Pricing d'une option européenne avec le modèle de Black-Scholes

Anthony Bataille, Khaoula Belahsen, Ingrid Ngouela Chabifor
Encadrant : Eric Lunéville

3 novembre 2019

Table des matières

1 Introduction du problème

Dans le contexte des marchés financiers, il est courant de manipuler des produits financiers appelés options. Une option est un contrat d'assurance d'une durée prédéterminée T (maturité), ouvrant le droit à acheter (ou vendre) à un certain prix K (appelé strike), à une date future, une certaine quantité d'un actif a (actions, matières premières,...) appelé sous-jacent. Le cours de l'actif désigne sa valeur x(t) sur le marché. L'émetteur de l'option s'est engagé irrévocablement à vendre (ou acheter) le sous-jacent au détenteur de l'option, si celui-ci désire exercer son droit.Dans le cas d'un droit de vente, l'option est appélée put. Et une option d'achat porte le nom de call. Suivant les droits de vente du titulaire du contrat, on distingue plusieurs types d'options. La plus simple, l'option européenne, stipule que le droit de vente ne peut s'exercer qu'à la fin du contrat(i.e. à maturité). Il existe évidemment d'autres types d'options telle que les options américaines qui peuvent être exercées à tout instant 0< t < T. Dans le cadre de ce projet, nous nous restreignons aux options européennes. On définit alors la "récompense" (payoff) $R(t) = (x(t) - K)^+$ qui représente le gain au moment de la vente de l'option. On constate qu'elle est positive si le cours du marché x(t) est supérieur au prix garanti K : le détenteur de l'actif a donc intérêt à vendre sans faire jouer son option. Dans le cas contraire, le détenteur du contrat fera jouer son option et vendra donc au prix K.

La problématique consiste donc à déterminer, dans un contexte fluctuant, le prix p(x, t) d'un tel contrat à tout instant t en fonction du cours x de l'actif sous-jacent à l'option. Pour le faire, nous allons utiliser le modèle de Black-Scholes à risque neutre où la fluctuation aléatoire du prix de l'actif p(x, t) suit un processus stochastique particulier (appelé mouvement brownien), on montre que le prix p(x, t) est donné par l'équation de diffusion rétrograde, $\forall x > 0, \forall 0 < t < T$:

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

où σ^2 est la variance de l'actif (volatilité) et r le taux d'intérêt sans risque. A l'échéance T du contrat, le prix est : p(x, T) = $(xK)^+$. Ainsi, nous connaissons le prix de l'option à l'échéance T : c'est pourquoi la condition initiale de notre problème sera donnée à l'instant T : d'où le qualificatif "rétrograde".

Les actions, obligations et autres produits financiers cotés en bourse ont des prix fluctuants établis en fonction de l'offre et de la demande. En finance, les produits financiers les plus courants sont appelés options.
Une option est une assurance sur le prix d'achat (K strike) d'un actif (action, matière première,...). Ce produit financier donne le droit d'acheter cet actif à un instant T fixé (maturité) à un prix K convenu à l'achat de l'option. Arrivé à échéance, le détenteur de l'option détient le droit de l'utiliser ou non, selon les prix du marché.

Par exemple, si Air France souhaite se prémunir de la hausse du prix de kérosène dans 3 mois, elle peut acheter une option qui dit "Dans 3 mois, Air France peut acheter 1L de kérosène à 1,2 euros/L". Si 3 mois après, le prix du kérosène sur le marché est de 0.9euros/L, Air France n'a pas intérêt à exercer son option, mais si ce prix est de 1.5euros/L, elle exercera son option et pourra économiser des milliers d'euros. On voit donc que dans le cas d'un call l'acheteur est toujours protégé car il choisit l'exercice de l'option donc il ne craint rien. En revanche pour le vendeur le risque est maximal car sa perte est possiblement infinie. Il est donc naturel qu'en mathématiques financières, le problème de pricing d' une option, c est-à-dire de détermination de la valeur de l option à toute date, soit très important.

Tout l'enjeu réside maintenant dans la détermination du prix de cette option.
Nous avons choisi la méthode d'évaluation risque-neutre, ce qui nous place dans un espace probabilisé

ou les prix des sous-jacents sont des martingales et sont donc d'espérance constante. Ainsi on peut considérer que le prix d'un call correspond son espérance de rendement futur actualisé.

2 Description du modèle

2.1 Modèle de Black-Scholes à risque neutre

Afin de déterminer le prix de notre option européenne, on se place dans le cadre au modèle de Black-Scholes. Ce dernier stipule que l'évolution du prix de l'action S_t est donnée par l'équation :

$$dS_t = \mu S_t dt + \sigma S_t dW_t$$
$$\text{avec } \mu, \sigma > 0$$

Le prix de l'actif dépend d'un mouvement brownien W_t qui régie la partie fluctuation aléatoire de ce prix.

2.2 Equation du prix de l'option européenne

Maintenant, si on se place dans le cadre d'une option call, le prix de cette dernière est une fonction p(x,t) qui dépend du temps t et du prix de l'action x (qui était appelé S_t plus haut).

En effectuant une dérivation, on peut montrer que ce prix p(x,t) est donné par une équation de diffusion rétrograde qui s'écrit :

$$\forall x > 0, \forall 0 < t < T :$$
$$\frac{\partial p}{\partial t}(x,t) + \frac{1}{2}\sigma^2 x^2 \frac{\partial^2 p}{\partial x^2}(x,t) + rx\frac{\partial p}{\partial x}(x,t) - rp(x,t) = 0$$

où σ^2 est la variance de l'actif (volatilité) et r le taux d'intérêt sans risque.

À l'échéance T du contrat, le prix est : p(x,T) = $(x-K)^+$.

Ainsi, nous connaissons le prix de l'option à l'échéance T. C'est pourquoi la condition initiale de notre problème sera donnée à l'instant T, d'où le qualificatif "rétrograde".

2.3 Modèle à deux actifs

Dans notre étude, nous nous restreignons à deux actifs notés a_1 et a_2. On considère donc le problème $(x = (x_1, x_2))$

$$\begin{cases} \frac{\partial p}{\partial t}(x,t) + \frac{1}{2}\sum_{i,j=1}^{2} \Xi_{ij} x_i x_j \frac{\partial^2 p}{\partial x_i x_j}(x,t) + r\sum_{i=1}^{2} x_i \frac{\partial p}{\partial x_i}(x,t) - rp(x,t) = 0 \ x > 0, \ 0 < t < T \\ p(x,T) = (x_1 + x_2 - K)^+ \ x > 0 \end{cases}$$

où Ξ est la matrice de covariance associée aux actifs $(a_1, a_2$ que l'on suppose, pour simplifier, indépendante de x et t. En effectuant le changement $t \dashrightarrow -t$ (renversement du temps), on aboutit à une équation de la chaleur à coefficients variables :

$$\begin{cases} \frac{\partial p}{\partial t}(x,t) - \frac{1}{2}\sum_{i,j=1}^{2} \Xi_{ij} x_i x_j \frac{\partial^2 p}{\partial x_i x_j}(x,t) - r\sum_{i=1}^{2} x_i \frac{\partial p}{\partial x_i}(x,t) + rp(x,t) = 0 \ x > 0, \ 0 < t < T \\ p(x,0) = (x_1 + x_2 - K)^+ \ x > 0 \end{cases}$$

Cette équation dégénère en $x_1 = 0$ ou $x_2 = 0$ car alors, des termes d'ordre 2 disparaissent. L'étude théorique de ce problème nous conduit à la résolution numérique du problème suivant :

$$\begin{cases} \frac{\partial p}{\partial t}(x,t) - \text{div}(A(x)(x,t) + V(x).(x,t) + rp(x,t) = 0 \ x > 0, \ 0 < t < T \\ p(x,0) = (x_1 + x_2 - K)^+ \ x > 0 \\ \frac{\partial p}{\partial n}(x,t) = 0 \text{ en } x_1 = \text{a ou } x_2 = \text{a} \end{cases}$$

dans lequel la dernière équation représente la condition aux limites sur la frontière artificielle (que nous avons rajouté pour résoudre le problème de la dégénérescence), et

$$A(x) = \frac{1}{2}\begin{bmatrix} \Xi_{11}x_1^2 & \Xi_{12}x_1x_2 \\ \Xi_{12}x_1x_2 & \Xi_{22}x_2^2 \end{bmatrix} \text{ et } V(x) = \begin{pmatrix} (\Xi_{11} + \frac{1}{2}\Xi_{21} - r)x_1 \\ (\Xi_{22} + \frac{1}{2}\Xi_{12} - r)x_2 \end{pmatrix}$$

Nous choisissons d'effectuer une discrétisation par éléments finis. et pour cela, nous utiliserons, dans un premier temps, le schéma d'Euler implicite ($\theta = 1$) avec un pas de temps constant (qui est d'ordre 1), et en second lieu, le schéma de Crank-Nicolson d'ordre 2.

3 Implémentation en C++

3.1 Matrice et Vecteur

3.1.1 Classe vecteur

On développe une classe vecteur héritée de la classe **vector** de la **stl** qui contient les différents constructeurs ainsi que toutes les opérations algébriques nécessaires.

Plus tard, dans le développement de la classe matrice qui utilise la classe vecteur, il a fallu construire deux fonctions : **put** et **remove** qui prennent en paramètre un index et une valeur réelle et qui insère (resp supprime) la valeur réelle du vecteur.

3.1.2 Classe matrice

3.1.2.1 Stockage profil :

Dans le cas de notre projet, nous travaillons avec des matrices de grande taille mais qui sont très creuses. Une implémentation d'une classe matrice qui stocke tous les zéros de ces matrices est donc inadaptée car elle entraîne des calculs lents pour un stockage inutile.
C'est pour cette raison qu'on privilégie l'implémentation d'une classe **matrice** avec un stockage profil.

Dans ce type de stockage, on ne stocke que les éléments à partir du premier élément non nul de chaque ligne et colonne. Donc on crée une classe matrice de type profil avec les attributs suivants :

— **valL** (vecteur *) : vecteur contenant les valeurs stockées de la partie triangulaire inférieure incluant la diagonale.

— **valU** (vecteur *) : contenant les valeurs stockées de la partie triangulaire supérieure.

— **pl** (vecteur *) : contenant les indices du premier élement non nul de chaque ligne.

— **pu** (vecteur *) : contenant les indices du premier élément non nul de chaque colonne.

— **q** (vecteur*) : contenant les indices des éléments de la diagonale dans valL.

— **dim_m** (int) : dimension de la matrice qui est considérée carrée ici vu que toutes les matrices auxquelles on a à faire sont carrées.

Il est clair que dans le cas des matrices symétriques et non symétriques à profil symétrique, on a que pl = pu.

Constructeurs

On crée 3 constructeurs différents :

— Constructeur par défaut
— Constructeur par valeur (double) : il est nécessaire de faire une disjonction de cas ici selon si la valeur est nulle (on ne stocke de la diagonale dans valL) ou non (on remplit valL et valU)
— Constructeur par copie

Accesseurs

Pour pouvoir récupérer les valeurs des attributs qui sont **protected** dans la classe, on définit des fonctions d'accès **dim()**, **L_()**, **U_()**, **pu()**, **pl()**, **q()**.

3.1.2.2 Fonction modifier :

L'idée derrière cette fonction est de pouvoir rassembler tous les différents cas qui à l'élément (i,j) d'une matrice M assigne une valeur x.
Ainsi, si on a cette fonction, elle peut être utilisée pour modifier l'élément (i,j) d'une matrice et donc calculer ultérieurement les différentes opérations, ainsi que la décomposition LU...

En effet, on ne voyait pas comment modifier directement la valeur de l'élément (i,j) à partir d'une fonction val(i,j) par exemple retrournant une référence sur réel **(double matrice : :val(int i, int j))** puisque le contenu de **valU**, **valL** mais surtout **pu**, **pl** et **q** est ammené à changer selon la valeur du réel qu'on met à l'emplacement (i,j).

La fonction modifier est complexe dans le sens où il existe plusieurs cas, elle a la structure suivante : **void matrice : :modifier(int i, int j, double a)** . ELle agit donc sur la matrice en changeant la valeur de l'élément (i,j). C'est en écrivant cette fonction qu'on a eu l'idée de créer une distinction entre **pu** et **pl**. En effet, au tout début, la classe matrice était définie pour les matrices à profil symétrique et donc il n'y avait qu'un seul p qui devait être changé dans certains cas où M(i,j) était changé mais qui donc devenait faux pour définir les indices dans la partie triangulaire supérieure.

Algorithm 1 Modifier(ligne i,colonne j, valeur x)

```
if M(i,j) non stocké then
  if x=0 then
    rien ne se passe et on s'arrête
  else
    M(i,j) devient le 1er élément non nul sur la ligne
    Les éléments qui suivent vont tous être stockés
    Mise à jour de pl et q
  end if
end if
if M(i,j) déjà stocké then
  if M(i,j) est le premier élément stocké sur la ligne then
    if x=0 then
      on doit supprimer M(i,j)
      on doit supprimer tous les éléments nuls qui le suivent
      mise à jour de p et q
    else
    end if
  else
    M(i,j) n'est pas le premier élément stocké de la ligne On modifie sa valeur p et q ne changent
    pas
  end if
end if
```

Cette démarche est valable quand on est dans la partie triangulaire supérieure et quand on est dans la partie triangulaire inférieure.

3.1.3 Décomposition LU et résolution du système linéaire Ax=b

En utilisant la fonction modifier développée précédemment, on développ eun algorithme de décomposition LU qui contient une partie de résolution du système linéaire en première partie en descente, puis en remontée.

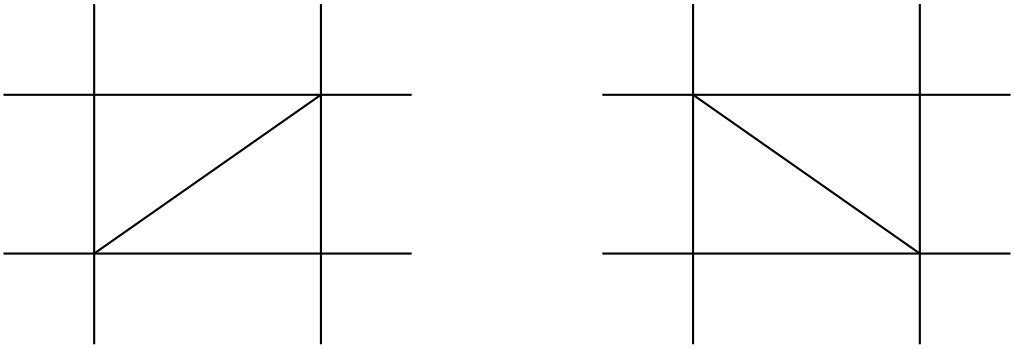
Étant donné que nous avons choisi de résoudre notre équation en utilisant des éléments finis, il nous faut donc construire un maillage qui nous permettra d'approcher la solution sur notre domaine.

3.2 Classe Triangle

On développe d'abord une classe textbfTriangle dont le but est de représenter les éléments de base de nos maillages que sont les triangles. Elle propose :

- un constructeur prenant 3 entiers, permettant intuitivement de représenter un triangle par les numéros globaux de ses sommets ;
- une fonction d'accès (obtenue en surchargeant l'opérateur ()) et renvoyant le numéro global associé au numéro local i d'un triangle ;
- une fonction d'impression (surcharge de l'opérateur «) permettant d'afficher les numéros globaux d'un triangle.

Pour rendre le maillage symétrique nous avons décidé d'alterner le sens de division d'un rectangle à l'autre. Ainsi, nous utlisons les deux divisions suivantes :



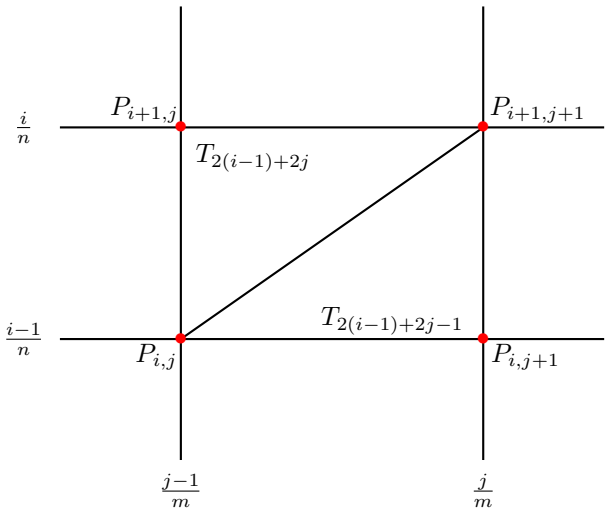
3.3 Classe Maillage

3.3.1 Fonctions maille_carre_unite et constructeur Maillage

Ensuite, on développe ainsi une classe **Maillage**, constituée d'un le générateur de maillage du carré unité nommé **maille_carre_uniteP1** (lorsque le maillage est réalisér avec des éléments P1) ou **maille_carre_uniteP2** (lorsque ce maillage est réalisé avec des éléments P2). Ce mailleur est réalisé à l'aide des classes conteneurs de la STL(vector et list). Il s'agit de générer un découpage régulier du carré unité en triangles non dégénérés. L'union des triangles est le carré et les triangles ne s'intersectent que par une arête. Pour ce faire, en découpe l'axe des abscisses en m>0 segments et celui des ordonnées en n>0 segments.On parcourt l'axe x avec j et l'axe y avec i. La classe maillage contient également un constructeur nommé **Maillage** et servant à appeler un des deux générateurs de maillage en fonction du type d'éléments précisé par l'objet **elmt_type**.

3.3.2 Types de maillage

a) **Maillage avec éléments** P_1
Sur chaque triangle on place 3 noeuds, repérés en sur les sommets du triangle. On a donc $1 \leq j \leq m+1$ et $1 \leq i \leq .n+1$



Après calculs, on trouve que le numéro global du sommet $P_{i,j}$ est $n^\circ P_{i,j}$=j+(m+1)(i+1), pour tout $1 \leq j \leq m+1$ et $1 \leq i \leq n+1$. On peut donc retrouver les numéros globaux de tous les sommets d'un rectangle, à partir du numéro global d'un des sommets du rectangle. Nous dee le faire en partant à chaque fois, à partir du sommet du rectangle se trouvant en bas, à gauche du rectangle. On obtient les numéros globaux suivants :

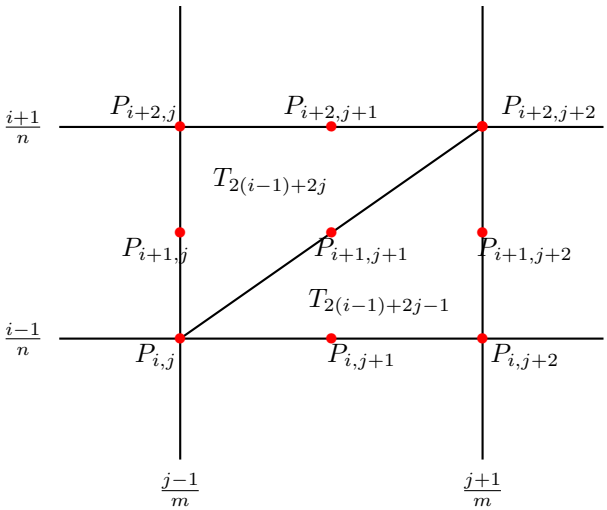
- $n^\circ P_{i,j+1}$=$n^\circ P_{i,j} + 1$
- $n^\circ P_{i+1,j}$=$n^\circ P_{i,j} + m + 1$
- $n^\circ P_{i+1,j+1}$=$n^\circ P_{i,j} + m + 2$

$T_{2(i-1)+2j}$ désigne le triangle de numéro global $2(i-1)+2j$ dans le maillage.

C'est en se basant sur les considérations ci-dessus, et en parcourant chaque triangle dans les sens direct à partir de l'angle droit, que nous avons écrit notre programme.

b) **Maillage avec éléments P_2**

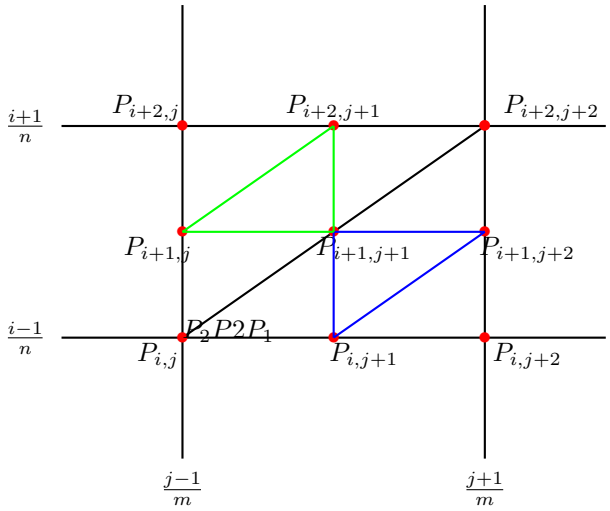
Sur chaque triangle on place 6 noeuds : répérés sur les sommets et les milieux des cotés du triangle. Ainsi, on a $1 \leq j \leq 2m+1$ et $1 \leq i \leq .2n+1$. On obtient le schéma suivant :



De la même que dans le cas des éléments P_1, les numéros globaux de tous les noeuds du rectangle contenu entre les abscisses $\frac{j-1}{n}, \frac{j+1}{n}$ et les ordonnées $\frac{i-1}{n}, \frac{i+1}{n}$, sont obtenus à partir du numéro global du sommet $P_{i,j}$, qui dans ce cas est $n^\circ P_{i,j} = (j-1)2m + j + (i-1)$. En effet, on a : -

$n°P_{i,j+1} = n°P_{i,j} + 1$
- $n°P_{i,j+2} = n°P_{i,j} + 2$
- $n°P_{i+1,j} = n°P_{i,j} + 2m + 1$
- $n°P_{i+1,j+1} = n°P_{i,j} + 2m + 2$
- $n°P_{i+1,j+2} = n°P_{i,j} + 2m + 3$
- $n°P_{i+2,j} = n°P_{i,j} + 2(2m + 1)$
- $n°P_{i+2,j+1} = n°P_{i,j} + 2(2m + 1) + 1$
- $n°P_{i+2,j+2} = n°P_{i,j} + 2(2m + 1) + 2$

3.3.2.1 Remarque Pour que le tracé de notre maillage P_2 fasse apparaître les 6 noeuds de chaque triangle, nous avons subdivisé chaque triangles P_2 en 4 triangles P_1 virtuels de la manière suivante (pour pouvoir les stocker dans le fichier texte des données plus faciles à utiliser) :



3.3.3 Objets liste_som et liste_tri

Pour stocker les données de nos maillages, nous définissons les objets suivants :

— **liste_som** qui est un **vector** et qui contient à l'indice i, le vecteur des 2 coordonnées du sommet de numéro global i ;

— **liste_tri** qui est un **list**, contient à l'indice i, le vecteur des numéros globaux des 3 sommets du triangle de numéro global i ;

3.3.4 Fonction Print et surcharge de l'opérateur «

Nous avons également prévu une fonction **Print** permettant d'imprimer les données de nos objets **liste_som** et **liste_tri** à l'écran. Elle est utilisée par l'opérateur surchargé «, pour afficher ces données à l'écran, directement à partir d'un objet de la classe **Maillage**.

3.3.5 Fonction save_to_file

Cette fonction permet d'écrire le contenu des objets **liste_som** et **liste_tri** dans un fichier texte, afin de pouvoir les utiliser avec matlab pour tracer et visualiser le maillage.

3.3.6 Fonctions tf_affine, maille_rectangle

tf_affine est une fonction membre de la classe maillage permettant d'effectuer une transformation affine d'un maillage à partir d'une matrice A$\in \mathcal{M}_2$ (qui représente une homothétie et/ou une rotation) et le vecteur de translation t $\in \mathbb{R}^2$.

maille_rectangle est une fonction membre permettant de mailler un rectangle dont les sommets sont passés en arguments d'entrée. Elle appelle à la fonction **tf_affine**.

3.4 Matrices élémentaires, assemblage et schéma itératif

Une fois les classe Matrice et Maillage créées, on peut calculer les itérés du schéma numérique et enregistrer les valeurs obtenues dans un fichier.

Afin de rendre le code modulable, on crée également une classe **Quadrature** qui définit la taille (**int**), les poids (**double ***) et les points (**Vecteur ***) de quadrature sur le triangle de référence. On implémente deux fonctions génératrice de quadrature qui génèrent respectivement une quadrature de Gauss-Legendre à 3 points et une formule de quadrature de Gauss-Lobatto à 7 points et retourne un pointeur vers l'objet quadrature définit dynamiquement. La première formule est exacte pour les calculs en éléments finis P^1 et la seconde est exacte pour les calculs en éléments finis P^2.

Afin de pouvoir interchanger facilement entre le calcul en éléments finis $P1$ et $P2$ on définit des fonctions de la forme **tau*x*P*j*** (fonctions de bases du triangle de référence) et **Dtau*x*P*j*** (gradients des fonctions de bases du triangle de référence) qui seront appelées par les fonction de construction des matrices élémentaires. Ainsi, x décrit les numéros des noeuds et $j = 1, 2$.

Le processus de calcul se résume de la manière suivante aussi bien dans les cas P^1 que P^2 :

1. Définition d'un maillage avec un constructeur de la classe **Maillage**.
2. Enregistrement des données du maillage dans un fichier.
3. Définition des données Ξ, r, les Matrice $\mathbb{M}$, $\mathbb{K}$, $\mathbb{B}$ vides et une quadrature Q.
4. Construction des matrices globales (fonction appelant des fonctions de construction des matrices élémentaires).
5. Calcul des itérés du schéma numérique et enregistrement des résultats dans un fichier.

3.4.1 Matrices élémentaires et assemblage

3.4.1.1 La fonction build. Cette fonction prend en argument par référence les matrice globales vides.
Elle agit sur chaque triangle du maillage en appelant la fonction **matrice_elem** qui rempli les matrices élémentaires Mel, Kel et Bel. Elle rajoute aux bonnes coordonnées des matrice globales les valeurs des matrices élémentaires.

3.4.1.2 La fonction matrice_elem. Cette fonction est appelée autant de fois qu'il y a de triangles dans le maillage. Elle prend en argument par référence les matrices élémentaires vides et les données du triangle courant et les remplies. Elle calcul d'abord la matrice **B**=B_l qui intervient dans la transformation affine du triangle de référence vers le triangle courant. Elle calcul aussi les valeurs **B_inv_t**=$(B_l^{-1})^\intercal$ et **ADet_B**$|\det(B)|$ qui seront utilisées dans le calcul d'intégrales par changement de variable.

Ensuite, elle appelle les fonctions **matM_elem, matK_elem** et **matB_elem** qui retournent les matrice élémentaires.

3.4.1.3 Les fonction matM_elem, matK_elem et matB_elem. Elles prennent en argument les différentes données du problème ainsi que **B** et **B_inv_t** calculés précédemment. Elles calculent les matrice élémentaires en utilisant la formule de quadrature **Q** définie plus tôt via une boucle sur les indices de la matrice et les points de quadrature. Le même code peut servir à calculer les matrices P^1 ou P^2 et avec n'importe quelle formule de quadrature car ces éléments sont définis au préalable. Le calcul des intégrales se fait avec changement de variable.

$$\begin{aligned} M_{ij}^l &= \int_{\hat{T}} \tau_i \tau_j |\det(B_l)| \, \mathrm{d}\Omega \\ K_{ij}^l &= \int_{\hat{T}} \left(A(\mathcal{F}_l)(B_l^{-1})^\intercal \nabla\tau_i\right) \cdot \left((B_l^{-1})^\intercal \nabla\tau_j\right) |\det(B_l)| \, \mathrm{d}\Omega \\ B_{ij}^l &= \int_{\hat{T}} \left(V(\mathcal{F}_l(V) \cdot (B_l^{-1})^\intercal \nabla\tau_j\right) \tau_i |\det(B_l)| \, \mathrm{d}\Omega \end{aligned}$$

Enfin, on ne calcul que les parties inférieures des matrices M^l et K^l car elles sont symétriques, cela accélère l'exécution du programme.

3.4.2 Schéma itératif

Une fois les matrices $\mathbb{M}$, $\mathbb{D}$ construites on peut calculer les vecteurs P^k. On définit un nombre d'itérations $k_{max:} = \frac{T}{\Delta_t}$. Une fonction **schema_iteratif** est dédiée au calcul des P^k.

3.4.2.1 La fonction schema_iteratif. Cette fonction prend en argument les matrices $\mathbb{M}$ et $\mathbb{D}$ ainsi que d'autres données du problème et le chemin du fichier dans lequel seront écrits les itérés. Elle appelle la fonction **LU_factor** qui calcul la décomposition LU de la matrice nécessaire au schéma (par exemple, $\mathbb{M} + \Delta_t \mathbb{D}$ pour le schéma d'Euler implicite). Puis, dans une boucle sur k allant jusqu'au nombre d'itérés, les valeurs de P^k sont calculées avec à l'aide de la fonction **LU_solve** qui résout un système linéaire de la forme $LUX = Y$ par descente et remontée. A chaque itération, le vecteur P^k est enregistré dans le fichier.

3.4.3 Validation du code de matrices élémentaires et assemblage

Afin de valider le code calculant les matrices élémentaires, on les applique au triangle de référence.

3.4.3.1 Matrices élémentaires. On sait que les fonctions de bases locales d'un triangle correspondent aux cordonnées barycentriques de ce triangle. Pour le triangle de référence, elles sont données

par

$$\begin{aligned}\tau_1(x,y) &= 1-x-y\\ \tau_2(x,y) &= x\\ \tau_3(x,y) &= y\end{aligned}$$

Donc la matrice de masse élémentaire du triangle l s'écrit

$$M^l = \begin{pmatrix} \frac{1}{12} & \frac{1}{24} & \frac{1}{24} \\ \frac{1}{24} & \frac{1}{12} & \frac{1}{24} \\ \frac{1}{24} & \frac{1}{24} & \frac{1}{12} \end{pmatrix}$$

Toujours pour le triangle de référence, on a en maillage $P1$,

$$\begin{aligned}\nabla\tau_1(x,y) &= \begin{pmatrix}-1\\-1\end{pmatrix}\\ \nabla\tau_2(x,y) &= \begin{pmatrix}1\\0\end{pmatrix}\\ \nabla\tau_3(x,y) &= \begin{pmatrix}0\\1\end{pmatrix}\end{aligned}$$

Cela donne, avec $A(x) = \mathrm{Id}$,

$$K^l = \begin{pmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} & 0 \\ -\frac{1}{2} & 0 & \frac{1}{2} \end{pmatrix}$$

Pour valider B^l, on peut remarquer qu'elle n'est pas symétrique.

3.4.3.2 Matrices globales La validation des matrices globales peut se faire en utilisant leurs propriétés particulières. On note $V = \begin{pmatrix}1\\ \vdots \\ 1\end{pmatrix} \in \mathbb{R}^N$ où N est le nombre de sommets du maillage. Alors,

$$\begin{aligned}\langle \mathbb{M}V, V\rangle &= \mathrm{mes}(\Omega)\\ \|\mathbb{K}V\|^2 &= 0\end{aligned}$$

où $\mathrm{mes}(\Omega)$ est l'aire du domaine.

Dans une fonction **run_FE**, on effectue tous ces tests. On peut voir que l'on a également la propriété $\|\mathbb{B}V\|^2 \simeq 0$ mais nous ne n'avons de preuve théorique de ce résultat.

4 Visualisation et interprétation des résultats en Matlab

Une fois le maillage et les vecteurs P^k enregistrés dans un fichier, on utilise les fonctions de visualisations de Matlab.

La routine **affichemaillage** lis les données du maillage et l'affiche dans une figure.

La routine **visu** représente les solutions itérés de façon dynamique.

On représente ci-dessous l'évolution du prix du contrat calculé avec des éléments finis P^1.

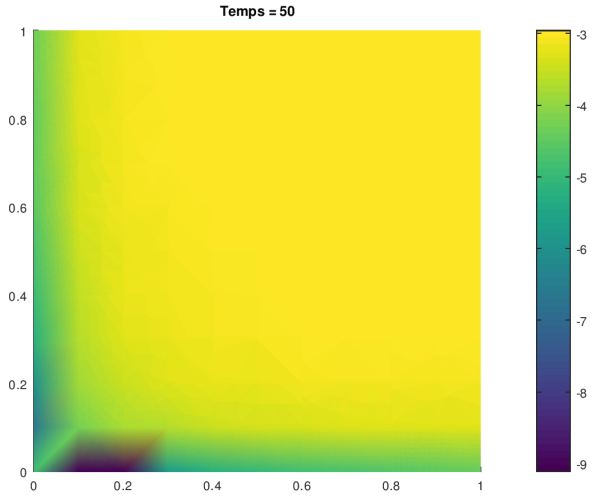


FIGURE 1 – Représentation de $\log(p(x,t))$ à $t = 50$, maillage P^1, Euler implicite

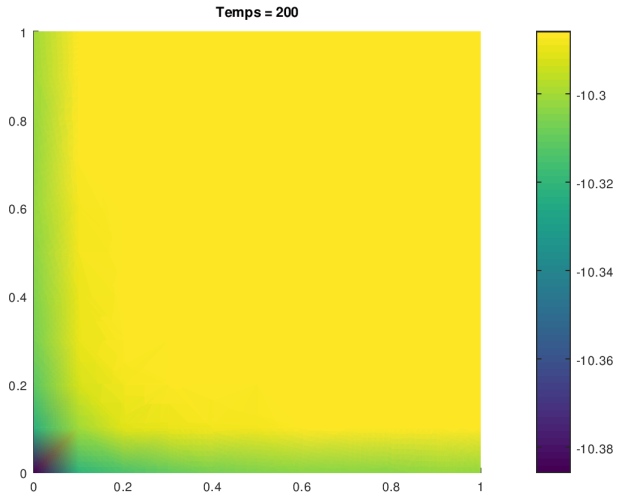


FIGURE 2 – Représentation de $\log(p(x,t))$ à $t = 20$, maillage P^1, Euler implicite

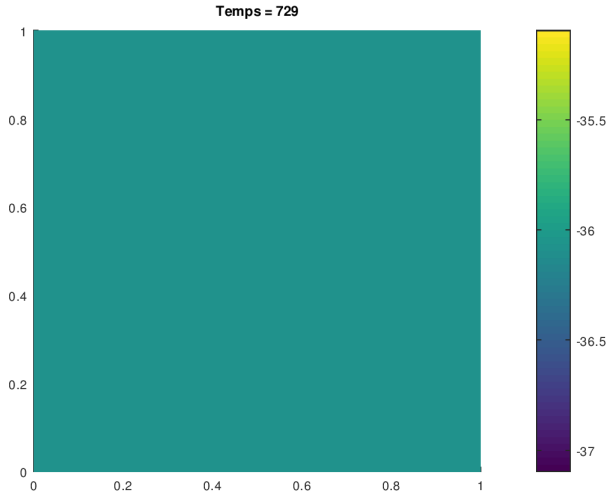


FIGURE 3 – Représentation de $\log(p(x,t))$ à $t = 729$, maillage P^1, Euler implicite

On représente maintenant la même quantité calculée avec des éléments finis P^2.

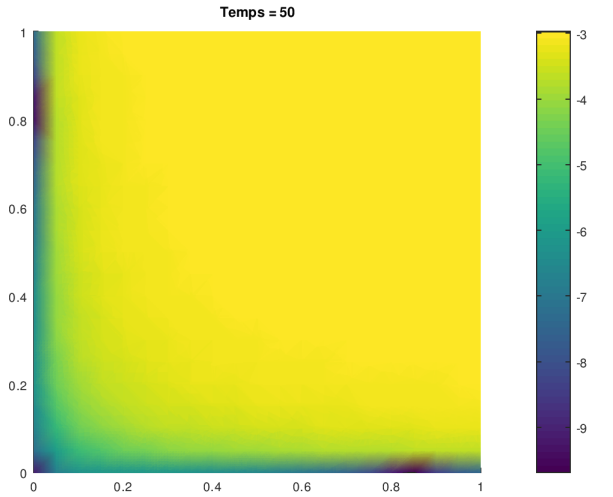


FIGURE 4 – Représentation de $\log(p(x,t))$ à $t = 50$, maillage P^2, Euler implicite

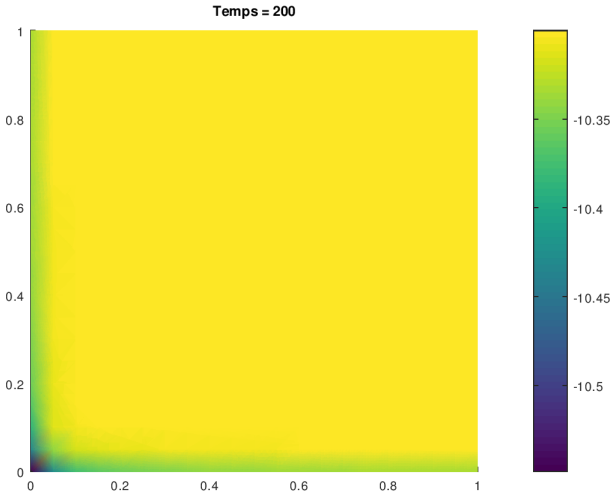


FIGURE 5 – Représentation de $\log(p(x,t))$ à $t = 20$, maillage P^2, Euler implicite

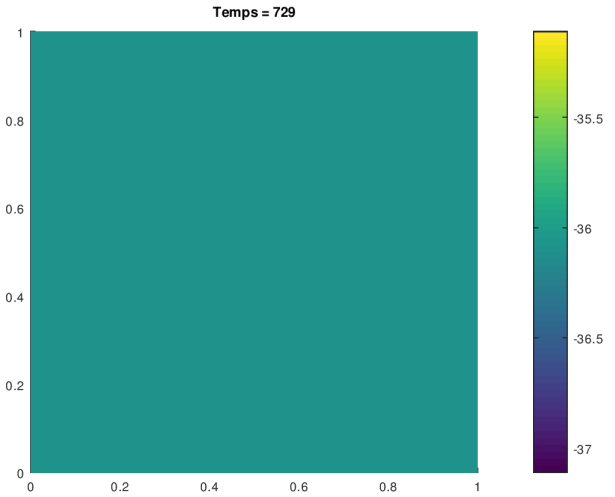


FIGURE 6 – Représentation de $\log(p(x,t))$ à $t = 729$, maillage P^2, Euler implicite

On voit que les deux simulations donnent des résultats similaires : La solution tend globalement vers 0.

Un essai avec le schéma de Cranck-Nicolson d'ordre 2 donne des résultats similaires.

5 Conclusion

Les simulations montrent que la solution converge fortement vers 0 dans le cas de $K = 1$. Le prix du call a donc tendance à diminuer lorsque l'échéance approche (i.e. la maturité baisse) à cause de l'effet temps et tend vers le payoff à maturité. Ainsi, un call d'échéance 7 mois est plus cher qu'un call d'échéance 6 mois. Donc nos résultats sont cohérents avec la théorie.