Travaux dirigés Produit matriciel n°2

Mathématiques pour l'informatique

—IMAC 2—

▸ **Exercice 1.** ***Matrices et C++***

Donner une structure de données efficace pour traiter les matrices en C++.

▸ **Exercice 2.** ***Premiers contact avec Eigen***

1. *Télécharger Eigen.*
2. *Tester l'exemple sur la page de l'enseignant.*

▸ **Exercice 3.** ***Produit scalaire***

1. *Soient x_m un vecteur de taille m et y_n un vecteur de taille n. Quelles conditions sur m et n doivent-être satisfaites pour pouvoir effectuer le produit scalaire $x_m y_n$?*
2. *Coder en C++ le produit scalaire de 2 vecteurs en utilisant la bibliothèque* Eigen.
3. *Donner la complexité en temps du produit scalaire.*

▸ **Exercice 4.** ***Multiplication de 2 matrices***

1. *Soient A_{mk} une matrice à m lignes et k colonnes et B_{ln} une matrice à l lignes et n colonnes. Quelles conditions sur l, m, n et k doivent-être satisfaites pour pouvoir effectuer le produit $A_{mk}B_{ln}$?*
2. *Coder en C++ la fonction de la multiplication de ces 2 matrices.*
3. *Faire de même en utilisant la fonction du produit scalaire de l'exercice précédent.*
4. *Donner la complexité en temps de la multiplication de ces 2 matrices. Généraliser pour des matrices carrées $n \times n$.*

▸ **Exercice 5.** ***Multiplication de Strassen***

1. *La méthodes de multiplication de matrices de Strassen permet de multiplier des matrices en $O(n^{2,81})$ plutôt qu'en $O(n^3)$. En théorie, à partir de quel ordre la multiplication de 2 matrices carrée cette méthode est-elle 2 fois plus rapide que la méthode standard?*

2. *Coder la méthode de Strassen en C++.*

$$\begin{array}{|c|c|} \hline r & s \\ ae+bg & af+bh \\ \hline t & u \\ ce+dg & cf+dh \\ \hline \end{array} = \begin{array}{|c|c|} \hline a & b \\ \hline c & d \\ \hline \end{array} \times \begin{array}{|c|c|} \hline e & f \\ \hline g & h \\ \hline \end{array}$$

   Avec :
   $P_1 = a(f - h)$
   $P_2 = (a + b)h$
   $P_3 = (c + d)e$
   $P_4 = d(g - e)$
   $P_5 = (a + d)(e + h)$
   $P_6 = (b - d)(g + h)$
   $P_7 = (a - c)(e + f)$
   et
   $r = P_5 + P_4 - P_2 + P_6$
   $s = P_1 + P_2$
   $t = P_3 + P_4$
   $u = P_1 + P_5 - P_3 - P_7$

3. *Faire un test de performance entre :*

   - *Strassen*
   - *votre version "3 boucles `for`"*
   - *le produit matriciel de Eigen*
   - *la version multithreadée de Eigen*

   La version multithreadée de Eigen fonctionne de la façon suivante :

   - `grep -c processor /proc/cpuinfo` *pour avoir une idée du nombre de coeur dans la machine*
   - *en rajoutant* `-fopenmp` *dans la ligne de compilation*
   - *en exécutant* `OMP_NUM_THREADS=n ./my_program` *plutôt que* `./my_program` *ou bien en ajoutant dans votre code* `Eigen::setNbThreads(n);` *(où n est le nombre de threads que vous voulez lancer).*

▸ Exercice 6. *Matrice de permutation*

Soit $A_{4\times 4}$ une matrice carrée d'ordre 4.

1. *Montrer qu'une matrice de permutation est carrée.*
2. *Trouver une matrice M telle que le produit MA renvoie une matrice où les lignes 2 et 4 de A ont été permutées. Vérifier avec Eigen.*
3. *Trouver une matrice N permettant de permuter les colones 2 et 4 de A. Vérifier avec Eigen.*

▸ Exercice 7. *Transposée*

1. *Monter que $(A+B)^\top = A^\top + B^\top$.*
2. *La matrice suivante représente une flèche vers la droite en ASCII art.*
Que représente sa transposée?

$$\begin{bmatrix}
8&8&8&8&8&8&8&8&8&8&8&8&8&8&8&8&8&8&8&8\\
8&8&8&8&8&8&8&8&8&8&8&8&1&8&8&8&8&8&8&8\\
8&8&8&8&8&8&8&8&8&8&8&8&1&1&8&8&8&8&8&8\\
8&8&8&8&8&8&8&8&8&8&8&8&1&1&1&8&8&8&8&8\\
8&8&8&8&8&8&8&8&8&8&8&8&1&1&1&1&8&8&8&8\\
8&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&8&8&8\\
8&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&8&8\\
8&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&8\\
8&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&8&8\\
8&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&8&8&8\\
8&8&8&8&8&8&8&8&8&8&8&8&1&1&1&1&8&8&8&8\\
8&8&8&8&8&8&8&8&8&8&8&8&1&1&1&8&8&8&8&8\\
8&8&8&8&8&8&8&8&8&8&8&8&1&1&8&8&8&8&8&8\\
8&8&8&8&8&8&8&8&8&8&8&8&1&8&8&8&8&8&8&8\\
8&8&8&8&8&8&8&8&8&8&8&8&8&8&8&8&8&8&8&8
\end{bmatrix}$$

▸ Exercice 8. *L'oeil du tigre*

Est-ce que B est l'inverse de A? Pourquoi?

$$A = \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 0 & -3 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix}$$

▸ Exercice 9. *Trace d'une matrice*

La trace d'une matrice carrée est la somme de ses éléments diagonaux. Montrer que les propriétés suivantes sont vraies :

1. $tr(A+B) = tr(A) + tr(B)$
2. $tr(\alpha A) = \alpha tr(A)$
3. $tr(A^\top) = tr(A)$

▸ **Exercice 10.** ***Matrice antisymétrique***

1. *Montrer que la trace d'une matrice antisymétrique est nulle.*
2. *Soit une matrice carrée A d'ordre n. On défini les matrices $A_s = \frac{1}{2}(A + A^\top)$ et $A_{as} = \frac{1}{2}(A - A^\top)$. Montrer que la matrice A_s est une matrice symétrique et que la matrice A_{as} est une matrice antisymétrique.*
3. *Montrer que n'importe quelle matrice carrée A peut s'exprimer comme la somme d'une matrice symétrique A_s et d'une matrice antisymétrique A_{as}.*

▸ **Exercice 11.** ***Multiplications matricielles enchaînées***

Il s'agit ici d'un exemple de programmation dynamique appliquée aux matrices. On dispose d'une chaîne de n matrices $< A_1A_2...A_n >$ et on désire calculer le produit $M = A_1A_2...A_n$. Chaque matrice A_i à une taille compatible avec la matrice A_{i+1} pour la multiplication. On ne sait multiplier que 2 matrices à la fois et on sait que l'ordre de multiplication des matrices ne change pas le résultat (en théorie). Par exemple pour la chaîne $< A_1A_2A_3A_4 >$, il existe 5 façons de multiplier les 4 matrices :

$$(A_1(A_2(A_3A_4)))$$
$$(A_1((A_2A_3)A_4))$$
$$((A_1A_2)(A_3A_4))$$
$$((A_1(A_2A_3))A_4)$$
$$(((A_1A_2)A_3)A_4)$$

La manière de parenthéser peut avoir un impact important sur le temps de calcul du produit des matrices.

1. *soit la chaîne $< A_1A_2A_3 >$ avec :*
   - *A_1 de taille 10 $\times$ 200*
   - *A_2 de taille 200 $\times$ 5*
   - *A_3 de taille 5 $\times$ 50*

   (a) *Donner la liste de tous les parenthésages possibles pour cette chaîne.*

   (b) *Pour chaque parenthésage, donner le nombre d'opérations à effectuer. Que constate-t-on?*

2. *Pour la suite, on admet qu'il existe un parenthésage optimal pour le calcul du produit des matrices. Soit n le nombre de matrices de la chaîne et $s[i,j]$ le tableau à 2 dimensions tel que $s[i,j] = k$ soit l'indice du parenthésage optimal pour séparer la sous-chaine $< A_i ... A_j >$ en 2 sous-chaines $< A_i ... A_k > \times < A_{k+1} ... A_j >$. Donner un pseudocode permettant de calculer le produit d'une chaîne de matrices connaissant s (supposé rempli correctement).*

3. *Malheureusement, en pratique, on ne connait pas le contenu de s, et pour le construire, on a besoin de données supplémentaires. Soit p un tableau de taille $n+1$ tel que chaque matrice A_i soit de dimension $p_{i-1} \times p_i$. Soit $m[i,j]$ le nombre minimal d'opérations nécessaires pour le calcul de la matrice $A_{i...j}$ correspondant au produit des matrices $A_i A_{i+1} ... A_j$, on peut définir récursivement $m[i,j]$ de la façon suivante :*

$$m[i,j] = \begin{cases} 0 & si\ i = j \\ min_{(i \leq k < j)}\{m[i,k] + m[k+1,j] + p_{i-1}p_kp_j\} & si\ i < j \end{cases}$$

   Reste encore à trouver la valeur de k pour chaque $m[i,j]$. Si pour chaque $m[i,j]$ on regarde toutes les possibilités pour k, on obtient un algorithme de complèxité exponentielle. Nous pouvons toutefois remarquer que ce genre d'algorithme récursif peut rencontrer plusieurs fois le même sous-problème dans différentes branches de son arbre de récursivité.

   La méthode à utiliser se décompose en plusieurs étapes :

   - *initialiser tous les $m[i,i]$ à 0, les autres à $+\infty$*
   - *pour tous les sous-produits de 2 matrices, calculer les $m[i,j]$ et les $s[i,j]$.*
   - *pour tous les sous-produits de 3 matrices, calculer les $m[i,j]$ et les $s[i,j]$.*
   - *...*

   Ecrire le pseudo-code du programme calculant les tableaux $s[i,j]$ et $m[i,j]$ à partir de p. Quelle est sa complexité en temps? Quelle est sa complexité en mémoire?

4. *Dans quel cas cette méthode ne permet-elle pas de gagner du temps? Quels sont les défauts de cette méthode?*