

**TELECOM**  
SudParis



**IP PARIS**

GESTION DES RISQUES

---

Projets

---

2023-2024

Victor Gertner

# Table des matières

<b>1</b>	<b>Projet 3 : Pricing</b>	<b>1</b>
1.1	Presentation globale . . . . .	1
1.2	Modèle de diffusion : . . . . .	2
1.3	Pricing Vanille . . . . .	2
1.4	Pricing Option Tunnel : . . . . .	3
1.5	Pricing de l'Option Himalaya : . . . . .	4
1.6	Pricing de l'option Napoleon : . . . . .	5
1.7	Spécificité du code : . . . . .	7
1.7.1	Classe MonteCarlo . . . . .	7
1.7.2	Classe Form : . . . . .	8
1.7.3	Fichiers de code : . . . . .	8
<b>2</b>	<b>Projet 4 : Elements finis :</b>	<b>9</b>
2.1	Cadre Général : . . . . .	9
2.2	Décomposition du code : . . . . .	9
2.3	Résultats : . . . . .	11

# 1 Projet 3 : Pricing

## 1.1 Presentation globale

Dans le cadre de ce projet, nous devons fournir des formulaires pour le pricing de certaines options. C'est dans ce cadre que j'ai fait les 4 indiquées dans l'énoncé.

### Interface Principale :

L'interface principale, qui est dans le *main.py* permet de choisir quelle option nous souhaitons pricer.

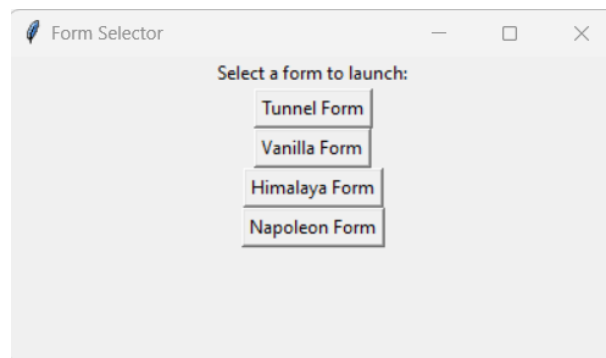


FIGURE 1 – Interface principale

Ainsi comme on le voit, on peut facilement choisir quel type d'action nous souhaitons pricer.

### Interface spécifique :

Ainsi une fois que nous cliquons sur un des pricers, voici ce qui s'affiche, ici le test est fait avec déjà des valeurs afin de montrer le produit fini. Nous avons donc la courbe du prix en fonction

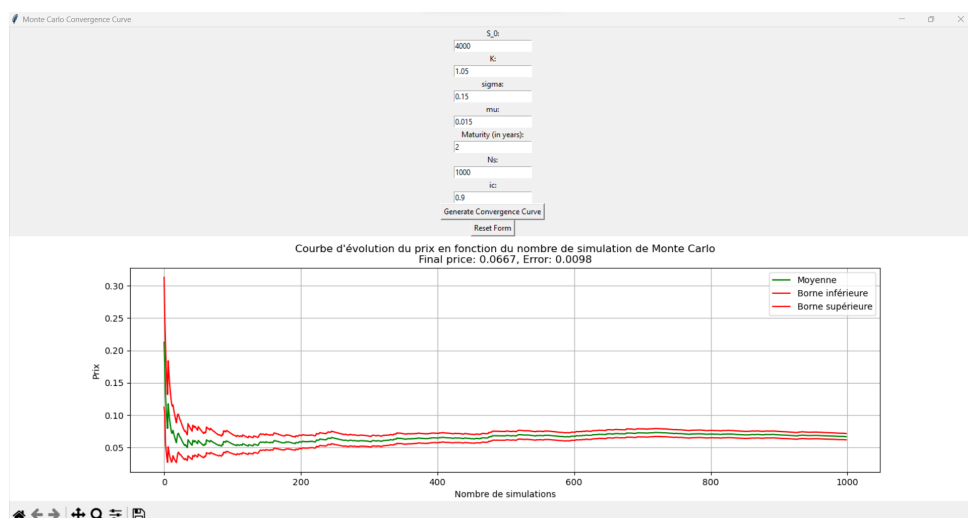


FIGURE 2 – Interface spécifique

du nombre de simulations mais aussi les bornes inférieure et supérieure de notre prix par rapport à la valeur d'intervalle de confiance que nous demandons dans le formulaire.

Dans le titre est affiche le prix final mais aussi l'erreur que l'on a ce prix.  
Il y a en bas (nécessité de se mettre en plein écran) la possibilité de zoomer et de se déplacer sur l'image mais aussi de la sauvegarder.

Il convient de rappeler que nous donnons le prix normalisé à la fin ie  $\frac{Prix}{S_0}$ .

## 1.2 Modèle de diffusion :

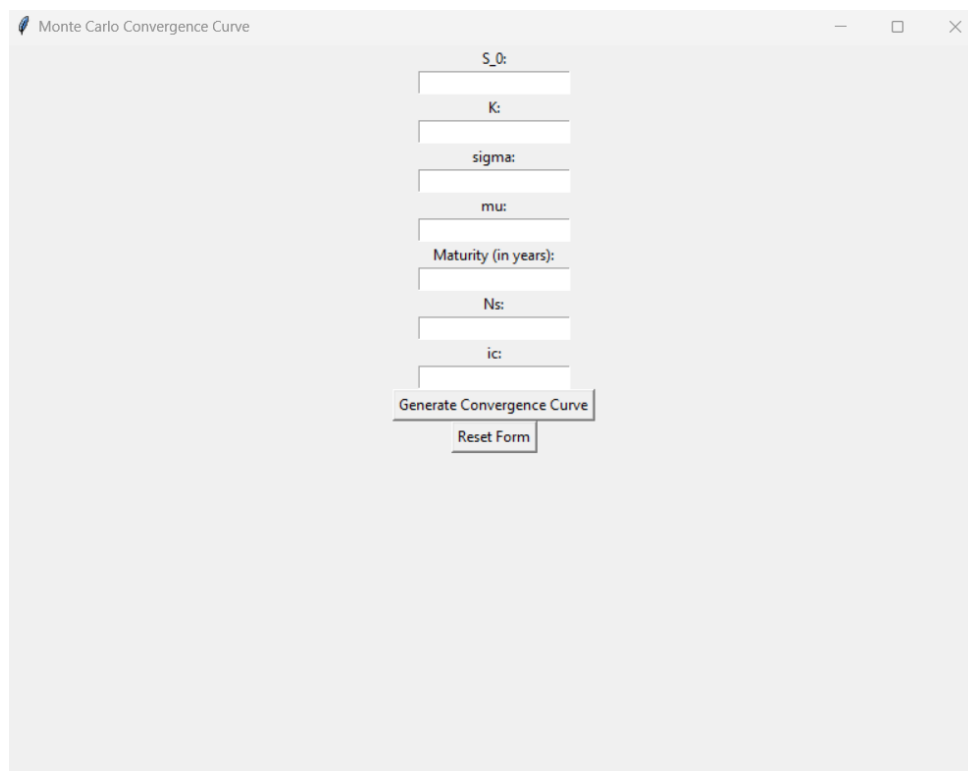
Dans le cadre de toutes nos options nous considéreront le modèle de diffusion :

$$S_t = S_{t-1} \exp\left(\left(\mu - \frac{\sigma^2}{2}\right)dt + \sigma\sqrt{dt}W\right) \quad (1)$$

Le processus W suit une loi normale centrée réduite.

## 1.3 Pricing Vanille

Dans le cadre d'un pricing vanille, il faut donc appuyer sur '*Vanille Form*' et voici l'interface qui s'affiche.



Monte Carlo Convergence Curve

S<sub>0</sub>:

K:

sigma:

mu:

Maturity (in years):

Ns:

ic:

Generate Convergence Curve

Reset Form

FIGURE 3 – Formulaire option Vanille

### Paramètres de l'options :

On remarque donc que plusieurs paramètres sont proposés en entrée :

- **S<sub>0</sub>** : C'est la valeur de départ de notre modèle de diffusion.
- **K** : C'est le strike de notre action, ie vous indique à quel prix vous pouvez acheter (dans le cas d'un call) ou vendre (dans le cas d'un put) le titre sous-jacent avant l'expiration du contrat.

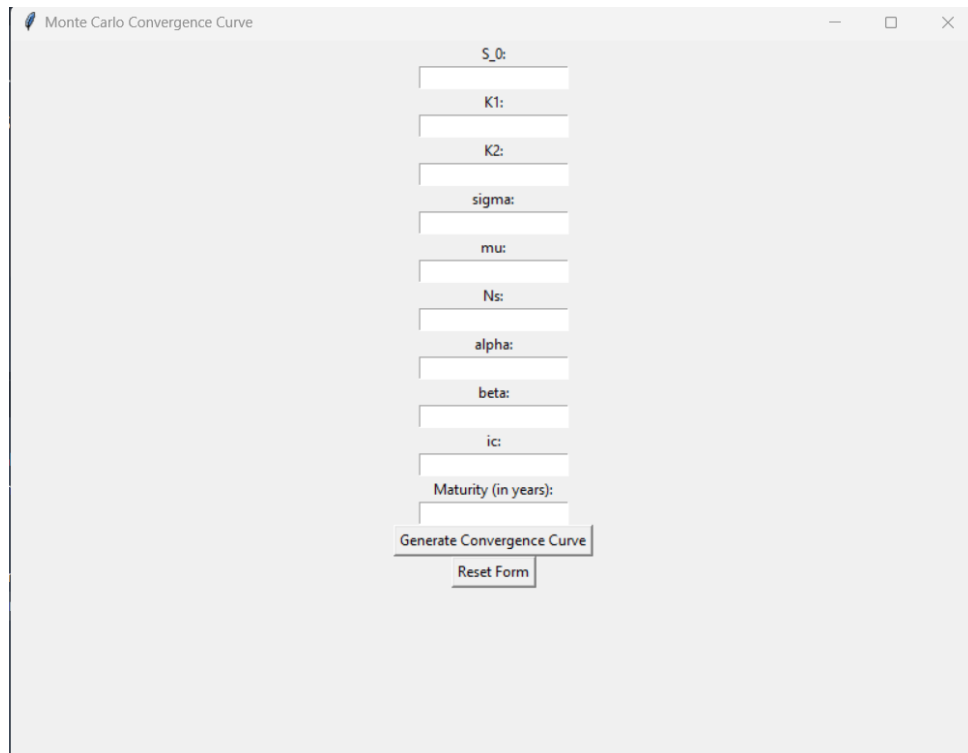
- **Sigma** : C'est la volatilité de l'action. Elle mesure la variation aléatoire des rendements de l'action par unité de temps. C'est la composante stochastique du modèle qui représente les fluctuations du prix de l'action autour de sa tendance moyenne.
- **Mu** : C'est le taux de rendement moyen attendu de l'action par unité de temps. Il est également appelé le rendement espéré ou la dérive (drift) de l'action. C'est la composante déterministe du modèle qui représente la tendance moyenne du prix de l'action dans le temps.
- **Maturity** : C'est le moment où l'on veut actionner ou non notre action.
- **Ns** : C'est le nombre de simulation de Monte Carlo que l'on veut effectuer.
- **ic** : C'est l'intervale de confiance que nous allons avoir autour de notre prix.

### Fonctionnement de l'option :

Dans le cadre d'une option Vanille, ici nous n'avons que considère le début : 2 premiers mois, et la fin : 2 derniers mois. C'est pourquoi dans notre code le vecteur  $t$  est défini de cette façon ! Et donc le payoff se mets sous cette forme : **Payoff = Max((Moyenne des 60 derniers jours)/(Moyenne des 60 premiers jours)-K, 0)**. Nous le faisons de cette façon afin d'éviter des surprises dans le paiement de l'option dû à une forte fluctuation inattendue sur un très court terme. Ainsi de cette façon, cette option Vanille est plus sécuritaire pour les deux parties.

## 1.4 Pricing Option Tunnel :

Dans le cadre d'un pricing Tunnel, il faut donc appuyer sur '*Tunnel Form*' et voici l'interface qui s'affiche :



Monte Carlo Convergence Curve

S<sub>0</sub>:

K1:

K2:

sigma:

mu:

Ns:

alpha:

beta:

ic:

Maturity (in years):

Generate Convergence Curve

Reset Form

FIGURE 4 – Formulaire Option Tunnel

### Paramètres de l'option :

Pour ce qui est des paramètres  $\mu$ ,  $\sigma$ ,  $S_0$ , Maturity,  $N_s$  et  $ic$ , il n'y a pas de changement par rapport à ce qui a été dit plus haut.

- **K1 et K2** : Ce sont les paramètres de notre Tunnel, avec K2 la valeur supérieure et K1 la valeur inférieure.
- **alpha** : C'est la valeur du coupon lorsque nous sommes dans le tunnel.
- **beta** : C'est la valeur du coupon lorsque nous sommes au dessus du tunnel.

**Fonctionnement de l'option Tunnel** Dans le cadre de l'option tunnel considéré en cours, nous regardons tous les 3 mois, c'est ce qui se passe ici. D'où la définition du vecteur temps.

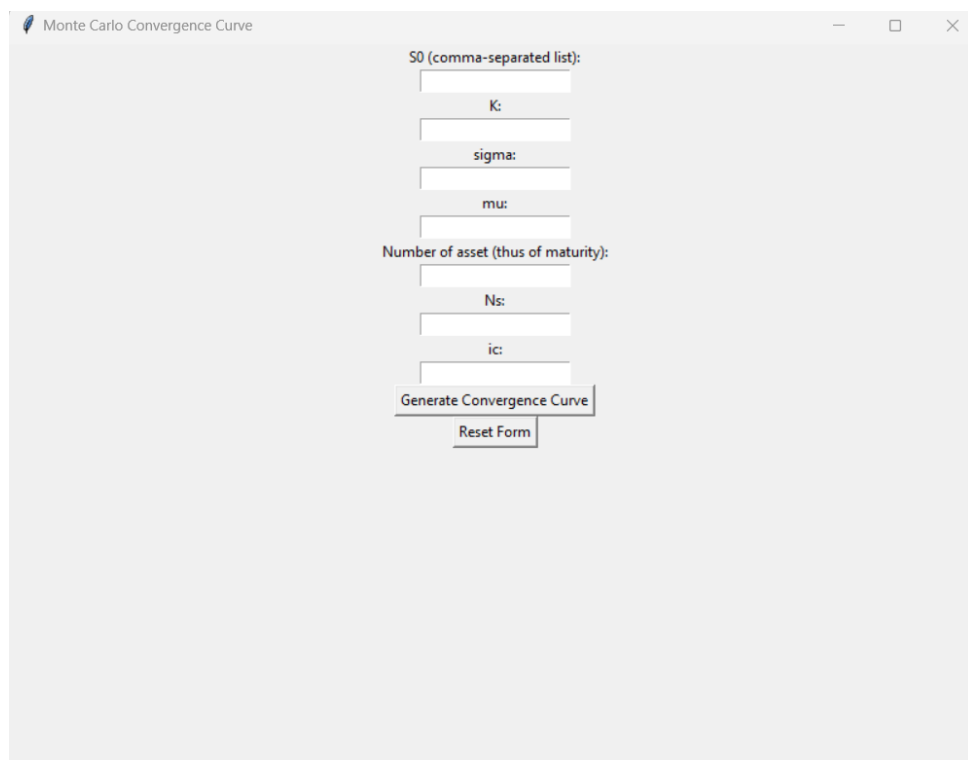
Ainsi, tous les trois mois, nous regardons la valeur de l'action ;

- Si elle est au dessus du tunnel, ie  $S_t > K_2$ , alors l'acheteur stocke un coupon ayant la valeur  $\frac{S_t}{S_0} * beta$ .
- Si elle est dans le tunnel, ie  $K_2 > S_t > K_1$ , alors l'acheteur stocke un coupon ayant la valeur  $\frac{S_t}{S_0} * alpha$ .
- Si elle est en dessous du tunnel, ie  $K_1 > S_t$ , alors l'acheteur perd tous les coupons qu'il a stocké auparavant.

Finalement le payoff est donnée par : Coupon +  $\max(\frac{S_n}{S_0} - K_2, 0)$  où  $n$  est l'année de maturité. Ainsi dans ce cas, même si l'option n'atteint jamais le strike final, l'acheteur peut toujours gagner de l'argent. C'est la différence fondamentale avec l'option Vanille.

## 1.5 Pricing de l'Option Himalaya :

Dans le cadre d'un pricing Himalaya, il faut donc appuyer sur '*Himalaya Form*' et voici l'interface qui s'affiche :



The image shows a web form titled "Monte Carlo Convergence Curve". It contains several input fields for parameters:   
 - "S0 (comma-separated list):" with an empty text box.   
 - "K:" with an empty text box.   
 - "sigma:" with an empty text box.   
 - "mu:" with an empty text box.   
 - "Number of asset (thus of maturity):" with an empty text box.   
 - "Ns:" with an empty text box.   
 - "ic:" with an empty text box.   
 Below the input fields are two buttons: "Generate Convergence Curve" and "Reset Form".

FIGURE 5 – Formulaire Option Himalaya

### Paramètres de l'option :

Tous les paramètres sont identiques à ce que nous avons pu avoir pour l'option Vanille, sauf que cette fois-ci nous ne donnons pas un seul  $S_0$  mais plusieurs cars nous avons plusieurs actions. Et la maturité est changée par le nombre d'actions, mais c'est la même chose dans notre cas.

### Fonctionnement de l'option Himalaya :

Dans le cadre de cette option, nous n'avons plus une seule action comme avant, mais un portefeuille de plusieurs actions. Chaque année nous regardons nos actions, et nous les classons par performances, puis nous ajoutons dans notre payoff cette valeur moins  $K$ . Puis nous enlevons cette action du portefeuille. Puis à la fin nous prenons le maximum entre cette accumulation de payoff et 0.

Ainsi on peut le définir rigoureusement par :

Pour un ensemble d'equities, on pose  $Y_1(t), Y_2(t), \dots, Y_d(t)$  leur prix respectif au temps  $t$ .

Soit  $t_1, \dots, t_n$  les temps de fixation de nos actions, ici nous considérons que nous allons lever toutes nos actions.

On note  $R_{i,j} = \frac{S_i(t_j)}{S_0(t_j)}$ . La performance de l'action  $i$ , au temps  $t_j$ .

Ainsi on a  $R_{k(i),i} = \max_{j \in 1, n \setminus \bigcup_{l=1}^{i-1} k(l)} R_{l,i}$ .

Enfin, on peut clairement écrire notre payoff :

$$Payoff = \max\left(\sum_{i=1}^n R_{k(i),i}, 0\right) \quad (2)$$

La logique derrière ce fonctionnement est que l'on peut se dire qu'une option qui a performé au maximum sur une période ne va jamais autant performer et donc on l'enlève de notre portefeuille d'action.

J'ai pris mes informations sur ce document : Himalaya Pricing

## 1.6 Pricing de l'option Napoleon :

Dans le cadre d'un pricing Napoleon, il faut donc appuyer sur '*Napoleon Form*' et voici l'interface qui s'affiche :

### Paramètres de l'option :

Tous les paramètres sont les mêmes que ceux de l'option Vanille, sauf que nous avons deux nouveaux paramètres qui sont un coupon et floor. Ce coupon est un entier (normalisé comme toujours), de même pour floor.

### Fonctionnement de l'option :

Cette option fonctionne encore différemment des autres. Soit  $t_1, \dots, t_{12n}$  les temps de maturité de nos actions, ici on a considéré que nous regardions tous les mois sur  $n$  années.

On pose donc  $R_{i,j} = \frac{S_{i+nj}}{S_{i-1+nj}} - 1$  comme étant la performance mensuelle de notre action au mois  $i$  pour l'année  $j$ . Elle est donc définie à chaque mois de chaque année.

Monte Carlo Convergence Curve

S<sub>0</sub>:

sigma:

mu:

Maturity (in years):

Ns:

ic:

Coupon:

Floor :

Generate Convergence Curve

Reset Form

FIGURE 6 – Formulaire pour option napoléon

Puis le payoff est définie chaque année comme :

$$Payoff_i = \max(0, C + \min_j(R_{i,j})) \quad (3)$$

Et donc le payoff global devient :

$$Payoff = \sum_{i=1}^n Payoff_i = \sum_{i=1}^n \max(0, C + \min_j(R_{i,j})) \quad (4)$$

On voit donc que l'acheteur, si la volatilité est faible a de fortes chances de recevoir le coupon, mais que s'il y a une forte volatilité il risque de ne rien gagner.

Une option dérivée de celle-ci est l'option de napoléon avec floor, ainsi le payoff de celle-ci devient :

$$Payoff = \sum_{i=1}^n Payoff_i = \sum_{i=1}^n \max(floor, C + \min_j(R_{i,j})) \quad (5)$$

Dans ce cas, s'il y a une forte volatilité, alors l'acheteur aura au moins le floor. C'est pour cela que nous avons mis floor dans notre pricing, car ce n'est qu'une généralisation de Napoléon, et il suffit de mettre le floor à 0 pour retrouver ce que nous avions avant.

J'ai pris mes informations sur ces liens :

Napoleon Pricing 1

Napoleon Pricing 2



## 1.7 Spécificité du code :

Nous avons décidé de faire deux classes dans notre code, une classe MonteCarlo, qui définit toutes les méthodes spécifiques à Monte Carlo et le pricing. Et une autre classe Form qui est relative à tout ce qui se passe par rapport au formulaire.

Nous allons faire une revue générale des fonctions définies dans notre code qui sont plus ou moins les mêmes dans tous nos formulaires. (Modulo le payoff qui peut changer mais l'idée générale est la même). Des commentaires plus exhaustifs sont présents dans le code.

### 1.7.1 Classe MonteCarlo

#### **`--__init__ :`**

Cette méthode permet d'initialiser les valeurs importantes de notre option, ces paramètres dépendent du type d'option.

#### **`generate_t :`**

Cette méthode permet de créer le vecteur temps qui nous permettra d'ensuite générer les trajectoires par rapport à ce vecteur.

#### **`simulate_gbm :`**

Cette méthode permet de simuler une trajectoire, elle se base donc sur le vecteur temps défini plus haut.

#### **`payoff :`**

Cette méthode permet de renvoyer le payoff d'une trajectoire. Elle est donc adaptée par rapport à l'option considérée.

#### **`simulate_monte_carlo :`**

Cette méthode est générale et permet de simuler un nombre choisi de trajectoire puis de faire le payoff de chacune d'entre elles.

#### **`convergence_mc :`**

Cette méthode est générale et permet d'effectuer la méthode de Monte-Carlo, et de faire l'intervalle de confiance qui a été spécifié par l'utilisateur.

#### **`generate_convergence_curve :`**

Cette méthode est générale et permet d'afficher notre simulation de Monte-Carlo, avec le prix et l'intervalle de confiance. On remarquera que le titre contient le prix de l'action mais aussi l'erreur.

### 1.7.2 Classe Form :

`__init__` :

Cette méthode permet d'initialiser notre formulaire, en particulier créer toutes les interactions dont nous aurons besoin pour l'utilisateur.

`displayconvergencecurve` :

Cette méthode permet d'afficher la courbe de convergence et de prendre les entrées de notre utilisateur.

`resetform` :

Cette méthode permet de supprimer l'ancien graphe pour pouvoir en afficher un nouveau.

`run` :

Cette méthode permet de lancer le formulaire.

### 1.7.3 Fichiers de code :

Nous avons donc en tout 5 fichiers, 1 pour chacune des options :

- **Vanilla form class** : Formulaire de l'option Vanille
- **Tunnel form class** : Formulaire de l'option Tunnel
- **Himalaya form class** : Formulaire de l'option Himalaya
- **Napoleon form class** : Formulaire de l'option Napoléon

Il y a ensuite un fichier **main**, celui-ci centralise les 4 autres fichiers afin de proposer un formulaire global pour choisir l'option que nous souhaitons pricer et lancer le bon formulaire.

## 2 Projet 4 : Elements finis :

### 2.1 Cadre Général :

Nous sommes donc dans le cadre général défini dans les diapositives du cours (diapositive 179). Ainsi dans notre algorithme nous allons définir, les éléments  $k$  et  $h$  qui permettent de discrétiser notre problème.

Puis nous sommes ensuite dans le cadre général de Crank-Nicholson et nous implémentons les formules pour les  $\alpha_i^m, \beta_i^m, \gamma_i^m$ . Puis nous définissons tous les éléments nécessaires pour définir notre problème.

Enfin nous définissons nos matrices  $\Lambda, \Xi_m, \epsilon_m$  a tout temps  $m$ , pour avoir le problème à résoudre. A partir de cela nous pouvons appliquer l'algorithme de Thomas. Il ne faudra pas oublier de mettre en place les conditions de bords proposées.

### 2.2 Décomposition du code :

Nous allons dans cette partie, décomposer le code de façon précise afin de bien comprendre notre implémentation.

Tout d'abord, nous avons défini toutes les variables fixes proposées dans l'énoncé.

#### Fonctions de base proposées :

Ensuite, nous avons mis en place toutes les fonctions de base que proposait l'énoncée ;

- **aProc(t,x)** : Retourne l'expression de  $a$  proposé dans le problème de black Scholes
- **bProc(t,x)** : Retourne l'expression de  $b$  proposé dans le problème de black Scholes
- **cProc(t,x)** : Retourne l'expression de  $c$  proposé dans le problème de black Scholes
- **dProc(t,x)** : Retourne l'expression de  $d$  proposé dans le problème de black Scholes
- **tminBound(t,x)** : Retourne la condition de bord quand nous sommes au tmin de la grille.
- **xminBound(t,x)** : Retourne la condition de bord quand nous sommes au xmin de la grille.
- **xmaxBound(t,x)** : Retourne la condition de bord quand nous sommes au xmax de la grille.
- **DxmaxBound(t,x)** : Retourne la condition de dérivée maximale.

#### Discretisation du problème :

Dans la cellule suivante, nous avons mis en place les fonctions pour discrétiser le problème :

- **k(t,x)** : retourne le pas en temps de notre problème.
- **h(t,x)** : retourne le pas spatial de notre problème.
- **t\_m(t,x)** : retourne le temps  $t_m$  tel qu'il est défini dans le cours.
- **x\_i(t,x)** : retourne la position  $x_i$  telle qu'elle est définie dans le cours.

#### Paramètres du problème :

Nous implantons ensuite les paramètres de la slide 187 :

- **alphaProc** : Retourne l'expression de  $\alpha$  a partir de toutes les fonction définies plus haut.

- **betaProc** : Retourne l'expression de  $\beta$  à partir de toutes les fonction définies plus haut.
- **gammaProc** : Retourne l'expression de  $\gamma$  à partir de toutes les fonction définies plus haut.

Puis une fois que ceux-ci sont définis, nous pouvons donc définir les paramètres de la diapositive 188 :

- **ksiProc(t,x)** : Retourne l'expression de  $\varsigma$  à partir de toutes les fonctions définies plus haut.
- **tauProc(t,x)** : Retourne l'expression de  $\tau$  à partir de toutes les fonctions définies plus haut.
- **nuProc(t,x)** : Retourne l'expression de  $\nu$  à partir de toutes les fonctions définies plus haut.
- **phiProc(t,x)** : Retourne l'expression de  $\Phi$  à partir de toutes les fonctions définies plus haut.
- **fiProc(t,x)** : Retourne l'expression de  $\phi$  à partir de toutes les fonctions définies plus haut.
- **khiProc(t,x)** : Retourne l'expression de  $\chi$  à partir de toutes les fonctions définies plus haut.
- **psiProc(t,x)** : Retourne l'expression de  $\psi$  à partir de toutes les fonctions définies plus haut.

### Système linéaire :

Maintenant que nous avons tous ces paramètres-là de défini, nous pouvons donc regarder notre système et le définir.

Pour cela, nous nous basons sur les slides 190, 191 et 192.

- **epsilon\_m(t,x)** : Permet de créer le vecteur  $\epsilon_m$  tel qu'il est défini dans le cours. On se base donc sur les conditions limites de notre modèle pour les valeurs de  $u$  en 0 et  $N_x - 1$ .
- **lambda\_m(t,x)** : Permet de crée la matrice  $\Lambda_m$  telle qu'elle est définie dans le cours. On se base donc sur les fonctions définies juste au dessus pour la créer.
- **omega\_m** : Permet de créer la matrice  $\Xi$  telle qu'elle est définie dans le cours. On se base donc sur les fonctions définies juste au dessus pour la créer.
- **right\_hand(t,x,u\_m\_1)** : Permet de retourner le coté droit de notre système, c'est à dire :

$$-[\Xi_{m-1} * u_{m-1} + \psi_m] + \epsilon_m \quad (6)$$

**Algorithme de Thomas** : Ainsi nous pouvons ensuite définir l'algorithme de Thomas tel qu'il a pu être défini dans le cours. C'est à ce moment la, que nous appliquons à la sortie ici le vecteur  $x$  qui représente  $u_m$  la condition limite de la dérivée. Ainsi la fonction **thomas\_algorithm(a,b,c,d)** applique l'algorithme de thomas dans la cadre général, avec  $a$  la fin la condition de la dérivée.

C'est aussi dans le cadre de l'algorithme de Thomas, que nous avons défini la fonction **get\_diagonals(matrix)** qui revoit la diagonale, diagonale inférieure et diagonale supérieure d'une matrice  $matrix$  passée en entrée.

### Conditions initiales :

Nous définissons ensuite les conditions initiale du vecteur  $u_0$  à partir de la fonction **tmin-Bound** définie plus haut. (Sans prendre en compte les condition de bord de  $x$  qui sont contenue dans la suite. Les vecteurs  $u_m$  sont donc de taille  $N_x - 2$ .)

## Résolution de l'équation de Black Scholes :

Arrive enfin le moment où nous mettons tout en place pour résoudre l'équation de Black Scholes. Ainsi celle-ci est résolue dans la fonction **resolve**.

Cette fonction commence par créer une grille de taille  $(Nt-1, Nx)$ , au départ rempli de 0. Tout d'abord nous appliquons les conditions de bord pour  $x_{min}$ , qui sont ici d'être égal à 0.

Puis nous appliquons l'autre condition de bord qui est celle à  $x_{max}$ , ici nous demandons que lorsque  $x$  s'approche de l'infini, alors la valeur de l'option s'approche de  $S-K$ , ie ici  $x_{max}-K = 50$ .

Puis nous mettons la valeur de  $u_0$  déjà définie plus haut dans la grille.

Ensuite, nous pouvons grâce à toutes les fonctions définies plus haut définir itérativement notre grille ie si nous sommes à l'étape  $i$  : on applique la fonction **get\_diagonals** sur la fonction **lambda\_m(t\_m(i,0),x\_i(0,i))**. Cela nous donne 3 vecteurs : **diag**, **supdiag** et **subdiag**. Ces vecteurs vont donc ensuite être fournis à la fonction **thomas\_algorithm** avec la fonction **right\_hand(t\_m(i,0),x\_i(0,i),u\_m\_1)** qui est aussi fournie à **thomas\_algorithm**. Cela nous donne donc la nouvelle valeur de  $u_i$  que nous ajoutons dans la grille, ensuite nous définissons  $u_{m_1}$  comme étant  $u_1$  : c'est le côté itératif de notre algorithme.

## 2.3 Résultats :

Ainsi voici la courbe 3D que nous avons obtenue suite à l'algorithme :

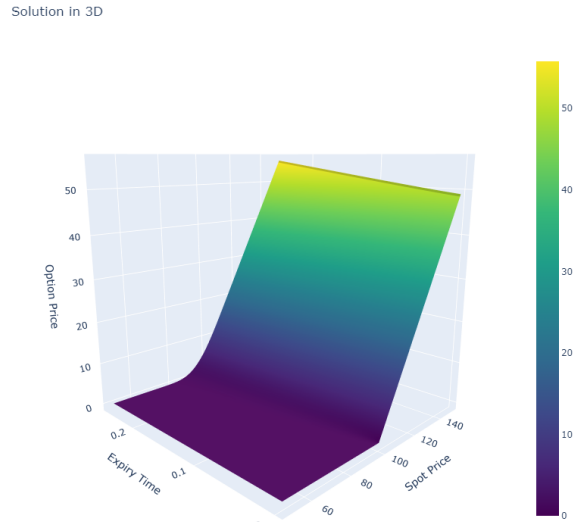


FIGURE 7 – Solution Black Scholes

En comparant avec ce qui est trouvable sur internet, j'ai l'impression d'avoir la bonne représentation.

Une chose qui confirme mon intuition, est le fait qu'avec nos conditions de bords, nous pouvons comparer nos résultats avec la diapositive 211 du cours. C'est dans ce cadre que je vois que nous avons bien ce qu'il faut dans notre vue de côté avec les conditions de Dirichlet. On remarque

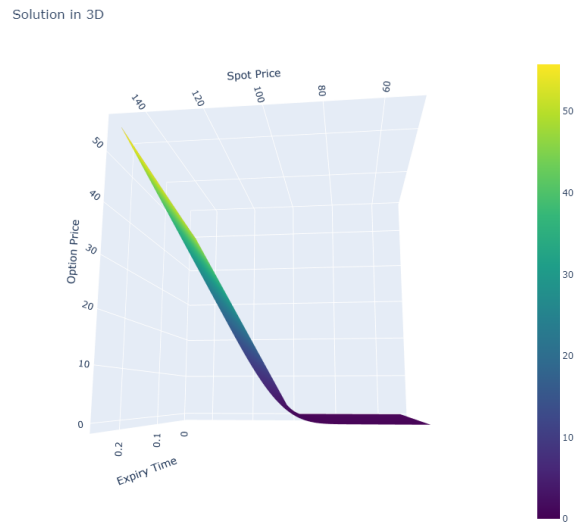


FIGURE 8 – Vue de côté

que nous avons bien une courbe non linéaire au départ, puis qui devient linéaire et qui reste tout droit.