Université Polytechnique HAUTS-DE-FRANCE

INSA

Année 2023 - 2024 SHPI 2

Méthodes numériques appliquées

TP 2

En plus du dépôt habituel dans `Moodle`*,* ***chaque membre du binôme*** *devra sauvegarder le travail commun en fin de séance dans ses documents personnels (sur clé, dans son ENT ou sur le réseau) pour pouvoir le ré-utiliser au TP 3 même si l'autre membre du binôme est absent.*

— Préliminaire —

Cette partie n'est pas seulement liée au thème du TP, mais vous la traiterez dans le même carnet `Jupyter` que la suite. Il s'agit de vous sensibiliser à des précautions à prendre quand on utilise des tableaux (ou des listes), à une ou deux dimensions, avec `Python`, mais aussi, en partie, sa bibliothèque `numpy`. On suppose l'avoir importée par : `import numpy as np`

1. Exécutez le code suivant, commentez.
```
L1 = np.array([1,2,3,4]) ; L2 = L1
L1[1] = 0 ; print(L2)
```

2. Même question avec :
```
M1 = [[0]*4]*4 ; print(M1)
M1[1][2] = 5 ; print(M1)
```

3. Même question en remplaçant la première instruction par `M1 = np.array([[0]*4]*4)`

4. Même question avec :
```
A = np.array([[1,2,3],[1,4,9],[1,8,27]])
AA = A ; AA[1,1] = -2 ; A
```

5. Même question en remplaçant `AA = A` par `AA = A.copy()`

— Fin du préliminaire —

. . . mais dans la suite, pensez à ce que vous venez d'apprendre.

Le thème des séances 2 et 3 de TP est l'algorithme du pivot de Gauss, pour la résolution des systèmes linéaires de la forme $AX = B$. Dans un premier temps, on considérera que A est une matrice carrée (n, n) inversible.

On appelle « matrice augmentée » et on note $A|B$ la matrice A à laquelle on a ajouté le second membre B sur une colonne à droite, donc de dimension $n \times (n+1)$. Ainsi, si par exemple

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 4 & 9 \\ 1 & 8 & 27 \end{pmatrix} \qquad \text{et} \qquad B = \begin{pmatrix} 5 \\ 15 \\ 47 \end{pmatrix}$$

alors

$$A|B = \begin{pmatrix} 1 & 2 & 3 & 5 \\ 1 & 4 & 9 & 15 \\ 1 & 8 & 27 & 47 \end{pmatrix}.$$

Si vous avez besoin du codage `markdown` des matrices, voyez le carnet *Modele* dans `Moodle`.

Voici un rappel du principe de l'algorithme de Gauss : en posant $A|B = (a_{i,j})_{i=1\cdots n, j=1\cdots n+1}$ et en notant L_i la i-ème ligne de $A|B$, le nombre k ci-dessous étant l'indice de colonne,
Pour $k = 1, \cdots, n-1$, faire

- trouver le plus petit $i \geqslant k$ tel que $a_{i,k} \neq 0$ (on cherche un pivot non nul)
- échanger L_i et L_k
- pour $l = k+1, \cdots, n$, faire $\{$remplacer L_l par $L_l - \dfrac{a_{l,k}}{a_{k,k}} L_k$ $\}$ (on fait apparaître des 0 dans la k-ème colonne, sous le pivot)

On obtient ainsi une matrice triangulaire supérieure. On résout enfin le système « en remontant ».

Remarques :

- *il y a plusieurs façons de présenter l'algorithme de Gauss. Si vous en aviez une autre et qu'elle fonctionne, gardez-la.*
- *attention, en pratique, les cœfficients des matrices sont presque toujours des flottants et non des entiers.*

La structure des données est imposée : le type `array` du module `numpy` (module qu'il faudra donc importer). Le document *Intro_numpy* dans `Moodle` vous donne les commandes principales.

Une bonne idée serait de commencer par écrire deux fonctions qui, avec en paramètre une matrice A, retournent le nombre de lignes, respectivement de colonnes, de A : ça servira souvent... si vous pensez à les utiliser.

Dans votre programme global pour l'algorithme de Gauss, la matrice A et le second membre B doivent être en paramètres *séparément*, pour qu'on puisse par exemple fixer A et tester plusieurs valeurs de B. Mais le principe décrit ci-dessus est de travailler avec la matrice augmentée (cela revient, si on pense en termes d'équations plutôt que de matrices, à tenir compte du second membre) plutôt qu'avec A et B séparément. Le mieux est donc de commencer par créer une fonction qui les assemble.

Bien sûr il faudra tester votre programme : voir s'il donne une solution **et** si cette solution est correcte. Pas sur des cas trop simples ($A = \mathrm{Id}$!), mais pensez à des systèmes faciles à résoudre à la main (et même de tête), par exemple avec

$$A = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Votre objectif pour cette séance sera d'obtenir le meilleur programme possible. « Le meilleur » signifie qu'il doit d'abord, *par ordre de priorité décroissante* :

- respecter les pratiques de programmation conseillées, et rappelées dans `Moodle` ; vous êtes invités à les relire en début de séance (et après si nécessaire). Le sujet de ce TP étant complexe, la *méthode descendante* est recommandée : décomposer votre problème en plusieurs sous-problèmes, écrire séparément une fonction qui résout chaque sous-problème *et la tester* avant de passer au sous-problème suivant.
- résoudre les systèmes linéaires dans le cas le plus simple : matrice carrée inversible et pas de problème numérique.
- pour une utilisation ultérieure, les matrices passées en paramètre de votre programme ne doivent pas être modifiées à sa sortie.

Une fois ces conditions minimales remplies, vous essaierez d'améliorer votre programme. D'abord pour pallier les problèmes numériques classiques, par exemple la résolution de $\mathrm{AX} = B$ avec

$$A = \begin{pmatrix} \varepsilon & 1 \\ 1 & 1 \end{pmatrix} \quad \text{et} \quad B = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \quad \varepsilon = 10^{-20}$$

puis de même avec

$$A = \begin{pmatrix} \varepsilon & 1 \\ \varepsilon & \varepsilon \end{pmatrix} \quad \text{et} \quad B = \begin{pmatrix} 1 \\ 2\varepsilon \end{pmatrix}$$

et bien sûr les problèmes analogues : il ne s'agit pas de concevoir un programme qui ne marcherait que sur ces deux exemples. Il y a deux niveaux de perfectionnement, et vous êtes invités à réfléchir au coût induit par ces perfectionnements.

S'il vous reste du temps, la dernière étape sera de généraliser votre programme au cas d'un système non carré. Attention, il est indispensable ici de reprendre la méthode descendante... au début, car il faudra créer de nouvelles fonctions.