Projet 2
Optimisation de la résolution d'un système linéaire par la méthode du gradient conjugué et application à l'équation de la chaleur

Groupe 3 - Equipe 5

Responsable : Abdelkodousse Fettouhi
Secrétaire : Baptiste Lecat
Codeurs : Imad Boudroua, Imane Bouzidi , Lucas Guedon

Résumé : Ce projet consiste à mettre en place des algorithmes de résolution de systèmes linéaires. Dans un premier temps, le projet suggère de comparer l'efficacité entre deux méthodes : celle de la décomposition de Cholesky et la méthode du gradient conjugué. Après avoir testé les implémentations, le but est de résoudre l'équation de la chaleur en se ramenant à un système linéaire matriciel auquel on peut appliquer la méthode du gradient conjugué.

1 Décomposition de Cholesky

Dans cette première partie, nous nous intéressons à la décomposition de Cholesky permettant de résoudre des systèmes linéaires sans calculer l'inverse d'une matrice.

1.1 Question 1

Nous avons réalisé l'algorithme de factorisation de Cholesky. L'algorithme permet de décomposer une matrice symétrique définie positive A en un produit $T *^t T$ avec T matrice triangulaire inférieure.

Pour tester notre algorithme, nous avons comparé A avec la décomposition de Cholesky obtenue. L'utilisation de nombres flottants affecte alors la comparaison des deux matrices. Une précision est nécessaire pour réaliser le test de notre algorithme.

La complexité de notre algorithme est de N^3 pour une matrice de taille N^2. En effet, la définition de la décomposition nous a amené à utiliser trois boucles imbriquées. Cela est dû à la dépendance des deux sommes entre elles.

1.2 Question 2

Nous avons donc un algorithme de complexité N^3. Pour résoudre le système linéaire $Ax = b$, nous utilisons l'algorithme pour obtenir la matrice T. Nous devons donc résoudre le système suivant noté (1) :

$$T.{}^tT.x = b \iff \begin{cases} T.y = b \\ {}^tT.x = y \end{cases} \tag{1}$$

Chaque égalité demande une résolution d'un système linéaire et donc de complexité N^2 en tout. Donc la résolution du système linaire est limité par le calcul des coefficients de la matrice T.

1.3 Question 3

Le but de l'algorithme est de générer des matrices définies positives. Pour cela on s'appuie sur la méthode de Cholseky en construisant des matrices T triangulaires inférieures. Une des particularités est que les matrices générées sont creuses. Notre algorithme produit donc des matrices avec un nombre de termes extra-diagonaux non nuls. Une fois la matrice T obtenue, il suffit de faire $T *^t T$ pour obtenir une matrice symétrique définie positive. Il faut donc calculer tous les coefficients de la matrice, l'algorithme a une complexité en N^2.

1.4 Question 4

L'algorithme de factorisation de Cholesky incomplète est très similaire à la version complète à la différence d'une condition supplémentaire. Celle-ci permet d'éviter les calculs lorsque le coefficient A[i, j] est nul, car les coefficients nuls n'influencent pas le reste de l'algorithme.

Pour tester cet algorithme, on ne peut pas comparer les matrices T obtenues dans les deux algorithmes, car elles peuvent être différentes. Toutefois, on peut, comme pour l'algorithme précédent, comparer $T *^t T$ et A à une précision près.

La complexité de l'algorithme pour une matrice creuse de taille N^2 avec D termes extra-diagonaux non nuls est $N^2 + D * N$. En effet, il y a trois boucles imbriquées, mais la troisième boucle est visitée uniquement pour les D termes extra-diagonaux non nuls et pour les N éléments de la diagonale.

1.5 Question 5

Comme $T.^tT$ est très proche de A, la matrice ${}^tT^{-1}T^{-1}$ est très proche de A^{-1}. T^tT est donc un très bon préconditionneur. Dans nos tests, on obtient un conditionnement pour ${}^tT^{-1}T^{-1}A$ très proche de 1. Ce résultat est prévisible, puisque ${}^tT^{-1}.T^{-1}.A$ est très proche de la matrice identité.

2 Méthode du gradient conjugué

Dans cette deuxième partie, nous nous intéressons à un algorithme permettant de résoudre des systèmes linéaires : **la méthode du gradient conjugué**. La matrice A doit être nécessairement *symétrique définie positive.*

2.1 Question 1

L'implémentation en Matlab donnée ne respecte pas certaines conventions de codage tels que, l'omission du **end** pour clôturer la fonction, ou encore l'utilisation du vecteur **x** donné en argument ; x n'est utile que s'il représente une solution potentielle au système linéaire. Il est tout à fait possible de l'initialiser à l'intérieur de la fonction au vecteur nul.
En outre, nous préférerons, pour un nom de variable composé de plusieurs mots, de différencier les mots par une majuscule, par exemple : **rsOld** à la place de **rsold** et **rsNew** à la place de **rsnew**. Enfin, l'absence des commentaires rend la compréhension de l'algorithme plus difficile.

2.2 Question 2

Dans la décomposition de Cholesky, la recherche de la matrice T a une complexité en $O(N^3)$ pour une matrice de taille N^2. La méthode du gradient conjugué permet une résolution du système $Ax = b$, sans modification de la matrice A. Le gain de complexité occasionné est dû, alors, au non calcul de la matrice T, mais il n'est pas systématique, car le choix de x peut influencer sur le temps de résolution. Dans le cas d'une solution potentielle x entrée en argument, nous épargnons du travail à l'algorithme ce qui implique une diminution du nombre des itérations. Dans le cas de x égal au vecteur nul le nombre des itérations est nécessairement conséquent.

2.3 Question 3

Pour implémenter la méthode du gradient conjugué, nous nous sommes basés sur l'implémentation en `Matlab` donnée dans l'énoncé. Nous avons utilisé la fonction auxiliare `MDPGenerator` qui génère, pour une taille donnée, une matrice A symétrique définie positive. A est le produit d'une matrice triangulaire supérieure et de sa transposée. Les coefficients de la matrice triangulaire sont strictement positifs *(générés aléatoirement entre 5 et 10).*

Le **premier** test a été effectué avec un système où la matrice A est de taille $2 * 2$. Nous avons comparé les coefficients (un à un) de la solution exacte à ceux de la solution trouvée par l'algorithme. Le **deuxième** test consistait à calculer l'erreur absolue, cette erreur désigne la valeur absolue de la différence entre les normes de la solution exacte et de la solution donnée par l'algorithme. Nous avons calculé cette erreur pour la méthode du gradient conjugué *sans* préconditionnement, *avec* préconditionnement et pour `linalg.solve`.
Ce dernier test est réalisé avec `test_conjgrad_linalg` en procédant comme suit :

1. On génère aléatoirement une matrice A et un vecteur solution X_s. On stocke dans la variable b le produit $A * X_s$.
2. On récupère dans la variable $xconjgrad$ la solution du système $Ax = b$ donnée par la méthode du gradient conjuguée, dans la variable $xpreconjgrad$ la solution obtenue avec préconditionnement et dans la variable $xlinalg$ la solution donnée par `linalg.solve`.
3. On calcule les 3 erreurs absolues $|X_s - xconjrad|$ et $|X_s - xlinalg|$ et $|X_s - xpreconjgrad|$. On génère la variation des erreurs pour des matrices A de taille croissante.
4. On modélise les erreurs par 3 courbes représentées sur la *figure 1* :

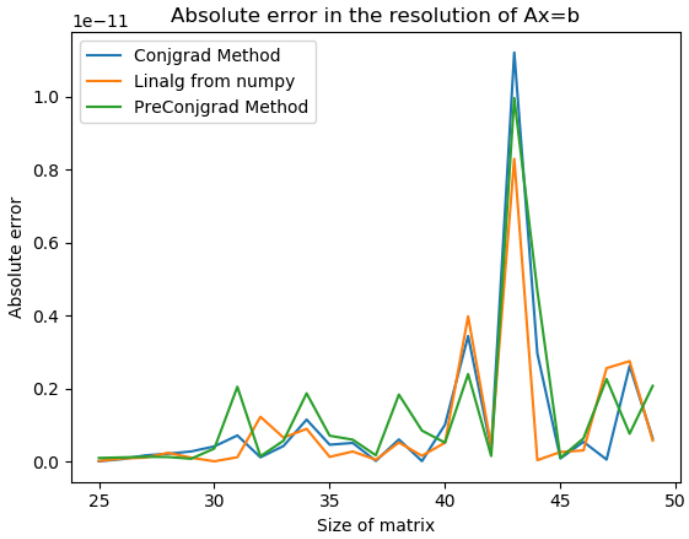

FIGURE 1 – Erreur absolue pour les 3 méthodes

Le graphe montre les variations de l'erreur absolue en fonction de la taille de la matrice A. Cette allure n'est pas unique car les valeurs de la matrice A sont générées aléatoirement.
De la figure 1, nous tirons ce qui suit :

— L'ordre de grandeur de l'erreur est de 10^{-11} ce qui signifie que les solutions données par les 3 algorithmes (`conjgrad, linalg.solve et preconjgrad`) sont, à une constante multiplicative près, égales à la solution attendue.

— A chaque maximum *(local ou global)* atteint, l'erreur de `conjgrad` est supérieure à l'erreur de `preconjgrad` qui est, à son tour, supérieure à celle de `linalg.solve`. Ce résultat est logique car plus l'algorithme est précis, moins l'erreur est significative. On peut dire alors, que `linalg.solve` est plus précis que `preconjgrad` qui est plus précis que `conjgrad`.

— L'erreur absolue **croit** avec la taille de la matrice ; sur le graphe le maximum global des 3 courbes est atteint pour une taille supérieure à 35.

2.4 Question 4

Pour l'implémentation de la méthode du gradient conjugué avec pré-conditionnement nous avons construit une matrice M de façon à ce que $M^{-1} \approx A^{-1}$. Pour cela, nous avons utilisé deux méthodes. La première s'appuie sur la factorisation incomplète de Cholesky qui permet de calculer une approximation de T découlant de la factorisation de Cholesky puis permet de remonter à une approximation de A par le produit matriciel $T.^tT$. Une fois M trouvée, nous pouvons conclure par une inégalité sur les normes que M^{-1} tend vers A^{-1} lorsque M tend vers A.
La deuxième méthode de préconditionnement est celle de *Jacobi*, cette méthode est relativement simple à implémenter puisque les éléments de la matrice *Jacobi* J sont $J_{ij} = \frac{\delta_{ij}}{a_{ij}}$.

On teste la méthode de préconditionnement par la vitesse de convergence de la solution x vers la solution exacte. Ci-dessous, la *figure 2* représente la convergence de l'erreur absolue pour la méthode avec préconditionnement et la méthode sans.

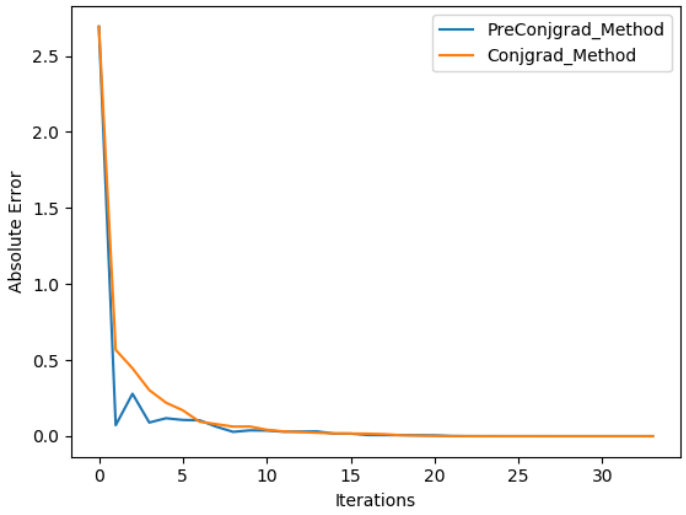

FIGURE 2 – Convergence de l'erreur absolue (sans et avec préconditionnement)

Nous constatons que l'erreur absolue tend vers 0 lorsque le nombre des itérations augmente. Le préconditionnement permet une convergence plus rapide vers la solution exacte.

3 Application à l'équation de la chaleur

3.1 Question 1

La résolution de l'équation de la chaleur peut se réduire à la résolution d'un système linéaire en factorisant par la variable "Température" : T dans l'équation. L'opérateur appliqué à T peut donc s'écrire selon une matrice A qui est "tridiagonale par blocs". Les blocs sont formés par une matrice identité : I et une matrice tridiagonale : T_D qui sont de taille N^2. La décomposition est de la forme suivante :

$$\mathrm{A} = \begin{bmatrix} T_D & I & 0 \\ I & T_D & I \\ 0 & I & T_D \end{bmatrix}$$

Nous obtenons désormais l'équation suivante : $\frac{1}{h^2} AT = f(x, y)$.
La dernière étape consiste à transformer $f(x, y)$ en un vecteur colonne, pour cela, il suffit de remarquer que $\forall (i, j) \in [|1, n|]^2, b[i * N + j] = f(x_i, y_j)$ avec x_i et y_i les ièmes composantes respectives de x et y.
Enfin, nous remarquons que l'équation de la chaleur se met bien sous la forme $AT = b$

3.2 Question 2

Pour modéliser un radiateur central, nous avons implémenté une fonction à deux variables qui renvoie -60 lorsque x et y décrivent un carré central et 0 sinon.
Le vecteur image obtenu est formé de 0 et d'éléments égaux à -60 lorsque x et y coïncident avec le carré central.
Après avoir généré A et b nous avons résolu le système linéaire $AT = b$ avec trois méthodes :

1. La fonction *numpy.linalg.solve* intégrée à **Numpy** *figure 3*
2. La méthode du **gradient conjugué** *figure 4*

3. La décomposition de **Cholesky** *figure 5*

Voici les résultats obtenus :

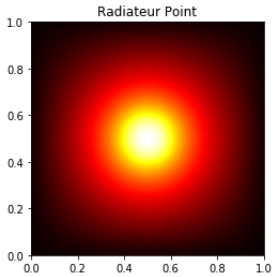

FIGURE 3 – Diffusion de la chaleur pour un radiateur placé au centre de la pièce, résolu avec Numpy

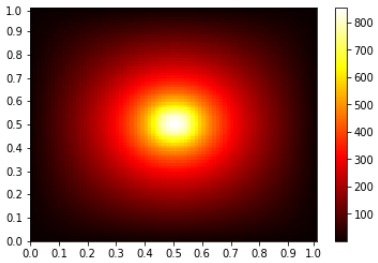

FIGURE 4 – Diffusion de la chaleur pour un radiateur placé au centre de la pièce, résolu avec la méthode du gradient conjugué

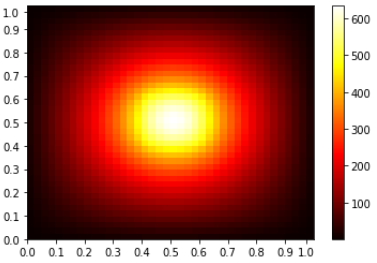

FIGURE 5 – Diffusion de la chaleur pour un radiateur placé au centre de la pièce, résolu avec la méthode de la décomposition de Cholesky

D'un point de vue graphique les résultats semblent assez proches. Pour vérifier ce résultat mathématiquement, nous avons calculé la norme de la matrice produite en utilisant les différentes méthodes. Ensuite, nous avons calculé les écarts relatifs par rapport au résultat de Numpy que nous avons choisi comme référence.
L'erreur relative se définit alors par : $E_{relatif} = \frac{|M_{grad} - M_{nump}|}{|M_{nump}|}$. Pour la méthode du gradient conjugué et pour $N = 20$, $E_{relatif\_gradient} = 1,5.10^{-15}$. En revanche, $E_{relatif\_Cholesky} = 2,2.10^{-15}$. Cet écart montre que la méthode du gradient semble très efficace pour le système linéaire traité comparée à la méthode de Cholesky dans ces configurations. Nous remarquons aussi que la méthode de Cholesky prend beaucoup de temps à s'exécuter ce qui justifie l'emploi d'un nombre N moins élevé.

3.3 Question 3

Dans cette question, il est demandé de modéliser un système composé d'un mur chaud situé au nord. Nous avons implémenté une fonction qui vaut -25 lorsque nous sommes à moins de 0,2cm du bord du nord. Cela représente donc un mur au nord d'épaisseur 0,2cm. Le vecteur image est composé en majorité de 0 et de -25 dès que les coordonnées coïncident avec le mur. Voici les résultats obtenus sur les *figure 6 figure 7* et *figure 8*, avec les trois méthodes de résolution décrites précédemment :

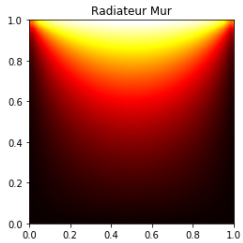

FIGURE 6 – Diffusion de la chaleur pour un mur chaud au nord, résolu avec Numpy

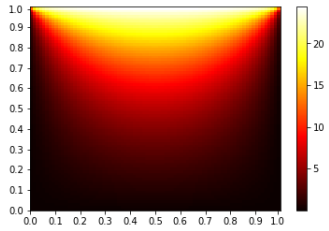

FIGURE 7 – Diffusion de la chaleur pour un mur chaud au nord, résolu avec la méthode du gradient conjugué

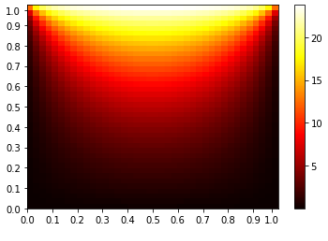

FIGURE 8 – Diffusion de la chaleur pour un mur chaud au nord, résolu avec la décomposition de Cholesky

Nous constatons que les figures sont assez similaires. Nous allons vérifier cela par le calcul de l'écart relatif défini auparavant. Pour N=40, $E_{relatif\_gradient} = 4,7.10^{-16}$ et $E_{relatif\_Cholesky} = 1,6.10^{-15}$. Cela montre que la méthode du gradient est plus efficace pour des systèmes linéaires particuliers (matrice symétrique, définie positive et creuse).