Institut Galilée

INFO1
2023-2024

Algorithmique
Devoir

1 Modalités

Le devoir est à déposer *individuellement* et *obligatoirement* sur l'ENT avant le **18 décembre 2023**. Il sera impossible de rendre ce devoir après cette date, et cela entraînera donc la note de 0 (tout plagiat sera noté de la même manière). Dans le cas de devoirs trop semblables, la note obtenue sera divisée par le nombre d'étudiants concernés.

On déposera un seul fichier (.tar.gz ou .zip) contenant :

— un rapport succinct en pdf (réponses aux questions du devoir). Une version manuscrite numérisée est tout à fait acceptable mais attention à la taille du fichier produit !

— les sources (.c et .h) et makefile éventuel

— quelques tests numériques (à inclure dans le rapport ou sous la forme de fichiers texte)

— Un fichier texte contenant des indications sur la façon de compiler/lancer le programme sera le bienvenu (arguments nécessaires, fichiers d'entrée utilisés, etc...). Aucun fichier binaire (exécutable) ne sera fourni.

2 Objectifs et résultats préliminaires

On souhaite élaborer un algorithme efficace d'inversion d'une matrice A réelle $n \times n$ en utilisant une approche de type "Diviser pour régner". Pour simplifier, on supposera dans ce devoir que A est de taille $2^k \times 2^k$ avec k entier naturel.

1. Montrer que si A est une matrice (réelle) inversible $n \times n$ alors $A^T A$ est symétrique et définie positive, i.e. $\forall x \neq 0 \in \Re^n \; x^T A^T A x > 0$.
2. En déduire que $A^{-1} = (A^T A)^{-1} A^T$, et donc que pour calculer l'inverse (si elle existe) d'une matrice quelconque, il suffit de savoir inverser les matrices définies positives. Le coût supplémentaire sera alors constitué d'une transposition et de deux multiplications.
3. On découpe $A^T A$ en blocs de taille $n/2$. Montrer que $A^T A = \begin{bmatrix} B & C^T \\ C & D \end{bmatrix}$ définie positive équivaut à $S = D - CB^{-1}C^T$ définie positive.
4. En déduire que $(A^T A)^{-1} = \begin{bmatrix} B^{-1} + B^{-1}C^T S^{-1} C B^{-1} & -B^{-1}C^T S^{-1} \\ -S^{-1} C B^{-1} & S^{-1} \end{bmatrix}$ (vérifier que le produit avec $A^T A$ vaut l'identité !).

3 Algorithme et analyse de sa complexité

On considère alors l'algorithme suivant pour calculer A^{-1}.

1. Calculer A^TA (multiplication standard ou algorithme de Strassen : cf le partiel !).
2. Calculer récursivement l'inverse de A^TA en découpant la matrice en quatre sous-matrices de taille $2^{k-1} \times 2^{k-1}$ et en relançant la fonction pour calculer les inverses de B et de S. Lorsque B et S sont de taille 1×1, il suffit de prendre les inverses des deux nombres réels (on peut détecter que la matrice est non inversible si on trouve $B = 0$ ou $S = 0$). A chaque retour de récursion, utiliser la formule de la question (4) de la section précédente pour calculer la matrice inverse.
3. Calculer $A^{-1} = (A^TA)^{-1}A^T$ pour obtenir l'inverse de A, puis afficher le résultat.

Lors de l'étape (2) pour obtenir $(A^TA)^{-1}$ on calcule dans l'ordre suivant les différents éléments : B^{-1}, CB^{-1} et sa transposée $B^{-1}C^T$, S, S^{-1}, $S^{-1}CB^{-1}$ et sa transposée, et enfin $B^{-1}C^TS^{-1}CB^{-1}$ et $B^{-1} + B^{-1}C^TS^{-1}CB^{-1}$.

Donner la complexité de cet algorithme en supposant que les multiplications sont effectuées à l'aide de l'algorithme de multiplication standard (ou celui de Strassen). On fera préalablement le compte du nombre des transpositions/additions/soustractions et des multiplications pour établir une relation de récurrence entre $C(n)$ (la complexité pour une matrice de taille $n \times n$) et $C(\frac{n}{2})$.

4 Implémentation

Rappel : on suppose dans ce devoir que A est de taille $2^k \times 2^k$ avec k entier naturel.

1. Coder en C une fonction de la multiplication standard de deux matrices carrées $n \times n$. On lira les éléments des deux matrices dans deux fichiers séparés contenant leur taille n suivie de leurs n^2 éléments.
2. Implémenter l'algorithme d'inversion décrit à la section précédente.
3. Pour les courageux :
   (a) Implémenter l'algorithme de multiplication de Strassen afin d'améliorer l'algorithme d'inversion (en remplaçant la multiplication standard).
   (b) Modifier l'algorithme et le code pour gérer le cas où n n'est pas une puissance de 2 (on fournira un programme distinct du précédent).