Calcul matriciel et pivot de Gauss

Motivation : Le but de la première partie du T.P (partie 1 et 2) est de manipuler les matrices pour se familiariser avec elles. Dans un second temps, vous allez implémenter les algorithmes de résolutions de systèmes linéaires et d'opérations élémentaires sur les matrices (vraisemblablement chez vous) dans les parties 5 et 6. Le but des parties (théoriques) 3 et 4 de ces notes est de présenter ces différents algorithmes, de voir comment ils s'articulent entre eux, de préciser quelques notions et outils qui pourront vous aider, et d'autre part d'analyser leur complexité.

0 Rappels sur les matrices en PYTHON pur :

a) Une matrice n'est qu'une liste de listes

Par exemple `A=[[ 1,2],[3,4]]` représentera pour nous la matrice $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$.

Les entrées de `A` sont obtenues via `A[i][j]`.

Noter que l'affichage d'une matrice n'est pas bien joli, mais qu'il est facile de programmer une petite fonction qui fasse un affichage joli.

Attention : En python, les indices commencent à 0, donc pour les matrices à `A[0][0]`.

b) Une liste est un objet mutable

On peut modifier les entrées de la matrice, c'est bien, les fonctions peuvent transformer la matrice qu'elles ont en argument, c'est bien, mais il faut prendre garde aussi aux problèmes des éventuelles copies d'une matrices qui pointeraient vers la même case mémoire... une modification de la copie modifiera aussi l'original.

On rappelle qu'il existe un module appelé `copy` qui résout ce problème des copies.

1 Opérations sur les matrices

À vous :

Exercice 1.

a) Écrire une fonction **matrice_nulle** prenant en entrée deux entier (n, m) et renvoyant la matrice nulle de taille $n \times m$.

b) Écrire une fonction **dimension** prenant en entrée une matrice et renvoyant le nombre de lignes et le nombre de colonnes.

c) Écrire une fonction **addition** prenant en entrée deux matrices (de même taille !) et renvoyant leur somme.

d) Écrire une fonction **transposee** prenant en entrée une matrice et renvoyant sa transposée.

e) Écrire une fonction **multiple** prenant en entrée une matrice A et un scalaire λ et renvoyant la matrice λA.

f) Écrire une fonction **multiplication** prenant en entrée deux matrices de tailles compatibles ((n, p) et (p, q) par exemple) et renvoyant leur produit.

g) Écrire une fonction **puissance** prenant en entrée une matrice A et un entier $n \geq 0$ et renvoyant la matrice A^n.

Exercice 2.
Déterminer la complexité des opérations **addition**, **transposee**, **multiple**, **multiplication** et **puissance**

Exercice 3.
Écrire une fonction **est_symetrique** prenant en entrée une matrice, testant le caractère symétrique de la matrice, et renvoyant un booléen.

2 Matrice magique

On travaille ici avec des entiers naturels. Une matrice carrée est dite « magique » lorsque la somme des éléments de chaque ligne, de chaque colonne, celles des éléments de la diagonale et de l ?antidiagonale sont toutes égales. Par exemple, la matrice

$$\begin{pmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{pmatrix}$$

est magique.

Exercice 4.

a) Écrire une fonction **total_ligne**(M,k) qui calcule la somme des éléments de la ligne k de la matrice M.

b) Écrire une fonction **total_colonne**(M,k) qui calcule la somme des éléments de la colonne k de la matrice M.

c) Écrire une fonction **total_diag**(M) qui calcule la somme des éléments de la diagonale de la matrice M.

d) Écrire une fonction **total_antidiag**(M) qui calcule la somme des éléments de l'antidiagonale de la matrice M.

e) Écrire une fonction **magique** prenant en entrée une matrice, testant le caractère magique de la matrice et renvoyant un booléen.

Exercice 5.
Il existe un algorithme permettant de construire des matrices magiques de taille impaire. Soit N un entier impair. On remplit la matrice $N \times N$ avec les entiers de 1 à N^2 dans cet ordre :

- On commence par la case du milieu de la première ligne.
- Quand une case est remplie, la suivante à remplir est celle située à la ligne précédente et à la colonne suivante (modulo la taille de la matrice !) si elle est libre.
- Si cette case est déjà remplie, on choisit la case qui est dans la même colonne que la dernière remplie mais à la ligne suivante.

Écrire une fonction **carremagique** prenant en entrée un entier n et renvoyant une matrice magique de taille $2n + 1$.

3 Résolution des systèmes triangulaires :

On considère un système $Y = TX$ avec $T \in TS_n(\mathbb{K})$ inversible et $Y \in M_{n,1}(\mathbb{K})$ fixés, et on cherche l'unique solution $X \in M_{n,1}(\mathbb{K})$ de ce système.

Un tel système se résout *de bas en haut.*

A la ligne L_n, on va seulement faire un quotient pour trouver $x_n = y_n/t_{n,n}$.

Mais ensuite à la ligne L_i, on obtiendra x_i par : $x_i = \dfrac{1}{t_{i,i}}(y_i - \sum_{j>i} t_{i,j}x_j)$.

Question : Donner l'ordre de grandeur de la complexité de cet algorithme de résolution :

$$\Theta(n), \Theta(n^2), \Theta(n^3) \, ?$$

4 Pivot : mise sous forme triangulaire d'une matrice

On fixe une matrice $A \in M_n(\mathbb{K})$ *inversible*.

4.1 La méthode du pivot, avec une spécificité pour les machines

Le premier pivot

Au début de la méthode du pivot, on veut « nettoyer » la première colonne en utilisant $A(1,1)$ comme pivot, via $L_i \leftarrow L_i - \mu_i L_1$ pour $i \geq 2$ où $\mu_i = A(i,1)/A(1,1)$. Mais le (seul) problème est que $A(1,1)$ peut être nul.

Ce que dirait un mathématicien :

On doit donc d'abord rechercher la première entrée non nulle dans la colonne 1 et une fois cette entrée trouvée à une ligne L_i échanger L_i et L_1. Comme A est inversible, on sait qu'on va toujours en trouver, car la première colonne ne peut pas être nulle.

Une précaution liée au calcul numérique pour le choix du pivot :

• Quand on manipule des flottants, le test de *l'égalité* à zéro n'est pas adapté, car parfois dans les calculs un nombre qui vaudrait théoriquement zéro sera remplacé par un nombre très petit. On pourrait remplacer le test d'égalité à zéro par la comparaison avec le epsilon machine, mais :

• pour des raisons de *meilleure précision* du calcul numérique,

il vaut mieux éviter de diviser par des très petits nombres.

Corollaire du principe précédent, ce que dit un mathématicien numérique :

il vaut mieux choisir comme pivot dans la colonne 1, *l'entrée* ayant la plus grande valeur absolue

Le pivot à l'étape i :

A l'étape i, en suivant le principe précédent, on choisit *parmi les lignes L_j avec $j \geq i$*, celle où l'entrée $A(j,i)$ a la plus grande valeur absolue, et on l'échange avec L_i. Ensuite on se sert de cette nouvelle L_i pour nettoyer la colonne C_i en dessous de la diagonale.

Remarque du mathématicien : là encore, on est sûr qu'une des entrées $A(j,i)$ avec $j \geq i$ est non nulle. Sinon, comme à ce stade les premières colonnes $C_1, \ldots, C_{i-1}$ sont déjà celles d'une matrice T.S., on aurait $C_1, \ldots, C_i$ liées.

4.2 Estimation de la complexité de la méthode du pivot précédente

Écrite de manière plus formelle, la description de l'algorithme du pivot donnée au paragraphe précédent devient :

On numérote ici les lignes en $L_0, \ldots, L_{n-1}$ dans l'esprit de PYTHON.

```
pour i de 0 à  n-2 :
        trouver j >=i  tel que |A(j,i)| soit maximum
        échanger L_i et L_j
        pour k de i+1 à  n
             L_k <- L_k - mu_k L_i
```

Pour chaque valeur de $i \in [\![0, n-2]\!]$:
- la recherche de j coûte $n-i$ comparaisons,
- l'échange éventuel de L_i et L_j coûte $2n+2$ affectations ($2n$ pour les entrées des lignes, et 2 pour i et j)
- pour chaque valeur de k entre $i+1$ et $n-1$, la transvection coûte n affectations, et autant de divisions, multiplications, soustractions (une par entrée de la ligne).

Propriété En ajoutant toutes ces contributions comme comptant 1, on obtient pour la méthode du pivot sur une matrice $A \in GL_n(\mathbb{K})$ un coût en $\Theta(n^3)$.

5 Application à la résolution de systèmes de Cramer

5.1 Codage matriciel d'un système *quelconque* avec second membre :

Remarque : Si on se donne un système linéaire quelconque S : $Y = AX$ avec $A \in M_{m,n}(\mathbb{K})$, $Y \in M_{m,1}(\mathbb{K})$ et $X \in M_{n,1}(\mathbb{K})$, où A et Y sont fixés et on cherche X, si on transforme ce système S par des opérations élémentaires sur les lignes, on arrive à un système équivalent S' : $Y' = A'X$ où Y' et A' ont été modifiées par les *mêmes* opérations opérations sur les lignes.

Ces systèmes équivalents ont bien sûr même ensemble de solution. On en déduit la :

Propriété-définition de la matrice augmentée associée à un système :

Toutes les opérations sur un système peuvent être codées sur ce qu'on appelle la *matrice augmentée* de ce système à savoir la matrice $(A|Y)$ obtenue en rajoutant Y comme dernière colonne à A. Les opérations faites sur lignes du système sont faites sur les lignes de cette matrice. On arrive à une matrice $(A'|Y')$ telle qu'on ait toujours l'équivalence : $AX = Y \Leftrightarrow A'X = Y'$.

Exemple : On considère le système

$$\begin{cases} x + y + 7z &= -1 \\ 2x - y + 5z &= -5 \\ -x - 3y - 9z &= -5 \end{cases}$$

A ce système, on associe la matrice augmentée :

$$\begin{pmatrix} 1 & 1 & 7 & -1 \\ 2 & -1 & 5 & -5 \\ -1 & -3 & -9 & -5 \end{pmatrix}$$

Comparons alors l'effet d'une opération élémentaire sur le système et la matrice augmentée. Par exemple, avec : $L_2 \leftarrow L_2 - 2L_1$

- le système devient :

$$\begin{cases} x + y + 7z &= -1 \\ -3y - 9z &= -3 \\ -x - 3y - 9z &= -5 \end{cases}$$

- la matrice augmentée devient :

$$\begin{pmatrix} 1 & 1 & 7 & -1 \\ 0 & -3 & -9 & -3 \\ -1 & -3 & -9 & -5 \end{pmatrix}$$

Ceci illustre bien qu'il suffit de conduire les opérations élémentaires sur la matrice augmentée pour coder les différents systèmes équivalents obtenus par opérations élémentaires.

Ici, en poursuivant le pivot avec la matrice augmentée : $L_3 \leftarrow L_3 + L_1$

$$\begin{pmatrix} 1 & 1 & 7 & -1 \\ 0 & -3 & -9 & -3 \\ 0 & -2 & -2 & -6 \end{pmatrix}$$

$L_2 \leftarrow \frac{-1}{3} L_2$

$$\begin{pmatrix} 1 & 1 & 7 & -1 \\ 0 & 1 & 3 & 1 \\ 0 & -2 & -2 & -6 \end{pmatrix}$$

$L_3 \leftarrow L_3 + 2L_2$

$$\begin{pmatrix} 1 & 1 & 7 & -1 \\ 0 & 1 & 3 & 1 \\ 0 & 0 & 4 & -4 \end{pmatrix}$$

Cette matrice augmentée *dont la première partie est une matrice TS* 3×3 code le système triangulaire équivalent à notre système initial :

$$\begin{cases} x + y + 7z &= -1 \\ y + 3z &= 1 \\ 4z &= -4 \end{cases}$$

5.2 Les deux étapes résolution d'un système de Cramer par le pivot

Définition : Un système de Cramer est un système $AX = Y$ avec A matrice carrée inversible données, Y colonne donnée, et d'inconnue une colonne X. On sait donc que ce système admet un unique solution.

La première étape de la résolution :

Transformer A en une matrice A' triangulaire sup. avec la méthode du pivot sur les lignes donnée au § **??**. Appliquer les mêmes transformations sur les lignes à Y ce qui donne une nouvelle colonne Y'. (On peut utiliser la matrice augmentée pour cela).

La deuxième étape de la résolution :

Résoudre le système triangulaire $A'X = Y'$ avec la méthode du § **??**.

Question : Quelle est la complexité totale de cet algorithme ?

6 Résolution d'un système triangulaire

Écrire une fonction PYTHON appelée `resout_triangle_sup` qui prend comme arguments une matrice triangulaire supérieure inversible $T \in TS_n(\mathbb{K})$ et une matrice $Y \in M_{n,1}(\mathbb{K})$ et qui renvoie l'unique solution $X \in M_{n,1}(\mathbb{K})$ du système $Y = TX$. (Ici les nombres dans $\mathbb{K}$ seront des flottants).
Outils utiles :

• Par simplicité, on codera `Y` et `X` comme des listes simples. On peut initialiser `X` à la bonne taille via `X=[0]*n`. On peut aussi créer `X` petit à petit avec des `append` mais attention que dans ce cas, pour le système T.S., on obtiendra `X` à l'envers (d'abord `X[n-1]`).

• Comme un tel système se résout de bas en haut, il est bon de rappeler qu'en PYTHON on peut faire un `range` dans l'ordre décroissant : attention à la façon dont les bornes sont comprises.
Essayer par exemple ce que fait un `range(4,-2,-1)`.

7 Les opérations du pivot

Les programmes suivants seront écrits pour s'appliquer même à une matrice rectangulaire.

a) Ecrire une fonction `echange_ligne(A,i,j)` qui échange les lignes `i` et `j` de la matrice `A`,

b) Ecrire une fonction `transvection_ligne(A,i,j,mu)` qui dans la matrice `A` fait l'opération $L_i \leftarrow L_i + \mu L_j$.

On tient compte du fait que les listes sont des objets mutables :

Les fonctions précédentes modifieront la matrice `A` passée en argument.

Remarque : *Pourquoi ai-je appelé la constante* `mu` *et pas* `lambda` *? Parce qu'en* PYTHON *le nom* `lambda` *est réservé pour la déclaration de fonction en une ligne.*

8 Pivot sur les lignes transformant une matrice en une matrice triangulaire

Travail à faire :

écrire une fonction `devient_triangle` qui prend comme argument une matrice `A` *qui ne sera pas modifiée par la fonction* et qui renvoie une matrice `T` triangulaire supérieure obtenue par pivot sur les lignes à partir de `A`.

Outils utiles :

• On pourra créer d'abord une petite fonction `cherche_pivot` auxiliaire, qui recherche la plus grande entrée non nulle en dessous de la diagonale dans une colonne `j`.

• La fonction `deepcopy` du module `copy` permet de créer des copies *autonomes* d'une liste.

9 Application aux système de Cramer : $Y = AX$ avec A inversible

a) Première étape : à l'aide de ce qui précède, en modifiant légèrement la fonction `devient triangle` précédente, écrire une fonction qui ramène un système `Y=AX` à un système triangulaire.

Pour cela, il suffit de faire subir à `Y` les mêmes opérations que celles faites sur `A`.

Ceci se formalise aussi avec la notion de *matrice augmentée :* $[A|Y]$.

b) Deuxième étape : écrire une fonction `resout_systeme` qui prend comme argument une matrice inversible `A` et une colonne `Y` et renvoie l'unique `X` tel que `Y=AX`.