Projet n 2
Méthode du gradient conjugué / Application à l'équation de la chaleur

Groupe n 1 - Equipe n 3

Responsable : Patry
Secrétaire : Marcelin
Codeurs : Benkirane, Raffin, Doghmi

Résumé : Ce projet consiste à découvrir et à comprendre des algorithmes efficaces (Cholesky et méthode du gradient conjugué) pour résoudre des systèmes linéaires de grande taille puis à appliquer ces algorithmes à la résolution de l'équation de la chaleur.

1 *Décomposition de Cholesky*

1.1 Implémentation et compléxité :

La décomposition de Cholesky a pour objectif de mettre une matrice symétrique définie positive A sous la forme $A = T * transpose(T)$ où T est une matrice triangulaire inférieure
Les coefficients de T sont définis pas les formules suivantes :

$$\begin{cases} t_{j,j}^2 = a_{j,j} - \sum_{k=1}^{j-1} t_{j,k}^2 \\ t_{i,j} = \frac{a_{i,j} - \sum_{k=1}^{j-1} t_{j,k} t_{i,k}}{t_{j,j}} (j \leq i) \end{cases}$$

On a 3 boucles imbriquées. Pour une matrice de taille $n * n$, si on compte le nombre de tours de boucle que l'on a : $1 * 1 + 2 * 2 + 3 * 3 + ... + (n+1) * (n+1)$ soit $\sum_{i=1}^{n+1} i^2 = \frac{(n+1)(n+2)(2n+3)}{6}$ soit une complexité en $O(n^3)$.

1.2 Génération de matrices creuses :

Afin de générer des matrices symétriques définies positives on applique l'algorithme suivant :

Entrées : N :Taille de la matrice ; k : termes extra-diagonaux non nuls
Sorties : S :matrices symétriques définies positives creuses

$B \leftarrow Matrice\_aleatoire(N)$; // tq $0 < b_{i,j} \leq 1$;
$nbr\_zeros \leftarrow N * (N-1) - k$;
$B \leftarrow mettre\_zero\_aleatoirement\_extra\_diagonale(B, nbr\_zeros)$;
$S \leftarrow S + transpose(S)$;
$S \leftarrow S + N * identite(N)$;
$S \leftarrow 10 * S$;
return S ;

Algorithme 1 : algorithme permettant de générer des matrices symétriques définies positives creuses

L'algorithme de Cholesky incomplet est implémenté de facon similaire à celui de Cholesky avec une petite différence à savoir l'évaluation des termes non nuls dans A. En ce qui concerne l'évaluation des termes non-nuls gagnés dans Cholesky incomplet, la fonction **terme_non_null_gagne** fournie dans le fichier python permet de calculer le nombre de termes gagnés par rapport à la factorisation dense.

1.3 Préconditionnement

Il s'avère donc que la méthode de Cholesky n'est pas assez efficace pour résoudre le système linéaire $Ax = b$, mais plutôt utile pour trouver rapidement une matrice M facile à inverser tel que son inverse est proche de l'inverse de la matrice A. A l'aide de cette matrice qui représente le préconditionneur, on saura résoudre le système en multipliant les deux membres par M^{-1} pour avoir au final un système mieux conditionné. A l'aide de la fonction **compare_preconditionneur** (voir fichier python), on en déduit que la méthode de Cholesky est un bon préconditionneur vu qu'il est proche du conditionnement de la matrice A.

2 *Méthode du gradient conjugué*

2.1 Implémentation ne respectant pas les règles de codage saines : Avantages et inconvénients

Cette méthode de codage est spéciale : on boucle au plus 10^6 fois ou jusqu'à ce que ce racine de l'erreur soit inférieure à 10^{-10}. L'avantage est qu'on est sûr de ne pas boucler pendant trop longtemps si on demande une précision trop grande. Cependant, ce n'est pas très propre d'écrire des constantes qui n'ont pas vraiment de sens dans le code. Ainsi, il faudrait plutôt définir des variables globales en début de programme telles que :
CONDITION_ARRET pow(10, 6)
EPSILON pow(10, -10)
Puis, demander une valeur à l'utilisateur pour ces variables et utiliser les valeurs prédéfinies s'il n'en rentre pas. En outre, la boucle for fourni dans cet algorithme diffère du for usuel à cause du break qui rend moins visible la condition d'arrêt et on sort de la boucle for avant d'effectuer toutes les itérations. Une boucle while serait plus efficace pour bien visualiser la condition d'arrêt et pour respecter les normes du codage saines.
Toutefois, l'algorithme fourni présente un grand avantage du point de vue complexité où on effectue au plus 10^6 itérations, par contre dans la méthode décrite mathématiquement, on ne sort de la boucle que lorsqu'on a la précision voulue. Ceci dit que malgré le gain de la complexité dans la méthode fournie, on perd au niveau de la précision.

La méthode avec préconditionneur est plus optimale que celle de cholesky incomplet avec préconditionneur étant donné qu'on gagne en moyenne de 6 à 7 itérations. La complexité de la méthode avec préconditionneur reste en $O(n^2)$.

3 *Application à l'équation de la chaleur*

La solution exacte d'un problème d'une équation aux dérivées partielles est une fonction continue. Les ordinateurs ne connaissent que le fini et le discret. En effectuant un calcul numérique, un ordinateur ne peut retenir qu'un nombre fini de chiffres pour représenter les opérandes et les résultats des calculs intermédiaires. Les solutions approchées seront calculées comme des ensembles de valeurs discrètes sous la forme de composantes d'un vecteur solution d'un problème matriciel.
Pour illuster cette approche nous allons voir un exemple d'application qui est l'equation de la chaleur en 2 dimension stationnaire avec source de chaleur :

Considérons le problème bidimensionnel stationnaire de la conduction de la chaleur dans un domaine rectangulaire [0,1] x [0,1] et avec l'apport d'une chaleur extérieure f, le champ de température T(x,y) vérifie l'équation de Laplace :

$$\begin{cases} \Delta T = \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = f(x,y) & \forall (x,y) \in [0,1]\text{x}[0,1], \\ \text{Conditions aux bords de Dirichlet} : \begin{cases} T(0,y) = T(1,y) = 0 \\ T(x,0) = T(x,1) = 0 \end{cases} & \forall (x,y) \in [0,1]\text{x}[0,1]. \end{cases}$$

Pour passer du problème exact continu au problème approché discret, on utilise la méthode des différences finies, qui consiste à approximer les dérivées de l'équation et qui se déduisent directement de la définition de la dérivée.

On discrétise le plan, on repésente donc les fonctions T et F par deux matrices carrées tel que $t_{i,j}$ et $f_{i,j}$ sont les valeurs de F et T en $(\frac{i}{N+1}, \frac{j}{N+1})$
On en déduit ensuite les formules des dérivées partielles approchées :

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

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

où h représente la distance entre deux points consécutifs sur la même ligne ou la même colonne, dont l'expression en fonction de N est donnée par :

$$h = \frac{1}{N+1} \tag{3}$$

En remplaçant les formules des dérivées partielle dans l'equation de la chaleur on obtient :

$$(N+1)^2 * (t_{i+1,j} + t_{i-1,j} + t_{i,j+1} + t_{i,j-1} - 4 * t_{i,j}) = f_{i,j} \tag{4}$$

La relation $T_{i,j} \to T_{N*i+j}$ permet de transformer la matrice T en un vecteur de taille N^2 (de même pour F). On peut donc réaliser l'opération $(N+1)^2 * M_c.T = F$ qui nous permettera de retrouver l'équation (4)
Il reste a résoudre le sytème linéaire $(N+1)^2 * M_c.T = F$ par l'une des méthodes traitées auparavant

Repréntation de la fonction T :
On représente la matrice de la fonction T en 2D sur un plan carré montrant la variation de la température et en 3D en prenant l'axe z comme axe de température.

FIGURE 1 – Mur chaud au nord du carré

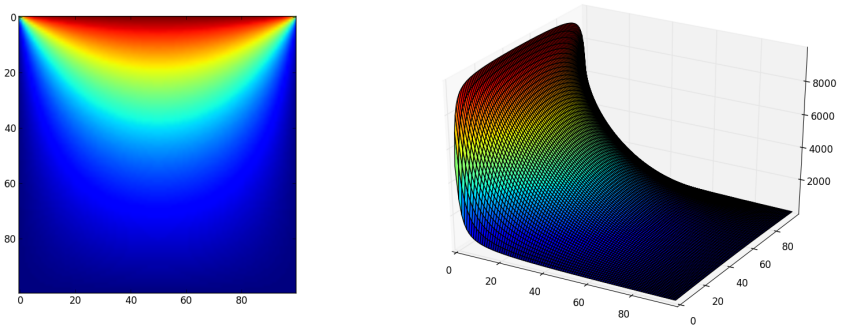
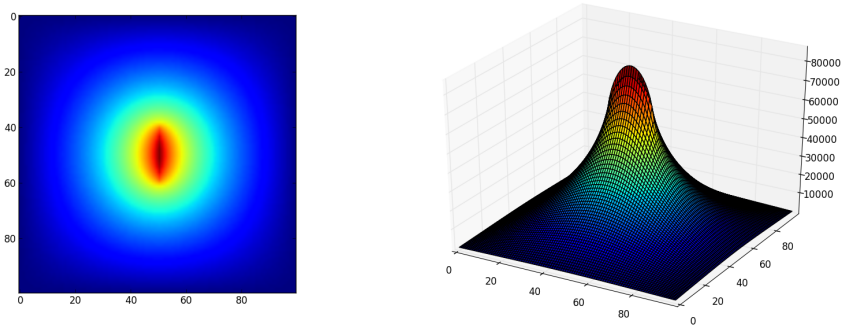


FIGURE 2 – Radiateur placé au centre du carré



Conclusion
Malgré le fait que cette méthode ait pu nous donner une solution approchée de l'équation de la chaleur, celle-ci est obtenue par une approximation des $\frac{\partial^2 T}{\partial x^2}$ et $\frac{\partial^2 T}{\partial y^2}$ par l'utilisation de la formule de Taylor tronquée a l'ordre 2. Nous avons donc commis une erreur de l'ordre de h^2.
Non seulement cette erreur commise est considérable mais le calcul des éléments du vecteur x (l'inconnu dans Ax = b) est de plus en plus couteux en mémoire.
Par exemple pour N = 100 c'est à dire h^2 de l'ordre de 10^{-5} la matrice Mc est de taille (10^4 x 10^4) et contient 10^8 éléments donc on a besoin de plus de $8 * 10^8$ octets à peu près 1 Giga de mémoire. Or pour affiner beaucoup plus la solution numérique vers la solution analytique, nous devons faire tendre h vers 0 ce qui augmentra encore la nécessité en mémoire.
Pour la résolution de $\frac{1}{h^2} * M_c T = F$ on a utilisé la methode du gradiant conjugué car moins coûteuse en terme de complexité que la méthode de cholesky.

A travers ce projet, nous avons approfondi la factorisation et la décomposition de matrices, permettant la résolution de systèmes linéaires. Nous avons, entre autres, pu les appliquer au problème concret qu'est l'équation de la chaleur nous permettant par la même occasion de manipuler le calcul matriciel et la représentation graphique des matrices sous *python*.