Projet 1
Méthodes du gradient conjugué / Application à l'équation de la chaleur

Groupe 4 - Equipe 8623

Responsable : PREAUT Clément
Secrétaire : PRINGALLE Antoine
Codeurs : DELPEUCH Sébastien, NADIR Souhail

Résumé : Le but de ce projet consiste à implémenter des algorithmes de résolution de systèmes linéaires de grande taille, et à les appliquer à un problème de résolution d'équation aux dérivée partielles. Le problème sous jacent est d'étudier à travers plusieurs algorithmes, la capacité de chacun à résoudre le problème du système linéaire.

1 Décomposition de Cholesky

Dans un premier temps une étude de la décomposition de Cholesky a été réalisée, l'idée est de décomposer la matrice symétrique définie positive A sous la forme d'un produit $T \times T^t$ où T est une matrice triangulaire inférieure. Pour obtenir les différents coefficients de nos matrices nous utilisons les formules fournies par le sujet.

1.1 Elements théoriques sur la décomposition de Cholesky

Avant d'étudier concretement les résultats de notre implémentation, parlons rapidement de la complexité de cette première implémentation de Cholesky.
Soit $C(n)$ le nombre d'opérations pour calculer le facteur de Cholesky d'une matrice d'ordre n. Nous obtenons donc la relation de récurrence suivante $C(n+1) = C(n) + n^2 + 0(n)$ car la résolution du système triangulaire qui calcule t est de complexité $n^2 + O(n)$. Nous déduisons alors

$$C(n) = \frac{1}{3}n^3 + O(n^2) \tag{1}$$

A partir de ce point nous pouvons regarder l'algorithme de résolution d'un système symétrique défini positif. Il comporte 3 étapes

- Factorisation de Cholesky $A = LL^t \quad O(n^3)$
- Résolution du système triangulaire $Ly = b$ (descente) $\quad O(n^2)$
- Résolution du système triangualaire $L^t x = y$ (montée) $\quad O(n^2)$

L'algorithme a ainsi une complexité polynomiale cubique, ce qui permet de résoudre des systèmes avec plusieurs milliers d'inconnues.

1.2 Mise en place de test

Pour commencer nous devons regarder si la factorisation de Cholesky produit un résultat correct ou non (nous entendons par correct un résultat qui n'est que peu éloigné de la "vraie" valeur . Pour cela nous allons étudier l'erreur relative entre le calcul que nous effectué et la valeur que nous calcul la fonction *numpy.linalg.cholesky*. Nous définissons alors l'erreur relative pour une matrice en regardant l'erreur relative sur chacun de ses membres

$$\delta_{ij} = \frac{|x_{ij}^{\text{calcul}} - x_{ij}^{\text{numpy}}|}{x_{ij}^{\text{numpy}}}$$

Pour réaliser les figures nous conservons uniquement $\delta_{\max} = \max(\delta_{i,j})$. Nous remarquons aisement sur la figure 1 que l'erreur relative est très proche de 0. Nous pouvons donc nous conclure que notre approche nous donne un résultat très proche de celui de *numpy*.

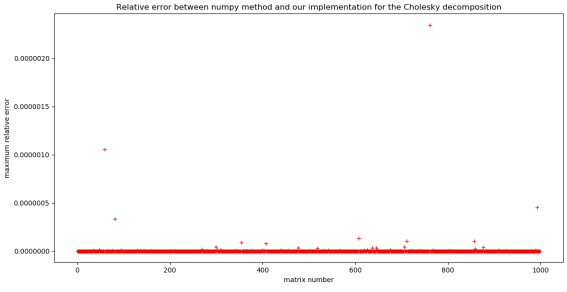

FIGURE 1 – Erreur relative $\delta_{\max}$ de la factorisation de cholesky pour 1000 matrices de dimension 100

Passons maintenant à la résolution du système linéaire. Pour ce faire nous allons regarder deux variables. Tout d'abord nous voulons savoir si notre résultat est similaire au résultat attendu, nous allons donc encore une fois étudier l'erreur relative entre la valeur produite par numpy et notre

résultat (c'est à dire, la factorisation de Cholesky, la montée et la descente par nos algorithmes). Nous réutilisons l'erreur relative δ_{max} que nous avons défini précédemment.

Nous réalisons des tests. Ces tests vont se diviser en deux parties, une partie où l'on fixe la matrice A et nous faisons varier le vecteur b puis nous regardons le $\delta_{\max}$ et une partie où nous faisons l'inverse. Voici les résultats que nous obtenons

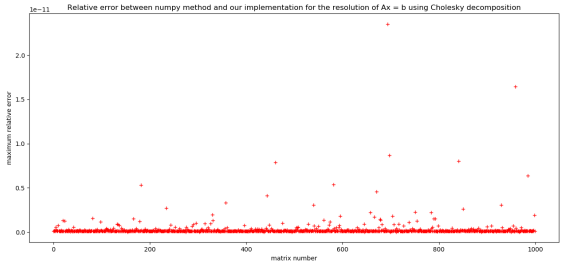

FIGURE 2 – Erreur relative $\delta_{\max}$ pour A fixé et b variant

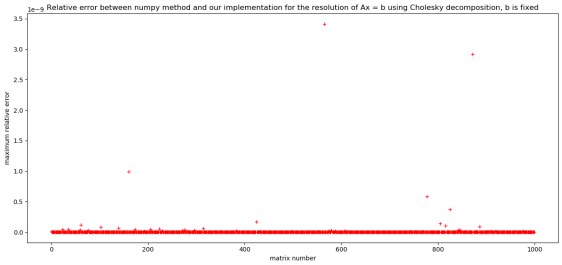

FIGURE 3 – Erreur relative $\delta_{\max}$ pour b fixé et A variant

Nous pouvons alors voir que la résolution de notre système est en accord avec celle de numpy, nous avons des erreurs relatives de l'ordre de 10^{-11} pour notre premier test et de l'ordre de 10^{-9} pour le deuxième. En somme nous pouvons conclure que la méthode que nous avons implémenté est précise.

Un autre argument est que les erreurs relatives sont majorées par le conditionnement. La formule du conditionnement est

$$C_A(b) = \lim_{b=0} \frac{||\delta x||}{||x||} / \frac{||\delta b||}{||b||} = \lim \frac{||\delta x||}{||\delta b||} \times \frac{||b||}{||x||} \tag{2}$$

Cette formule de conditionnement dans notre cas peut être majorée

$$C_A(b) \leq ||A^{-1}|| \times |||A||| = \frac{\sigma_{\max}}{\sigma_{\min}} = \frac{|\lambda_{\max}|}{|\lambda_{\min}|} \tag{3}$$

Où λ désigne la valeur propre et σ la valeur singulière. Cette majoration du conditionnement nous renforce dans l'idée que l'erreur relative commise par notre algorithme ne va pas "exploser".

Il reste cependant une autre variable interessante à tester, la stabilité numérique, c'est à dire à quel point les résultats diffèrent si nous modifions légèrement la valeur d'entrée. Pour cela nous allons fixer arbitrairement A et b et trouver x via nos algorithmes puis réaliser des petites variations sur b et regarder si la nouvelle valeur de x "explose". Nous obtenons le graphique 4.

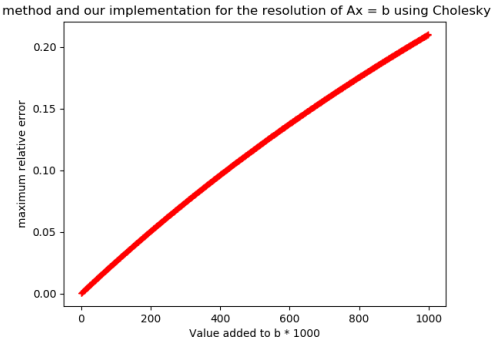

FIGURE 4 – Graphique représentant la résolution d'un système où A et b sont fixes, en faisant varier b

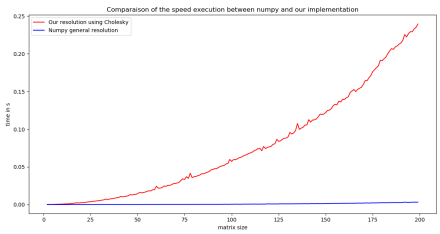

FIGURE 5 – Temps de calcul de résolution de $Ax = b$ avec la méthode de Cholesky et la méthode de numpy

Nous pouvons observer ici une augmentation linéaire de l'erreur, autrement dit il n'y a pas "d'explosion" nous avons donc la preuve empirique que l'algorithme de Cholesky est un algorithme numériquement stable, ce qui en fait un très bon algorithme pour la résolution de système linéaires. Pour finir sur l'algorithme de Cholesky, maintenant que nous avons montré qu'il est stable et qu'il fournit une solution plutôt précise. Nous allons maintenant comparer son temps de calcul au temps de calcul de numpy.

La figure 5 nous montre clairement que notre méthode de calcul prend beaucoup plus de temps que celui de numpy (qui est constant). Ainsi même si nous avons des résultats similaires à ceux de numpy nous n'avons pas du tout un algortihme optimal.

Nous pouvons rapidement évoquer l'algortihme de Cholesky incomplet. Son principal avantage par rapport à l'algorithme de Cholesky précédent est son temps de calcul. Nous pouvons donc étudier cet avantage. Pour mettre en évidence cela nous mettons en parallèle le temps de calcul des 2 versions de Cholesky sur des matrices aléatoires de taille 100 en faisant varier le nombre de termes extra diagonaux non nuls. Cela nous donne le résultat suivant.

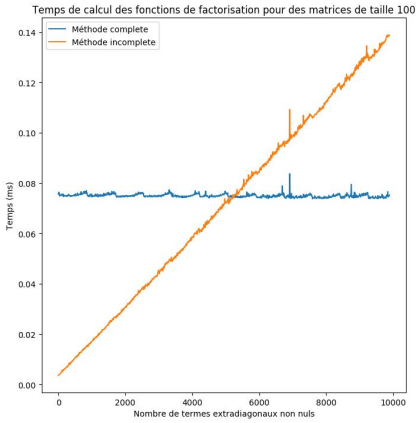

FIGURE 6 – Temps de calcul des deux algorithmes pour des matrices de taille 100

Nous pouvons remarquer que le temps de calcul pour l'algorithme complet, courbe bleue, est constant (ce qui est normal, l'algorithme fais le même travail peu importe la matrice d'entrée). En revanche la version incomplète possède un temps visiblement linéaire, courbe orange, (ce qui est normal puiqu'il ignore les valeurs nulles). Nous pouvons clairement voir que le deuxième algorithme de Cholesky à un temps de calcul inférieur lorsqu'il y a beaucoup de valeurs extra diagonales nulles. Cela le privilégie donc face au premier pour une implémentation où nous savons que nous sommes en présence de matrices creuses avec peu de termes extra diagonaux non nuls.

2 Méthode du gradient conjugué

Maintenant que nous avons étudié une première méthode pour résoudre un système linéaire nous allons rapidement évoquer la méthode du gradient conjugué. Cette méthode ne se base pas sur la décomposition d'une matrice en deux matrice triangulaires inférieures mais en utilisant une méthode itérative qui converge vers un vecteur solution. Le principe de cette méthode est de minimiser la fonction $f : x \mapsto \frac{1}{2}(Ax, x) - (b, x)$. Cela nous permettra de trouver une solution du problème $Ax = b$, en effet nous avons $\nabla f(x) = Ax - b$. Nous utilisons une méthode itérative pour résoudre ce problème.

L'implémentation itérative proposée ne respecte pas les standars de codage très sains puisque le code proposé est assez indigeste, difficile à lire à cause du manque de commentaire, du manque d'espace entre les opérateurs et des noms de variables peu explicites.

Cependant ce qui pourrait être reproché ici est que la condition de sortie de la boucle est implémentée en dur alors que cela aurait pu être paramétrable, lors de l'appel de la fonction par exemple.

Nous pouvons mettre en évidence la convergence de l'algorithme vers la solution pour ce faire nous choisissons arbitrairement A et b et nous faisons varier la condition de sortie de l'algorithme (c'est à dire la "précision" de la solution). Nous calculons alors l'erreur relative $\delta_{\max}$.

Nous pouvons voir sur la figure 7 que plus nous augmentons la précision demandée plus nous avons une erreur relative faible. Dans tous les cas cette erreur relative est inferieur à 2×10^{-9} ce qui est relativement faible ce qui nous conforte dans l'idée que la méthode fonctionne. Nous n'irons pas plus loin dans l'étude de cette méthode et nous allons directement passer à l'application à l'équation de la chaleur.

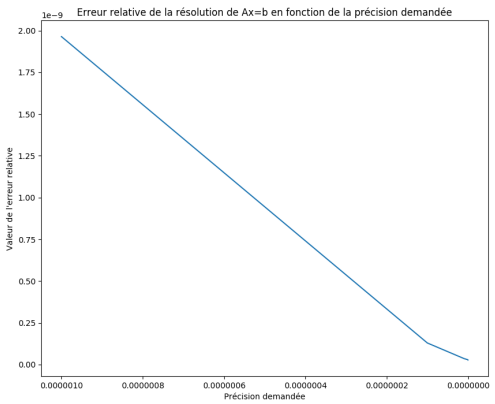

FIGURE 7 – Graphe représentant la solution pour A et b fixé en faisant varier la précision de la solution

3 Application à l'équation de la chaleur

La troisième partie s'intéresse à la résolution d'un système linéaire particulier : celui de l'équation de la chaleur en mode stationnaire. En suivant la méthode des différences finies, nous avons représenté un espace continu en une grille de points ayant chacun une température. L'équation de la chaleur s'écrit de la façon suivante

$$\frac{\delta^2 T}{\delta x^2} + \frac{\delta^2 T}{\delta y^2} = f(x, y) \tag{4}$$

Où f est l'apport de la chaleur et $T(x, y)$ la température au point (x, y).

3.1 Objectif

Nous souhaitons appliquer les fonctions implémentées précédemment dans la résolution de l'équation de la chaleur. Cette exemple concret permet de comparer les résultats de nos implémentations à ceux qu'on obtient avec les fonctions de résolution déjà existantes, qui servent de référence.

3.2 Modélisation du problème

L'idée est de considérer une surface discrétisée, dans laquelle chaque point possède une température et est soumis à une chaleur. Nous noterons h le pas entre deux points de la surface. h est donc une distance infinitésimale. Cela permet d'approximer les dérivées partielles aux coordonnées (i, j) de la sorte :

$$\left(\frac{\partial T}{\partial x^2}\right)_{i,j} \approx \frac{t_{i+1,j} + t_{i-1,j} - 2 \times t_{i,j}}{h^2} \tag{5}$$

$$\left(\frac{\partial T}{\partial y^2}\right)_{i,j} \approx \frac{t_{i,j+1} + t_{i,j-1} - 2 \times t_{i,j}}{h^2} \tag{6}$$

Nous estimerons par ailleurs que les températures en bordures de surface sont nulles. L'enjeu ensuite et de mettre ce problème sous la forme d'un système linéaire $A \times X = b$

La matrice A est une matrice tridiagonale par blocs de taille $N^2 \times N^2$ dont la forme est donnée par le sujet. La solution X est un vecteur de N^2 lignes dont les coefficients sont les températures aux différents points de notre discrétisation de la surface.

Le sujet demande de traiter deux situations : un radiateur chaud placé au centre du carré ou au nord du carré. Deux fonctions f représentant l'apport en chaleur permettent de représenter ces situations. Pour le premier cas nous avons choisi $f(x, y) = e^{-\sqrt{x^2+y^2} \times T}$ où T est la température constante du radiateur. Pour le deuxième cas nous avons choisi $f(x, y) = e^{-(y+l) \times T}$ où l est la longueur d'un côté du carré.

Ces deux fonctions permettent de générer le vecteur b du système linéaire. En effet, une boucle parcourant les différents points de notre discrétisation de la surface permet d'affecter à chaque coefficient l'apport en chaleur au point qui lui correspond.

Il faut alors générer la matrice A nous générons tout d'abord une matrice nulle de taille $N^2 \times N^2$. Ensuite nous modifions ses coefficients. Tous les coefficients diagonaux $A_{i,i}$ sont modifiés en la valeur -4. De plus nous avons

$$A_{i,j+1} = A_{i+1,j} = 1 \; \forall 1 \leq i \leq N^2 - 1 \text{ et } 1 \leq j \leq N^2 - 1 \tag{7}$$

Finalement nous pouvons terminer de compléter la matrice A avec

$$A_{i,j+N} = A_{i+N,j} = 1 \; \forall \; 1 \leq i \leq N^2 - N \text{ et } 1 \leq j \leq N^2 - N \tag{8}$$

A ce stade nous avons une manière d'avoir les matrices A et b pour le système linéaire. Nous allons maintenant résoudre le système avec les différentes méthodes.

3.3 Résolution des deux systèmes

Nous souhaitons d'abord obtenir une résolution de référence grâce aux foncions fournies par numpy. Nous calculons alors A et b comme nous l'avons précédemment expliqué, puis nous appelons la fonction *numpy.linalg.solve* pour résoudre le problème linéaire. Nous créons ensuite les graphes en couleur suivant

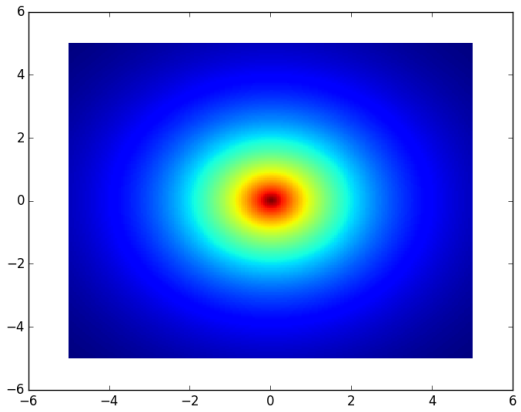

FIGURE 8 – Graphique représentant la répartion de la température avec une source de chaleur au centre en utilisant *numpy.linalg.solve*

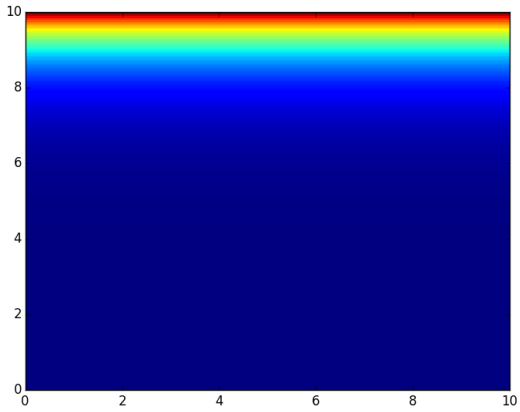

FIGURE 9 – Graphique représentant la répartition de la température avec une sour de chaleur au nord en utilisant *numpy.linalg.solve*

Les résultats que nous avons avec *numpy* sont totalement cohérents, nous allons maintenant réaliser des schémas identiques avec la méthode de Cholesky complet que nous avons présenté au début. Nous choisissons cette méthode pour rester cohérent, en effet c'est la méthode que nous avons le plus analysé, il est donc logique de continuer l'analyse avec cette dernière.

Nous réalisons alors les deux graphiques suivants

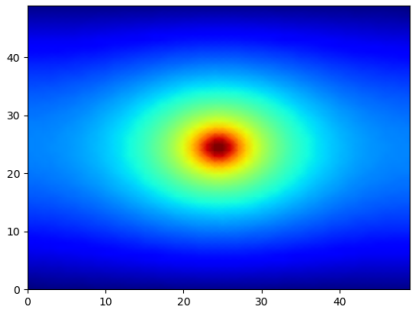

FIGURE 10 – Graphique représentant la répartion de la température avec une source de chaleur au centre en utilisant nos algorithmes

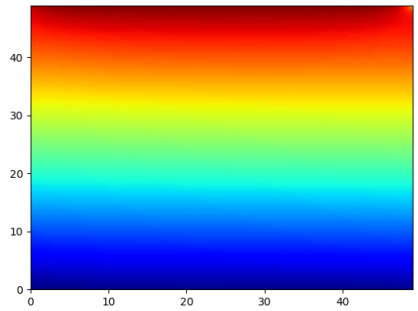

FIGURE 11 – Graphique représentant la répartition de la température avec une source de chaleur au nord en utilisant nos algorithmes

Nous pouvons alors comparer les différentes figures. Nous voyons que nous obtenons avec nos algorithmes des résultats tout autant cohérents (une légère inclinaison est perceptible). Nous pouvons donc conclure que notre implémentation de la résolution d'un système $Ax = b$ avec nos implémentations est relativement précise.