Master 1 Mathématiques et Applications
U.E. ANAC++, année 2025–2026

ANAC++ TP 4 : Poisson, chaleur et Black-Scholes

Le travail sera à faire en binôme. Il sera considéré pour la note Projet

---

Les programmes en C++, à rendre sur la plateforme Moodle, doivent être accompagnés d'un compte-rendu répondant aux questions théoriques et faisant une analyse bien soignée des résultats.

Exercice 1 Résolution du problème de Poisson 1D

Soit $\Omega =]0,1[$ et on cherche une fonction $u : \Omega \to \mathbb{R}$ telle que

$$\begin{cases} -u''(x) + \alpha u(x) = f(x), & \forall x \in \Omega, \\ u(0) = u(1), \end{cases} \tag{1}$$

avec $\alpha \geq 0$ et la fonction f données.
Afin de discrétiser le problème par la méthode des différences finies, on introduit un maillage formé de $N+1$ points équidistants, dont $N-1$ à l'intérieur de l'intervalle $]0,1[$:

$$0 = x_0 < x_1 < \cdots < x_{n-1} < x_N = 1 \text{ avec } h = \frac{1}{N} \text{ et } x_i = ih \, (i = 0, \ldots, N).$$

On note u_i une approximation de $u(x_i)$, la solution de (1) au point x_i. On obtient le système linéaire :

$$\begin{cases} \dfrac{-u_{i-1} + 2u_i - u_{i+1}}{h^2} + \alpha u_i = f_i & \forall 1 \leq i \leq N-1, \\ u_0 = u_N. \end{cases}$$

1. Ecrire la fonction `matrice_1D_profil`(N, c_0, c_1) qui donne en sortie la matrice A du problème de Poisson en stockage profil, avec N sa dimension, c_0 le coefficient sur la sous-diagonale et c_1 celui sur la diagonale. Faire attention à la forme de la matrice liée aux conditions aux limites périodiques.
2. Les fonctions `factorisation_LDLt(A)` et `resolution_LDLt(A,b)` développées au TP3 permettent de résoudre le système d'équations $Ax = b$. Pour valider toutes ces fonctions, on introduit d'abord la fonction exacte $\texttt{uex}(x) = sin(2\pi x)$ et on calcule (à la main) second membre de l'équation $f(x) = (-\texttt{uex}'' + \alpha \texttt{uex})(x)$.
3. On écrit la fonction `verif_poisson(N,file)` qui résout le problème de Poisson avec cette fonction f. On écrit N et l'erreur $\varepsilon_N = \max_j |u_j - \texttt{uex}(x_j)|$ à l'écran et dans un fichier "file".

4. Le programme principal exécutera `verif_poisson` pour un α de votre choix, en prenant $N = \{4, 8, 16, 32, 64, 128, 256, ...\}$. Sauver les résultats dans un fichier, où dans chaque ligne on écrira N, ε_N. Que se passe-t-il si on prend $\alpha = 0$?
5. Avec python, tracer le nuage $(\log(N), \log(\varepsilon_N))$ et calculer la droite de régression pour déterminer expérimentalement le taux de convergence β dans $\varepsilon_N \approx C\, h_N^\beta$. Commenter les résultats obtenus.

<u>Exercice 2</u> Deux schémas numériques pour l'équation de la chaleur

Soit $\Omega =]0,1[$ et $u_0 \in L^2(\Omega) \cap L^\infty(\Omega)$. On s'intéresse à la résolution numérique par différences finies de l'équation de la chaleur donnée par :

$$\left\{\begin{array}{rll} \dfrac{\partial u(t,x)}{\partial t} - \nu(x)\dfrac{\partial^2 u(t,x)}{\partial x^2} &= f(t,x), & \forall (t,x) \in]0,T]\times]0,1[, \\ u(0,x) &= u_0(x), & \forall x \in]0,1[, \\ u(t,0) &= g(t,0), & \forall t \in \; [0,T], \\ u(t,1) &= g(t,1), & \forall t \in \; [0,T], \end{array}\right. \tag{2}$$

avec $\nu(x) > 0$ une fonction qui donne la conductivité thermique de la barre Ω. Ici on prendra

$$\nu(x) = 1 \text{ si } x \in]0, 0.5[\;\text{ et }\; \nu(x) = 10 \text{ si } x \in [0.5, 1[, \tag{3}$$

qui correspond à une barre formée de deux matériaux de conductivité différente.
On va mettre en œuvre les schémas en temps d'Euler explicite $(\mathcal{S}_{expl})$ et Euler implicite $(\mathcal{S}_{impl})$ (donnés en cours) afin de comparer leur précision avec une solution exacte donnée, ainsi que leur rapidité de calcul. Si on note h le pas d'espace et k le pas de temps, on peut montrer que $(\mathcal{S}_{expl})$ est stable sous la condition $k < \dfrac{h^2}{2\|\nu\|_\infty}$, tandis que $(\mathcal{S}_{impl})$ est toujours stable (donc on prendra par exemple $k = h$).

1. On considère la solution exacte : $\texttt{uex}(t,x) = cos(t)cos(\pi x)$ et on en déduit la condition initiale u_0, le second membre $f(t,x) = \texttt{uex}_t(t,x) - \nu(x)\,\texttt{uex}_{xx}(t,x)$ et les conditions au bord $g(t,0) = \texttt{uex}(t,0)$ et $g(t,1) = \texttt{uex}(t,1)$.
2. Donnés la fonction ν, le temps final T, le nombre de mailles N, et le vecteur correspondant à la discrétisation de la solution initiale u_0, écrire respectivement les fonctions nommées `chaleur_explicite` et `chaleur_implicite` qui implémentent les méthodes respectives. L'argument de sortie de ces fonctions sera le vecteur correspondant à l'approximation de la solution au temps $T = Kk$ (noté U^K). On remarque que :
   — la fonction `prod_mat_vect` développée au TP3 permet de calculer une itération en temps du schéma $(\mathcal{S}_{expl})$;
   — les fonctions `factorisation_lu` et `resolution_lu` développées au TP3 permettent de résoudre le système linéaire dans le schéma $(\mathcal{S}_{impl})$.
   Expliquer pourquoi ici on ne peut pas utiliser la factorisation de Crout LDL^T.
3. Ecrire respectivement les fonctions `verif_chaleur_explicite(T,N)` et `verif_chaleur_implicite(T,N)` qui résolvent l'équation de la chaleur avec les données T, N, la fonction ν donnée dans (3) et les fonctions u_0, f et g calculées précédemment à partir de `uex`. Elles écrivent N et l'erreur finale $\varepsilon_N = \max_j |u_j^K - \texttt{uex}(T, x_j)|$ à l'écran et dans un fichier "file".

4. Grâce à un `switch/case`, le programme principal exécutera soit `verif_chaleur_explicite` soit `verif_chaleur_implicite` pour $T = 1.0$, en prenant $N = \{4, 8, 16, 32, 64, 128, ...\}$. Sauver les résultats dans un fichier, où dans chaque ligne on écrira N, ε_N.
5. Avec python, tracer le nuage $(\log(N), \log(\varepsilon_N))$ et calculer la droite de régression pour déterminer expérimentalement le taux de convergence β dans $\varepsilon_N \approx C\, h_N^\beta$. Commenter les résultats obtenus.

<u>Exercice 3</u> Résolution de l'équation de Black-Scholes

On cherche $u \in \mathcal{C}^{1,2}([0,T] \times (0,L))$ solution de l'équation de Black-Scholes réécrite comme une équation de la chaleur avec donnée initiale (en ayant changé la variable t en $T - t$) :

$$\begin{cases} \dfrac{\partial u(t,x)}{\partial t} - \dfrac{1}{2}\sigma^2 x^2 \dfrac{\partial^2 u(t,x)}{\partial x^2} = 0, & \forall (t,x) \in (0,T] \times (0,L), \\ u(0,x) = h(x), & \forall x \in (0,L), \\ u(t,0) = 0, & \forall t \in [0,T], \quad (C.L.\, de\, Dirichlet) \\ \dfrac{\partial u(t,L)}{\partial x} = 0, & \forall t \in [0,T]. \quad (C.L.\, de\, Neumann) \end{cases} \tag{4}$$

Dans le cas d'un *call européen* (option d'achat) on fixera par exemple le *strike* de l'option à : $K(x) = 0.95\, x$ avec $x \in (0,L)$ le cours actuel de l'action (le *strike* varie en fonction du cours actuel de l'action et, pour simplifier, on parie que le prix final sera inférieur à celui actuel, quelque soit la valeur x). On rappelle que, dans ce cas, on a : $h(x) = max(x - K, 0)$.
Si on note h le pas d'espace et k le pas de temps, on va utiliser le schéma implicite $(\mathcal{S}_{impl})$ qui est toujours stable, donc in prendra par exemple $k = h$.

1. Donné le vecteur X de discrétisation de l'intervalle $[0,L]$ avec N sous-intervalles (donc $N+1$ points si on compte les 2 extrémités), définir les fonctions $K(x)$ et $u_0(L,N)$ pour le *payoff* du contrat d'achat.
2. Donné le vecteur X, la volatilité σ et les pas h et k, définir la fonction `matrice_BS_bande` qui assemble la matrice tridiagonale de Black-Scholes en stockage bande. Attention à la condition de Neumann en $x = L$.
3. Donnés la longueur de l'intervalle L, le temps final T, la volatilité σ, le nombre de mailles N, et le vecteur correspondant à la discrétisation de la solution initiale u_0, écrire la fonction `BS_implicite` qui implémente la méthode $(\mathcal{S}_{impl})$. L'argument de sortie de cette fonction sera le vecteur correspondant à l'approximation de la solution au temps T. De plus, à chaque itération en temps n, la solution approchée U^n sera écrite dans un fichier, afin de faire un plot 3D en post-traitement. On remarque que les fonctions `factorisation_lu` et `resolution_lu` développées au TP3 permettent de résoudre le système linéaire.
4. Ecrire le programme principal qui lira les données et exécutera `BS_implicite`. Prendre par exemple : $L = 10000$ (euros), $T = 365$ (jours), $\sigma = 0.2$ et N assez grand pour que la solution calculée soit assez précise. D'autres choix sont aussi possibles. Avec python, tracer l'évolution de la solution au cours du temps (plot 3D), le *payoff* du contrat et le prix de l'option d'achat (solution calculée au temps T). Commenter les résultats obtenus.