A. Marino/ N. Chireux
Lycée Joffre MPSI

TP 10: Le Pivot de Gauss

Le but du TP est la programmation de l'algorithme de Gauss pour l'inversion de matrices. Bien relire le cours pour la description de l'algorithme.
Dans ce TP :

- nous travaillons avec des matrices de flottants. (écrire 1. au lieu de 1)
- nous nous autorisons le transtypage int-float.
- jusqu'à indication du contraire, nous faisons fi des erreurs liées au type flottant.

Nous allons utiliser le module numpy ... bien lire le formulaire de Centrales. Par exemple A = np.array([[1,2],[3,4]]) représente la matrice $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$.
Attention au piège : les listes sont numérotées à partir de 0, pas à partir de 1 ! ! !
On rappelle aussi qu'une variable de type liste est un pointeur : une fonction peut la modifier, une copie peut se faire avec la commande **deepcopy** après import de copy.
L'affichage d'une matrice se fait avec la commande **print(A)**.

Comme vous pourrez le contacter dans le formulaire, Il existe une commande pour résoudre les systèmes linéaires alg.solve(A,b) ou inverser des matrices alg.inv(A). Vous pourrez la tester mais l'objet du TP est de programmer un algorithme d'inversion.

EXERCICE 1: Les transformations élémentaires

Dans un premier temps, on ne demande pas de gérer le mauvais emploi d'un utilisateur qui donnerait un indice incorrect (par exemple qui demanderait de permuter les lignes 1 et 4 d'une matrice à trois lignes). A la fin du sujet, vous pourrez réfléchir à ce problème (test if, commande assert, . . .). Attention à la numérotation à partir de 0 !

1. Programmer une fonction **dilat** qui prend en arguments une matrice A, un indice i et un coefficient lamda (et pas lambda, qui est réservé par Python), et qui rend la matrice dont la ligne i est multipliée par lamda. Cette fonction, comme les deux suivantes, modifiera la matrice.
2. Faire de même pour la permutation (fonction **permut**). Les arguments seront une matrice A et deux indices i et j.
3. Faire de même pour la transvection (fonction **transvect**). Les arguments seront une matrice A, deux indices i et j ($i \neq j$), et un coefficent lamda. L'opération sera : $L_i \leftarrow L_i + \lambda L_j$.

EXERCICE 2: L'algo

Attention à la numérotation à partir de 0 !
Nous nous proposons dans cette partie d'écrire l'algorithme de Gauss pour l'inversion d'une matrice. Vous devrez effectuer simultanément les opérations sur la matrice identité np.eye(n).

1. A l'aide de transvections et de dilatations inverser la matrice $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$.
2. Automatiser cette méthode pour une matrice carrée de taille n dans la fonction **gauss**. Le seul argument est la matrice A. (nous commencerons par écrire une fonction "pivot"). Comparer vos résultats avec alg.inv(A).
3. Tester votre programme sur des exemples issus du TD de Maths.

EXERCICE 3: Des erreurs d'arrondis : pivot partiel

Le choix du pivot n'est pas anodin ! Si la matrice est telle qu'un pivot est nul, notre fonction marchera mal.

1. Une première idée est de tester si le pivot est nul, et au besoin échanger deux lignes pour régler le problème. Pourquoi est-ce une mauvaise idée ?
2. Il faut en fait choisir un autre pivot non-nul. Pour minimiser les erreurs d'arrondi (inhérentes au type float), il faut choisir un pivot maximal en valeur absolue (dans une ligne non encore traitée). On aura besoin de **|x|=math.fabs(x)** avec le module math chargé. Comprendre l'intérêt de cette idée de pivot maximal.
3. Fabriquer une fonction **pivotmax** qui prend en arguments une matrice A, et un indice i, et qui retourne l'indice de ligne (pas la valeur) du pivot maximal (en valeur absolue) de la colonne i sur les lignes i et suivantes.
4. Modifier la fonction **gauss** en une fonction **gauss1** optimisée pour que le choix du pivot vérifie ce critère.

5. Constater l'intérêt de ces modifications sur l'exemple suivant : (tester en premier Gauss, Gauss1 puis alg.inv(A))

$$A = \begin{pmatrix} 10^{-8} & 0 & 1 \\ 3 & 4 & 1 \\ 10^{8} & 2 & 1 \end{pmatrix}$$

Regardez le coefficient $(0, 0)$ de la matrice inverse. Notez que l'exemple n'induit qu'un seul choix de pivot qui pose problème. Avec trois choix, les erreurs seraient encore plus multipliées. Vous pouvez le constater en modifiant les valeurs des coefficients (prendre des puissances de 10 pour bien le voir).

EXERCICE 4: Compléxité de la méthode

Evaluer le nombre d'opérations par la méthode pivot partiel.
On désignera par "opération élémentaire" chaque comparaison, affectation ou opération arithmétique sur les flottants.

1. Montrer que le coût de la mise sous forme triangulaire est en $O(n^3)$ où n^2 est la taille de la matrice.
2. Montrer que le coût de la phase de remontée de en $O(n^2)$.
3. Montrer que le coût du pivot partiel est en $O(n^3)$.

Un peu de Culture : peut-on faire mieux que n^3 dans le cas général ?

La question est la suivante : quel est le coût d'inversion d'une matrice ?

1. Une approche théorique :
   Pour appuyer ce commentaire, on peut citer un résultat remarquable :

   Théorème :
   En notant $M(n)$ la complexité (dans le pire des cas) du calcul du produit matriciel de deux matrices de taille $n \times n$ et sous l'hypothèse $n^2 = 0(M(n))$ alors on peut inverser une matrice de taille $n \times n$ en $O(M(n))$.

   Une multiplication naïve requiert de l'ordre de n^3 opérations élémentaires. Une amélioration (via un "diviser pour régner") permet de descendre cette complexité à n^α avec $\alpha = \ln_2(7) \sim 2,69$ (Strassen 1972)
   La question "peut-on, pour tout $\alpha > 2$ trouver un algorithme de multiplication de complexité en $O(n^\alpha)$? " est à ce jour encore ouverte.
2. Une approche pratique : nous pourrions évidemment améliorer les méthodes en considérant les caractéristiques d'une matrice. (par ex : elle contient beaucoup de 0)