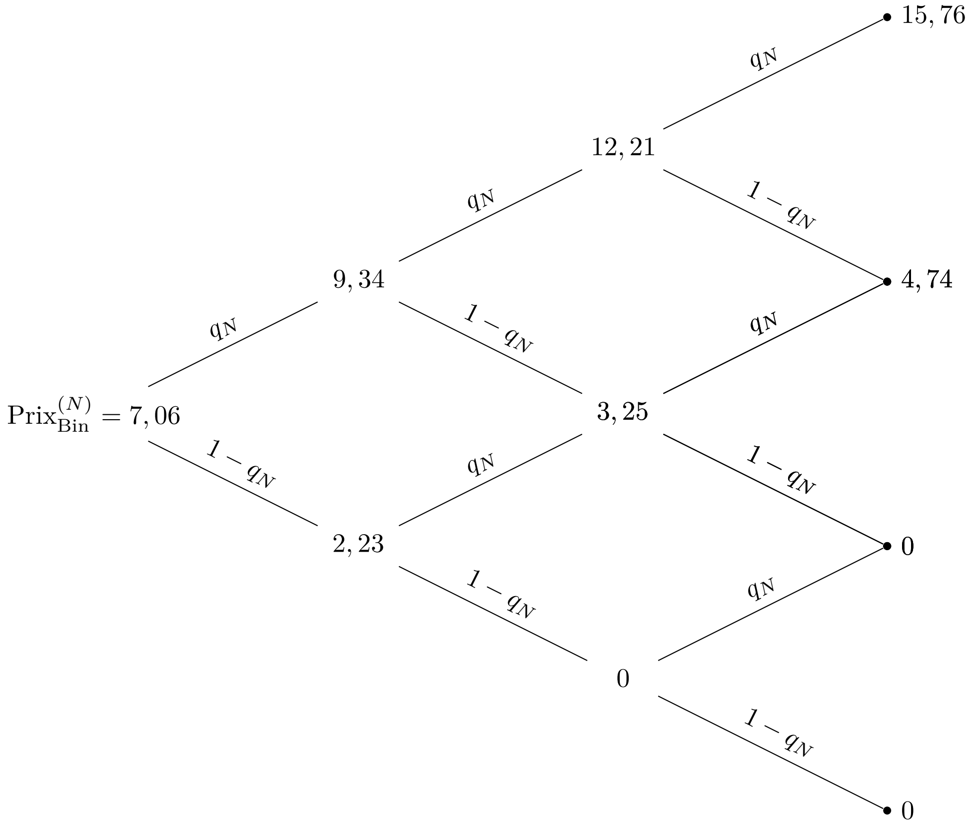
Fonction Pricer2(N, r_N, h_N, b_N, s, f) :
  $q_N \leftarrow \dfrac{r_N - b_N}{h_N - b_N}$
  `arbre`$\leftarrow (f(s((1+b_N)^k(1+h_N)^{N-k})))_{0\leqslant k<N}$
  Pour $k = N$ jusqu'à 0, **faire** :
    `aux`$\leftarrow [\,]$
    Pour $i = 0$ jusqu'à $k-1$, **faire** :
      $f_\uparrow \leftarrow$`arbre`$[-1][i]$
      $f_\downarrow \leftarrow$`arbre`$[-1][i+1]$
      $v_N \leftarrow \dfrac{1}{1+r_N} \cdot q_N \cdot f_\uparrow + (1-q_N)\cdot f_\downarrow$
      ajouter v_N à `aux`
    Fin pour
    ajouter `aux` à `arbre`
  Fin pour
Renvoyer `arbre`
Fin Fonction

Ainsi nous obtenons cette implémentation python :

```
def pricer_2(N,rN,hN,bN,s,f):
    qN = (rN-bN)/(hN-bN)
    tree = [[f(s*((1+bN)**k)*((1+hN)**(N-k))) for k in range(0,N+1)]]
    for n in range(N,0,-1):
        aux = []
        for i in range(0,n):
            fup = tree[-1][i]
            fdown = tree[-1][i+1]
            vn = (1/(1+rN))*(qN*fup+(1-qN)*fdown)
            aux.append(vn)
        tree.append(aux)
    return tree
```

Question 6

Voici l'arbre des $v_k(.)$:



4 Comparaison

Question 7

Pour n'importe quel N nous trouvons des valeurs similaires, les faibles écarts sont probablement dus aux opérations sur les flottants. Cependant, la première différence est que le *pricer*$_1$ est plus lent que le *pricer*$_2$. Le second problème avec *pricer*$_1$ est principalement le stockage d'entiers bien trop grands ainsi que les opérations effectuées sur ces derniers. En effet, il y a une erreur : `int too large to convert to float`, il faudrait donc utiliser d'autres bibliothèques comme `Decimal` pour y pallier.

5 Couverture

Question 8

En posant $x = S_{t_{N-1}}^{(N)}$ Nous avons comme système :

$$\begin{cases} f((1+h_N)x) = \alpha_{N-1}(x)(1+h_N)x + \beta_{N-1}(x)(1+r_N)^N \\ f((1+b_N)x) = \alpha_{N-1}(x)(1+b_N)x + \beta_{N-1}(x)(1+r_N)^N \end{cases}$$

Ainsi en isolant α_{N-1} on obtient :

$$\begin{cases} \alpha_{N-1}(x) = \dfrac{f((1+h_N)x) - \beta_{N-1}(x)(1+r_N)^N}{x(1+h_N)} \\ \beta_{N-1}(x) = \dfrac{f((1+b_N)x)(1+h_N) - f((1+h_N)x)(1+b_N)}{(1+r_N)^N(h_N - b_N)} \end{cases}$$

En injectant β_{N-1} dans la première équation, nous avons finalement comme solution :

$$\begin{cases} \alpha_{N-1}(x) = \dfrac{f((1+h_N)x)}{x(1+h_N)} - \dfrac{f((1+b_N)x)(1+h_N) - f((1+h_N)x)(1+b_N)}{x(1+h_N)(h_N-b_N)} \\ \beta_{N-1}(x) = \dfrac{f((1+b_N)x)(1+h_N) - f((1+h_N)x)(1+b_N)}{(1+r_N)^N(h_N-b_N)} \end{cases}$$

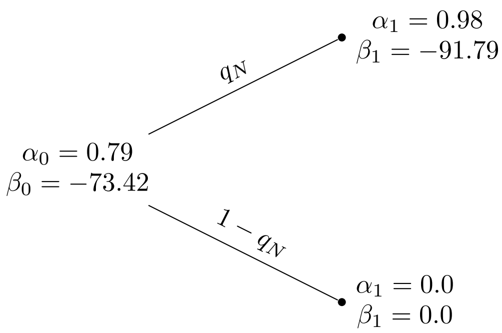
Question 9

La démonstration est identique à la question 8 en remplaçant le N par un k. La question précédente est un cas particulier de cette question où $N = k$. Ainsi on trouve :

$$\begin{cases} \alpha_{k-1}(x) = \dfrac{v_k((1+h_N)x)}{x(1+h_N)} - \dfrac{v_k((1+b_N)x)(1+h_N) - v_k((1+h_N)x)(1+b_N)}{x(1+h_N)(h_N-b_N)} \\ \beta_{k-1}(x) = \dfrac{v_k((1+b_N)x)(1+h_N) - v_k((1+h_N)x)(1+b_N)}{(1+r_N)^k(h_N-b_N)} \end{cases}$$

Question 10

Application des prochains algorithmes nous donne cette couverture :




```python
def beta(x,hN,bN,vku,vkd,rN,k):
    num = vkd*(1+hN)-vku*(1+bN)
    den = (hN-bN)*((1+rN)**k)
    return num/den

def alpha(x,hN,bN,vku,vkd,rN,k):
    num = vku-beta(x,hN,bN,vku,vkd,rN,k)*((1+rN)**k)
    den = x*(1+hN)
    return num/den
```

Ainsi en utilisant la fonction `pricer_2` et les deux précédentes nous avons en pseudo code :

Fonction Couverture(N, r_N, h_N, b_N, s, f) :
  `tableauVk` $\leftarrow$ Pricer2(N, r_N, h_N, b_N, s, f)
  renverser `tableauVk`
  $a \leftarrow \alpha(s, h_N, b_N,$ `tableauVk`$[1][0]$, `tableauVk`$[1][1], r_N, 1)$
  $b \leftarrow \beta(s, h_N, b_N,$ `tableauVk`$[1][0]$, `tableauVk`$[1][1], r - N, 1)$
  `arbre` $\leftarrow [[(s, a, b)]]$
  Pour $i = 1$ jusqu'à $N - 1$, **faire** :
    `aux` $\leftarrow [\ ]$
    Pour $j = 1$ jusqu'à longueur `arbre`$[-1]$, **faire** :
      $x \leftarrow$ Premier élément de `arbre`$[-1][j]$
      $vk_{\uparrow,1} \leftarrow$ `tableauVk`$[i+1][j]$
      $vk_{\downarrow,1} \leftarrow$ `tableauVk`$[i+1][j+1]$
      $vk_{\uparrow,2} \leftarrow$ `tableauVk`$[i+1][j+1]$
      $vk_{\downarrow,2} \leftarrow$ `tableauVk`$[i+1][j+2]$
      $x_\uparrow \leftarrow x \cdot (1 + h_N)$

$x_{\downarrow} \leftarrow x \cdot (1 + b_N)$
$a_{\uparrow} \leftarrow \alpha(x_{\uparrow}, h_N, b_N, vk_{\uparrow,1}, vk_{\downarrow,1}, r_N, i+1)$
$b_{\uparrow} \leftarrow \beta(x_{\uparrow}, h_N, b_N, vk_{\uparrow,1}, vk_{\downarrow,1}, r_N, i+1)$
$a_{\downarrow} \leftarrow \alpha(x_{\downarrow}, h_N, b_N, vk_{\uparrow,2}, vk_{\downarrow,2}, r_N, i+1)$
$b_{\downarrow} \leftarrow \beta(x_{\downarrow}, h_N, b_N, vk_{\uparrow,2}, vk_{\downarrow,2}, r_N, i+1)$
ajouter $(x_{\uparrow}, a_{\uparrow}, b_{\uparrow})$ à `aux`
ajouter $(x_{\downarrow}, a_{\downarrow}, b_{\downarrow})$ à `aux`
Fin pour
ajouter `aux` à `arbre`
Fin pour
Renvoyer `arbre`
Fin Fonction

Et l'implémentation Python nous donne :

```python
def couverture(N,s,rN,hN,bN,f):
    tabVk = pricer_2(N, rN, hN, bN, s, f)
    tabVk.reverse()
    a = alpha(s,hN,bN,tabVk[1][0],tabVk[1][1],rN,1)
    b = beta(s,hN,bN,tabVk[1][0],tabVk[1][1],rN,1)
    tree = [[(s,a,b)]]
    for i in range(1,N):
        aux = []
        for j in range(len(tree[-1])):
            x,aa,bb = tree[-1][j]
            vku1 = tabVk[i+1][j]
            vkd1 = tabVk[i+1][j+1]
            vku2 = tabVk[i+1][j+1]
            vkd2 = tabVk[i+1][j+2]
            xup = x*(1+hN)
            xdown = x*(1+bN)
            aup = alpha(xup,hN,bN,vku1,vkd1,rN,i+1)
            bup = beta(xup,hN,bN,vku1,vkd1,rN,i+1)
            adown = alpha(xdown,hN,bN,vku2,vkd2,rN,i+1)
            bdown = beta(xdown,hN,bN,vku2,vkd2,rN,i+1)
            aux.append((xup,aup,bup))
            aux.append((xdown,adown,bdown))
        tree.append(aux)
    return tree
```

Partie II

Modèle de Black-Scholes

Sommaire

1 Le modèle

Question 11

Nous avons :

$$\mathrm{d}S_t = S_t(r\mathrm{d}t + \sigma \mathrm{d}B_t)$$

En appliquant la loi d'Itô avec $g(x) = \ln(x)$ qui est bien $\mathscr{C}^2$. On obtient :

$$\begin{aligned}
\mathrm{d}\ln(S_t) &= \frac{1}{S_t}\mathrm{d}S_t - \frac{(\sigma S_t)^2}{2S_t^2}\mathrm{d}t \\
&= \frac{1}{S_t}(rS_t\mathrm{d}t + \sigma S_t \mathrm{d}B_t) - \frac{\sigma^2}{2}\mathrm{d}t \\
&= (r - \frac{\sigma^2}{2})\mathrm{d}t + \sigma \mathrm{d}B_t
\end{aligned}$$

D'où :

$$\frac{\mathrm{d}\ln(S_t)}{\mathrm{d}t} = r - \frac{\sigma^2}{2} + \sigma\frac{\mathrm{d}B_t}{\mathrm{d}t}$$

Ainsi sachant que $B_0 = 0$ et que $S_O = s$, en intégrant nous avons :

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

En passant à l'exponentielle, nous avons donc finalement :

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

2 Le pricer par la méthode de Monte-Carlo

Question 12

Voici le pseudo-code :

Fonction $\mathrm{PricerMC}(n, s, r, \sigma, T, f)$:
  $normal \leftarrow (x_i \in X \sim \mathcal{N}(0,1))_{[}0 \leqslant i \leqslant n]$
  $\Sigma \leftarrow 0$

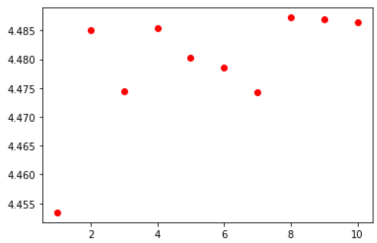
    Pour $i = 0$ jusqu'à n, **faire** :
        `evaluationF` $\leftarrow f(...)$
        $\Sigma \leftarrow \Sigma + \exp^{-r*T}$ `evaluationF`
    Fin pour
Renvoyer $\frac{\Sigma}{n}$
Fin Fonction

Voici l'implémentation Python :

```
def pricer_MC(n,s,r,sig,T,f):
    normale = np.random.normal(0,1,n)
    somme = 0
    for i in range(n):
        evalf = f(s*math.exp((r-(sig**2)/2)*T+sig*(T**0.5)*normale[i]))
        somme += math.exp(-r*T)*evalf
    return somme/n
```

Question 13

Voici un graphe utilisant la méthode de Monte-Carlo



Question 14

Si Z suit un processus de Wiener (mouvement Browien) standard alors la variation ∂B_t durant un court intervalle ∂t s'écrit :

$$\partial Z = \varepsilon\sqrt{\partial t}, \text{ où } \varepsilon \sim \mathcal{N}(0,1)$$

Ainsi :

$$\mathrm{d}\ln(S_t) = (r - \frac{\sigma^2}{2})\mathrm{d}t + \sigma\varepsilon\sqrt{\mathrm{d}t}$$

$r - \frac{\sigma^2}{2}$ et σ sont des constantes, donc $\ln(S_t)$ suit un processus de Wiener. Donc la variation $\ln(S_t)$ entre la date 0 et T suit une loi Normale :

$$\ln(S_t) \sim \mathcal{N}(\ln(S_0)(r - \frac{\sigma^2}{2})T, \sigma\sqrt{T})$$

Ainsi S_t suit une loi log-normale et on a :

$$S_T \overset{\text{loi}}{=} s\exp^{(r-\frac{\sigma^2}{2})T+\sigma\sqrt{T}\varepsilon}$$

Avec $S_0 = 0$ et $\varepsilon \sim \mathcal{N}(0,1)$.
Sachant que S_T est une variable aléatoire alors $\exp^{-rT} f(S_T)$ est une variable aléatoire. Or les $(\varepsilon_i)_{1\leqslant i\leqslant n}$ sont une suite de variables aléatoires indépendantes et identiquement distribuées de loi $\mathcal{N} \sim (0,1)$. On peut en déduire que $(Prix_{MC}^{(n)})_{n\in\mathbb{N}}$ converge presque surement vers $P = \mathbb{E}[\exp^{-rT} f(S_T)]$. Ceci découle directement de la Loi forte des grands nombres.

3 Le pricer par formule fermée

Question 15

Voici le pseudo-code avec $Y \sim \mathcal{N}(0,1)$:

Fonction PUTBS(s, r, σ, T, K) :
$d_1 = \frac{1}{\sigma * \sqrt{T}} \ln \frac{s}{K} + (r + \frac{\sigma^2}{2} T)$
$d_2 = d_1 - \sigma\sqrt{T}$
$F_1 \leftarrow \mathbb{P}(Y \leqslant -d_1)$
$F_2 \leftarrow \mathbb{P}(Y \leqslant -d_2)$
Renvoyer $-sF_1 + K \exp^{-rT} F_2$
Fin Fonction

Voici l'implémentation python :

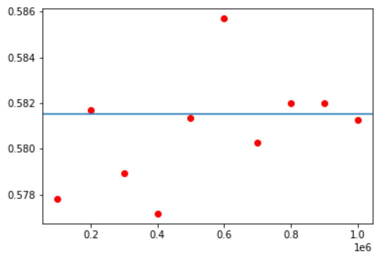
```
def put_BS(s,r,sig,T,K):
    d1 = (1/(sig*math.sqrt(T)))*(math.log(s/K)+(r+(sig**2)/2)*T)
    d2 = d1 - sig*math.sqrt(T)
    F1 = stats.norm.cdf(-d1,0,1)
    F2 = stats.norm.cdf(-d2,0,1)
    return -s*F1+K*math.exp(-r*T)*F2
```

Question 16

On trouve avec les valeurs proposées : $Prix_{BS} = 0.5815$

Question 17

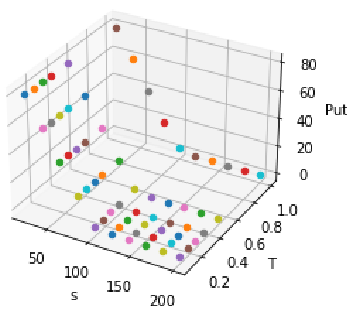
Voici le tracé de put et la méthode de Monte-Carlo

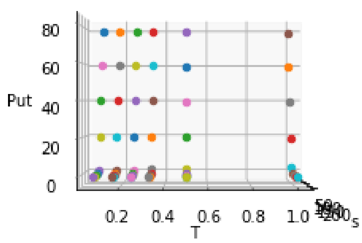


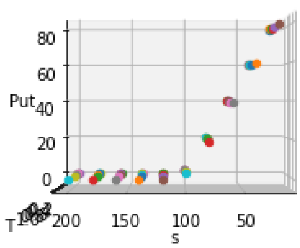
On observe la convergence de la méthode de Monte-Carlo vers la valeur réelle.

Question 18

Voici différents tracés du put lorsque s et T varient. On observe une décroissance exponentielle du put en s, mais une décroissance lente en T.







Partie III

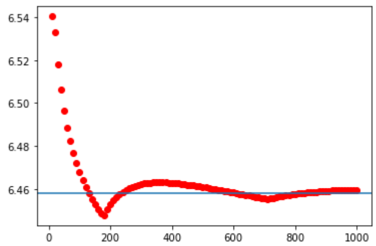
Convergence des prix

Sommaire

1 Application

Question 19

Avec les données proposées, nous obtenons ce graphique.



Il semble que $pricer_2$ converge vers le prix du put lorsque N devient significativement grand. Ainsi nous pouvons en déduire que le modèle de Cox-Rubinstein et le modèle de Black-Scholes covergent vers cette valeur.

Partie IV

EDP de Black-Scholes

Résolution

Question 20

Nous avons :

$$\frac{\partial p}{\partial t}(t,x) - \frac{1}{2}\sigma^2 \frac{\partial^2 p}{\partial x^2}(t,x) - (r - \frac{\sigma^2}{2})\frac{\partial p}{\partial x}(t,x) + rp(t,x) = 0$$

Avec $(t,x) \in [0,T] \times [x_min, x_max]$. En discrétisant l'espace comme dans le sujet et en utilisant les approximations pour un schéma explicite nous obtenons :

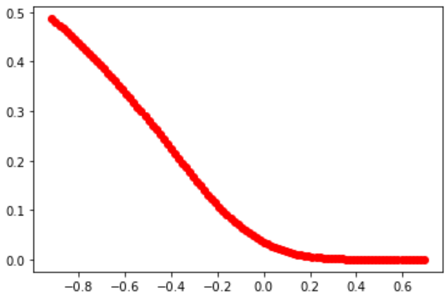
$$p(t_{m+1},x_j) = p(t_m,x_j)(1-r\Delta t) + \frac{\Delta t}{2}\frac{\sigma^2}{h^2}(p(t_m,x_{j+1}) + 2p(t_m,x_j) + p(t_m,x_{j-1})) + (r - \frac{\sigma^2}{2})\frac{\Delta t}{2h}(p(t_m,x_{j+1}) - p(t_m,x_{j-1}))$$

Ainsi avec l'algorithme suivant nous pouvons utiliser la méthode d'Euler explicite :

```
def euler_exp(K,r,sig,T,x_min,x_max,N,M):
    delta_t = T/M
    liste_t = [m*delta_t for m in range(0,M+1)]
    h = (x_max - x_min)/N
    liste_x = [x_min+j*h for j in range(0,N+1)]
    matrice_res = [[0 for _ in range(N+1)] for _ in range(M+1)]
    for i in range(N+1):
        matrice_res[0][i] = max(K - math.exp(liste_x[i]),0)
    for i in range(M+1):
        matrice_res[i][0] = K*math.exp(-r*liste_t[i])-math.exp(x_min)

    for i in range(1,M+1):
        for j in range(1,N):
            matrice_res[i][j] = matrice_res[i-1][j] \
            +delta_t*0.5*(sig**2)*(1/(h**2))*(matrice_res[i-1][j+1] \
            -2*matrice_res[i-1][j]+matrice_res[i-1][j-1]) \
            +delta_t*(r-(sig**2)/2)*(1/(2*h))*(matrice_res[i-1][j+1] \
            -matrice_res[i-1][j-1])-delta_t*r*matrice_res[i-1][j]
    return matrice_res[-1]
```

Nous obtenons donc :



Concernant la méthode d'Euler implicite, nous obtenons comme schéma :

$$\begin{cases} \dfrac{\partial p}{\partial t}(t_m, x_j) \approx \dfrac{1}{\Delta t}(p(t_m, x_j) - p(t_{m-1}, x_j)) \\ \dfrac{\partial p}{\partial x}(t_m, x_j) \approx \dfrac{1}{2h}(p(t_m, x_{j+1}) - p(t_m, x_{j-1})) \\ \dfrac{\partial^2 p}{\partial x^2}(t_m, x_j) \approx \dfrac{1}{h^2}(p(t_m, x_{j+1}) - 2p(t_m, x_j) + p(t_m, x_{j-1})) \end{cases}$$

En injectant cela dans notre équation de base :

$$\frac{\partial p}{\partial t}(t,x) - \frac{1}{2}\sigma^2 \frac{\partial^2 p}{\partial x^2}(t,x) - (r - \frac{\sigma^2}{2})\frac{\partial p}{\partial x}(t,x) + rp(t,x) = 0$$

Nous obtenons :

$$p(t_{m-1}, x_j) = ap(t_m, x_{j-1}) + bp(t_m, x_j) + cp(t_m, x_{j+1})$$

Avec

$$\begin{cases} a = -\dfrac{\sigma^2 \Delta t}{2h^2} + \dfrac{r\Delta t}{2h} - \dfrac{\sigma^2 \Delta t}{4h} \\ b = r\Delta t + \Delta t \dfrac{\sigma^2}{h^2} + 1 \\ c = -\dfrac{\sigma^2 \Delta t}{2h^2} - \dfrac{r\Delta t}{2h} + \dfrac{\sigma^2 \Delta t}{4h} \end{cases}$$

On pose $P_m = \begin{pmatrix} p(t_m, x_0) \\ \vdots \\ \vdots \\ \vdots \\ p(t_m, x_N) \end{pmatrix}$ et on pose également :

$$A = \begin{bmatrix} a & b & c & 0 & \dots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & a & b & c \end{bmatrix}$$

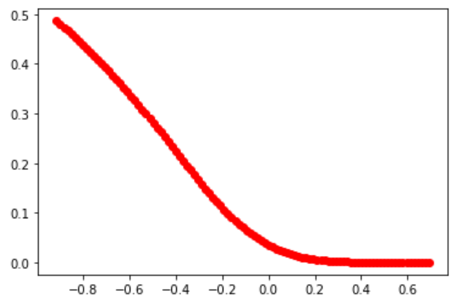
Ainsi nous avons le problème $P_{m-1} = APm$ à résoudre. En faisant bien attention lors de l'implémentation à donner les bonnes valeurs aux premiers et derniers coefficients de Pm.
Ainsi avec l'algorithme suivant nous pouvons utiliser la méthode d'Euler implicite :

```
def euler_imp(K,r,sig,T,x_min,x_max,N,M):
    delta_t = T/M
    liste_t = [m*delta_t for m in range(0,M+1)]
    h = (x_max - x_min)/N
    liste_x = [x_min+j*h for j in range(0,N+1)]
    Pm = [ max(K - math.exp(liste_x[i]),0) for i in range(0,N+1)]
    a = - (delta_t * (sig**2))/(2*(h**2)) + ((delta_t)*r)/(2*h) \
        - (delta_t*(sig)**2)/(4*h)
    b = r*delta_t + delta_t*(sig**2)*(1/(h**2)) + 1
```

```
    c = - (delta_t/2) * (sig/h)**2 - (delta_t *r)/(2*h) \
        + (delta_t*(sig)**2)/(4*h)
    A = [[0 for _ in range(N+1)] for _ in range(N+1)]
    for j in range(1,N):
        A[j][j-1] = a
        A[j][j] = b
        A[j][j+1] = c
    A[N][N] = 1

    for j in range(1,M+1):
        A[0][0] = (K*np.exp(-r*liste_t[j-1])-np.exp(x_min)) \
            /(K*np.exp(-r*liste_t[j])-np.exp(x_min))
        Pm = np.linalg.solve(A,Pm)
        Pm[0] = K*math.exp(-r*liste_t[j])-math.exp(x_min)
        Pm[-1] = 0
    return Pm
```

Nous obtenons donc :



Concernant le schéma de Crank-Nicholson nous avons :

$$\begin{cases} \dfrac{\partial p}{\partial t}(t_m, x_j) \approx \dfrac{1}{\Delta t}(p(t_{m+1}, x_j) - p(t_m, x_j)) \\ \dfrac{\partial p}{\partial x}(t_m, x_j) \approx \dfrac{1}{4h}(p(t_m, x_{j+1}) - p(t_m, x_{j-1}) + p(t_{m+1}, x_{j+1}) - p(t_{m+1}, x_{j-1})) \\ \dfrac{\partial^2 p}{\partial x^2}(t_m, x_j) \approx \dfrac{1}{2h^2}(p(t_m, x_{j+1}) - 2p(t_m, x_j) + p(t_m, x_{j-1}) + p(t_{m+1}, x_{j+1}) - 2p(t_{m+1}, x_j) + p(t_{m+1}, x_{j-1})) \end{cases}$$

En posant :

$$\begin{cases} a = \dfrac{\sigma^2}{4h^2} - (r - \dfrac{\sigma^2}{2})\dfrac{1}{4h} \\ b = \dfrac{1}{\Delta t} - \dfrac{\sigma^2}{2h^2} - r \\ c = \dfrac{\sigma^2}{4h^2} + (r - \dfrac{\sigma^2}{2})\dfrac{1}{4h} \\ a' = (r - \dfrac{\sigma^2}{2})\dfrac{1}{4h} - \dfrac{\sigma^2}{4h^2} \\ b' = \dfrac{1}{\Delta t} + \dfrac{\sigma^2}{2h^2} \\ c' = -(r - \dfrac{\sigma^2}{2})\dfrac{1}{4h} - \dfrac{\sigma^2}{4h^2} \end{cases}$$

Donc en combinant les éléments ci-dessus nous obtenons :

$$ap(t_m, x_{j-1}) + bp(t_m, x_j)) + cp(t_m, x_{j+1}) = a'p(t_{m+1}, x_{j-1})) + b'p(t_{m+1}, x_j) + c'p(t_{m+1}, x_{j+1})$$

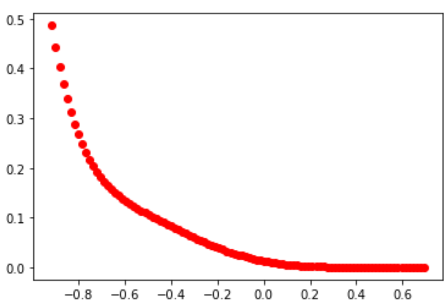
Nous devons donc résoudre un problème du type : $AP_{m+1} = BP_m$ de la forme :

$$\begin{bmatrix} a' & b' & c' & 0 & \dots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & a' & b' & c' \end{bmatrix} P_{m+1} = \begin{pmatrix} p(t_{m+1}, x_0) \\ ap(t_m, x_0) + bp(t_m, x_1) + cp(t_m, x_2) \\ \vdots \\ ap(t_m, x_{N-2}) + bp(t_m, x_{N-1}) + cp(t_m, x_N) \\ p(t_{m+1}, x_N) \end{pmatrix}$$

Ainsi avec l'algorithme suivant nous pouvons utiliser la méthode Crank-Nicholson :

```python
def CosM(K,r,sig,T,x_min,x_max,N,M):
    delta_t = T/M
    liste_t = [m*delta_t for m in range(0,M+1)]
    h = (x_max - x_min)/N
    liste_x = [x_min+j*h for j in range(0,N+1)]
    Pm = np.array([ max(K - math.exp(liste_x[i]),0) for i in range(0,N+1)])
    a = (sig**2)/(4*(h**2))-(r-(sig**2)/2)*(1/(4*h))
    b = (1/delta_t)-(sig**2)/(2*h**2)-r
    c = (sig**2)/(4*h**2)+(r-(sig**2)/2)*(1/(4*h))
    ap = -(sig**2)/(4*h**2)+(r-(sig**2)/2)*(1/(4*h))
    bp = (1/delta_t)+(sig**2)/(2*h**2)
    cp = -(sig**2)/(4*h**2)-(r-(sig**2)/2)*(1/(4*h))
    AP = np.array([[0 for _ in range(N+1)] for _ in range(N+1)])
    for j in range(1,N):
        AP[j][j-1] = ap
        AP[j][j] = bp
        AP[j][j+1] = cp
    AP[N][N] = 1
    AP[0][0] = 1
    for j in range(1,M+1):
        nPm = copy.deepcopy(Pm)
        nPm[0] = K*math.exp(-r*liste_t[j])-math.exp(x_min)
        nPm[-1] = 0 #p(t_j+1,0)
        for i in range(1,N):
            nPm[i] = a*Pm[i-1]+b*Pm[i]+c*Pm[i+1]
        Pm = np.linalg.solve(AP,nPm)
        Pm[0] = K*math.exp(-r*liste_t[j])-math.exp(x_min)
        Pm[-1] = 0
    return Pm
```

Nous obtenons donc :



Ce schéma n'est pas absurde, cependant il s'écarte beaucoup des 2 précédents, je suppose qu'il y a une erreur dans mon algorithme ou dans mes calculs.