Travaux Pratiques 4

Dans ce TP, nous allons manipuler des matrices. Avec les éléments de base de la syntaxe Python, il serait possible de représenter des vecteurs par des listes et des matrices par des listes de listes. Cependant, il est bien plus pratique et efficace d'utiliser le package NumPy. Pour cela, importez-le au début de votre code par la commande suivante.

```
import numpy as np
```

Voici quelques éléments de syntaxe utilisant NumPy.

```
a[i, j]                    # Élément d'indices (i, j) de la matrice a
a[i:j, k:l]                # Sous-matrice (de i à j-1, de k à l-1)
n, m = a.shape             # Dimensions de la matrice a
a = np.zeros((n, m))       # Matrice n*m remplie de zéros
a = np.random.rand(n, m)   # Matrice à coefs aléatoires dans [0, 1[
```

NumPy propose également des fonctions pour additionner ou multiplier des matrices. Nous les ignorerons délibérément, puisque le but de ce TP est précisément d'implanter de telles fonctions.

Exercice 1 Multiplication classique

Question 1.1 Écrivez une fonction `classic_matrix_mult` qui, étant donné deux matrices carrées a et b de même taille, renvoie leur produit (en exploitant la formule qui le définit).

Exercice 2 Algorithme de Strassen pour les matrices de taille 2^k

Le but de cet exercice est d'implanter l'algorithme de Strassen (rappelé en page suivante).

Question 2.1 Écrivez une fonction `matrix_plus` qui, étant donné deux matrices a et b de mêmes dimensions, renvoie leur somme. De même, écrivez une fonction `matrix_minus` qui renvoie la différence $a - b$.

Question 2.2 On considère à présent des matrices carrées dont la taille est une puissance de 2. Écrivez une fonction `strassen_matrix_mult` qui, étant donné deux matrices a et b, utilise l'algorithme de Strassen pour renvoyer leur produit.

Question 2.3 Sur quelques exemples, vérifiez que les fonctions `classic_matrix_mult` et `strassen_matrix_mult` renvoient le même résultat.

Exercice 3 Temps d'exécution

Question 3.1 Rappelez la complexité en pire cas des deux algorithmes ci-dessus.

Question 3.2 Sur quelques exemples, comparez leur temps d'exécution. Que remarquez-vous ? Comment l'expliquer ?

Question 3.3 (Question bonus) Comment pouvez-vous rendre plus efficace votre implémentation de l'algorithme de Strassen ? Écrivez une fonction `strassen_matrix_mult_2` qui implante cette amélioration.

Question 3.4 (Question bonus) Le package NumPy possède aussi une fonction pour calculer le produit de deux matrices : `np.dot(a, b)`. Comparez son temps d'exécution avec les fonctions précédentes. Que remarquez-vous ? Comment l'expliquer ?

Exercice 4 Généralisation aux matrices carrées de taille quelconque

Question 4.1 (Question bonus) Écrivez une fonction `strassen_matrix_mult_3` qui, étant donné deux matrices a et b carrées de même taille quelconque, renvoie leur produit. Pour cela, faites appel à la fonction `strassen_matrix_mult_2` définie précédemment.

Rappel : l'algorithme de Strassen recherche le produit $C = AB$ de deux matrices carrées A et B de taille 2^k en divisant les trois matrices A, B et C en matrices par blocs de tailles égales :

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

où X_{ij} est la sous-matrice carrée de X formée des 2^{k-1} premières (resp. dernières) lignes si $i = 1$ (resp. si $i = 2$) et des 2^{k-1} premières (resp. dernières) colonnes si $j = 1$ (resp. si $j = 2$).
La matrice C est alors déterminée en calculant :

$$\begin{aligned} P_1 &= (A_{11} + A_{22})(B_{11} + B_{22}), \\ P_2 &= (A_{21} + A_{22})(B_{11}), \\ P_3 &= (A_{11})(B_{12} - B_{22}), \\ P_4 &= (A_{22})(B_{21} - B_{11}), \\ P_5 &= (A_{11} + A_{12})(B_{22}), \\ P_6 &= (A_{21} - A_{11})(B_{11} + B_{12}), \\ P_7 &= (A_{12} - A_{22})(B_{21} + B_{22}). \end{aligned}$$

et en remarquant que :

$$\begin{aligned} C_{11} &= P_1 + P_4 - P_5 + P_7, \\ C_{12} &= P_3 + P_5, \\ C_{21} &= P_2 + P_4, \\ C_{22} &= P_1 + P_3 - P_2 + P_6. \end{aligned}$$