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

Groupe 1 - Equipe 3

Responsable : mbailleul001
Secrétaire : mzizi001
Codeurs : amdahmani, ndo001, helfani

Résumé : Ce projet s'intéresse à la résolution des équations linéaires. Il traite spécifiquement le cas des matrices symétriques définies positives et creuses. La première partie s'intéresse à la décomposition de Cholesky des matrices S.D.P. La deuxième partie cherche à résoudre un système linéaire en utilisant la méthode du gradient conjugué. La troisième partie utilise les résultats des parties précédentes pour résoudre des cas spécifiques de l'équation de chaleur en la transformant en système linéaire.

Table des matières

1 Décomposition de Cholesky

1.1 Résolution d'un système linéaire

La décomposition de Cholesky d'une matrice A symétrique définie positive consiste à trouver une matrice L triangulaire telle que : $A = LL^T$. Nous avons réussis à implémenter cette décomposition en algorithme de complexité temporelle $O(n^3)$ (n étant la taille de la matrice A). Après avoir trouvé la matrice L la résolution d'un système linéaire $Ax = b$ se traduit de la façon suivante :

$$LL^T x = b \Longleftrightarrow \begin{cases} Ly = b \\ L^T x = y \end{cases}$$

La résolution de ces deux systèmes est simple puisque les deux matrices L et L^T sont triangulaires. Nous avons réussi à l'implémenter en une complexité de $O(n^2)$. La résolution du système $Ax = b$ se fait donc en une complexité totale de $O(n^3)$.

1.2 Générateur de matrices S.D.P

Pour générer des matrices symétriques définies positives creuses, nous avons d'abord commencé par générer une matrice diagonale à coefficients aléatoires strictement positives. Nous commençons ensuite à injecter des coefficients extra-diagonaux aléatoires non nuls de façon symétrique tout en vérifiant que la matrice reste définie positive. Si ce n'est pas le cas nous enlevons les coefficients que nous venons d'insérer et nous recommencons. Cela continue jusqu'à l'obtention du nombre de coefficients extra-diagonaux non nuls demandé.

Pour éviter que cet algorithme ne se termine jamais dans des cas où il est impossible de construire une matrice S.D.P avec les coefficients insérés dans les étapes précédentes, s'il dépasse un certain nombre d'itérations (Nous avons choisi 10 fois le nombre de coefficients extra-diagonaux non nuls) l'algorithme abandonne la matrice qu'il est en train de construire et recommence.

1.3 Décomposition de Cholesky incomplète

La version incomplète de Cholesky ignore le calcul de certains éléments afin de diminuer sa complexité temporelle ($O(m)$ où m est le nombre de coefficients non nuls de la matrice au lieu de $O(n^2)$). Cette réduction se fait au prix de l'approximations du résultat final. Avec cette méthode, la résolution du système linéaire se fait en $O(m * n)$.

1.4 Préconditionneur

Pour tester la qualité de $M = LL^T$ comme préconditionneur, nous avons comparé $cond(M^{-1}A)$ et $cond(A)$. Nous avons obtenu les résultats suivants :

$cond(M^{-1}A)$ dense	~ 1	~ 1	~ 1	~ 1	~ 1	~ 1	~ 1
$cond(M^{-1}A)$ incomplète	10.7	127.5	5.95	308.39	232.52	2204.87	660.95
$cond(A)$	1484.3	1536.2	25.9	1241.06	909.66	8571.31	7213.5

Dans tous ces cas, on a pour Cholesky incomplète : $cond(M^{-1}A) < cond(A)$. Cette méthode produit donc des préconditionneurs de bonne qualité.

1.5 Les tests

— Les tests des fonctions *cholesky_dense*() et *cholesky_incomplete*() consistent à choisir une variété de matrices A (entière, flottante et identité) et de générer leur décomposition respectives L. Nous vérifions ensuite que L est triangulaire (même inférieur dans notre cas) et que $LL^T = A$.
— Pour le générateur de matrices S.D.P, nous avons créé une fonction $is\_positive\_definite()$ qui détermine si une matrice est définie positive en calculant les déterminants de ses sous-matrices.
— Pour la fonction *solve_cholesky*(), on compare les résultats qu'elle fournit sur différentes matrices avec les résultats de la fonction *numpy.linalg.solve*() de numpy.

2 Méthode du gradient conjugué

2.1 Principe

Au contraire de la méthode Cholesky, pour résoudre les systèmes linéaires, la méthode du gradient conjugué prend un vecteur de départ x et le modifie graduellement pour converger vers la solution. Cette méthode peut donc s'arrêter à n'importe quel moment pour donner une valeur approximative de la solution.

2.2 Défauts de l'algorithme proposé

L'algorithme proposé contient un $if(A)$: *break* à l'intérieur d'une boucle for ce qui (bien que fonctionnel) ne respecte pas les standards du codage très sains. Une solution alternative est de remplacer la boucle for par une boucle *while* ayant deux conditions d'arrêt :

$$
\begin{array}{|l}
\hline
\vdots \\
rnew = 1 \\
i = 0 \\
while(sqrt(rsnew) > 1e-10 \ and \ i < 10e6) : \\
\quad \vdots \\
\quad i+ = 1 \\
\vdots \\
\hline
\end{array}
$$

Le choix de la valeur initiale de *rnew* est arbitraire et n'affecte pas l'exécution de l'algorithme. Il faut juste que cette valeur permette une première exécution de la boucle *while*.

2.3 Préconditionnement

Pour accélérer l'algorithme précédent, nous avons implémenté la méthode avec préconditionnement. Vu les résultats de la première partie, nous avons pris comme préconditionneur LL^T avec L le résultat de la décomposition de Cholesky. Nous avons comparé la vitesse des versions avec et sans préconditionnement en prenant comme référence la norme du vecteur r à chaque itération puisqu'elle détermine l'arrêt de l'algorithme.

La figure 1 montre clairement l'avantage qu'apporte la version avec préconditionnement, celle-ci se terminant en 4 itérations alors que la version initiale en prend 267.

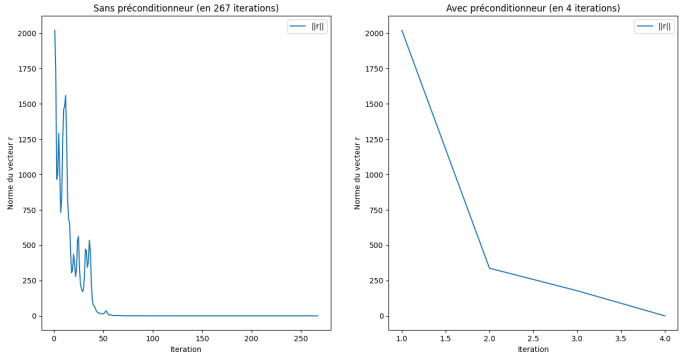

FIGURE 1 – Vitesse de convergence avec et sans préconditionnement

2.4 Les tests

Comme pour la fonction *solve_cholesky*() de la partie précédente, on compare ici les résultats de nos deux méthode avec ceux de la fonction *numpy.linalg.solve*() de numpy.

3 équation de la chaleur

3.1 Forme matricielle

On essaye de trouver une solution T à l'équation :

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

En posant les matrices suivantes : $\forall (i, j) \in [\![0\ ;\ N-1]\!]^2$

$$T, F \in \mathcal{M}_{N^2,1}(\mathbb{R}), \quad T_{i*N+j} = T(x_i, y_j) \quad F_{i*N+j} = f(x_i, y_j)$$

Et en approximant la dérivée seconde comme suit :

$$f''(x) = \frac{f(x+h)+f(x-h)-2f(x)}{h^2}$$

L'équation devient : $\forall (i, j) \in [\![1\ ;\ N-2]\!]^2$

$$\frac{T_{(i+1)N+j}+T_{(i-1)N+j}+T_{iN+(j+1)}+T_{iN+(j-1)}-4T_{iN+j}}{h^2} = F_{iN+j}$$

(Remarque : Pour respecter les conditions aux bords, on prend 0 au lieu des termes $T_{(i-1)N+j}$, $T_{(i+1)N+j}$, T_{iN+j-1} et T_{iN+j+1} si $i = 0$, $i = N-1$, $j = 0$ ou $j = N-1$ respectivement).
Supposons qu'il existe une matrice $Mc \in \mathcal{M}_{N^2,N^2}(\mathbb{R})$ vérifiant :

$$\frac{1}{h^2} Mc * T = F$$

En calculant le produit matriciel on obtient : $\forall (i, j) \in [\![1\ ;\ N-2]\!]^2$

$$\frac{1}{h^2}\sum_{k=0}^{N^2-1}(Mc_{iN+j,k} * T_k) = F_{iN+j}$$

$$\frac{1}{h^2}\sum_{k=0}^{N^2-1}(Mc_{iN+j,k} * T_k) = \frac{T_{(i+1)N+j}+T_{(i-1)N+j}+T_{iN+(j+1)}+T_{iN+(j-1)}-4T_{iN+j}}{h^2}$$

$$\sum_{k=0}^{N^2-1}(Mc_{iN+j,k} * T_k) = T_{(i+1)N+j} + T_{(i-1)N+j} + T_{iN+(j+1)} + T_{iN+(j-1)} - 4T_{iN+j}$$

(La remarque précédente s'applique ici).

Et donc, en prenant $p = iN + j$:

$$\sum_{k=0}^{N^2-1}(Mc_{p,k} * T_k) = T_{p+N} + T_{p-N} + T_{p+1} + T_{p-1} - 4T_p$$

En faisant correspondre les termes T_k des deux côtés on trouve alors les coefficients de la matrice Mc :

$$Mc_{p,k} = \begin{cases} -4 & \text{Si p = k} \\ 1 & \text{Si k} \in \{\text{ p + N, p - N, p + 1, p - 1 }\} \\ 0 & \text{Sinon} \end{cases}$$

Nous avons ainsi retrouvé la forme de la matrice Mc proposée par l'exercice. Il est important de noter pour la prochaine sous-partie que cette matrice est S.D.P.

3.2 Application

Puisque la résolution de l'équation de la chaleur en régime stationnaire revient à résoudre un système linéaire $Mc * T = F$ où Mc est S.D.P, il suffit d'appliquer la méthode de la partie précédente.

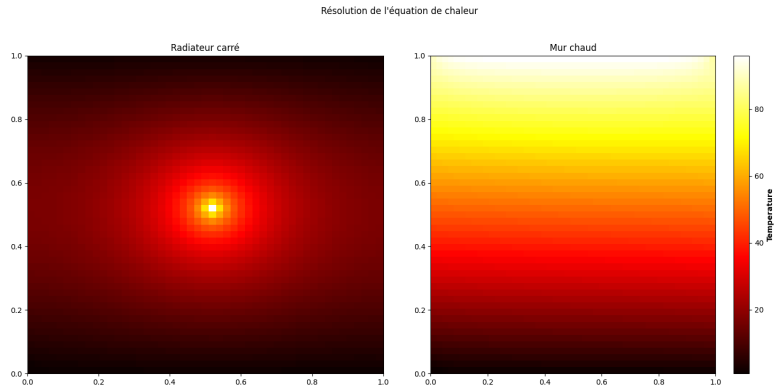

FIGURE 2 – Résolution de l'équation de chaleur

La matrice F ici représente les sources de chaleur (en signe négative). Pour simuler les cas demandés il faut choisir F à coefficients nuls sauf pour :

— Pour le radiateur au milieu du plan : $i = j = \left\lfloor \frac{N}{2} \right\rfloor$, $F_{iN+j} = -temp$

— Pour le mur chaud au nord : $\forall j \in [\![0\ ;\ N-1]\!]$, $F_{(N-1)N+j} = -temp$

La figure 2 montre les résultats obtenus en suivant cette démarche.