Feuille de TP Python 08

Calcul matriciel et algorithme de GAUSS

1 Introduction

L'objet de ce TP est d'une part de fixer les techniques de calcul matriciel pour une utilisation minimale de Python/Numpy de type calculatrice et d'autre part de travailler l'aspect algorithmique de l'algorithme de GAUSS de résolution de systèmes linéaires qui est au programme.

2 Tableaux numpy : construction, indexation et opérations

2.1 Travail demandé

Utiliser un fichier nommé `td08.py`.

Exercice 1.—On pose, pour $\theta \in \mathbb{R}$, la matrice $M(\theta) = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}$. Nous avons vu dans les exercices du cours de maths que

- $\forall \theta, \theta' \in \mathbb{R}, M(\theta).M(\theta') = M(\theta + \theta')$
- Chaque matrice $M(\theta)$ est diagonalisable dans $\mathcal{M}_2(\mathbb{C})$, son spectre est $\text{Spec}(M(\theta)) = \{e^{\varepsilon.i\theta}, \varepsilon \in \{-1, +1\}\}$.
- Une matrice diagonalisant $M(\theta)$ est $P = \begin{pmatrix} 1 & 1 \\ i & -i \end{pmatrix}$

Vérifier informatiquement ces faits :

- définir une fonction `M` de signature `M(theta : float) -> ndarray (2,2)`
- Pour deux angles $\theta = \text{theta}$, $\theta' = \text{thetap}$ variant indépendamment dans la gamme `np.linspace(0, 2*np.pi, 20)`, vérifier¹ que $M(\theta).M(\theta') = M(\theta + \theta')$.
- Vérifier sur le même modèle la formule de changement de base $P^{-1}.M(\theta).P$ est diagonale (avec les éléments propres bien connus). On rappelle qu'un nombre « flottant » complexe imaginaire pur s'écrit à l'aide du symbole `j` accolé à flottant réel. P.ex. `i` se code `1.0j`, `3 - 5/2i`, `3 - 2.5j`.
- Tester les fonctions d'inversion `np.linalg.inv` et de diagonalisation `np.linalg.eig` sur ces matrices.

Exercice 2.—On considère pour $n \in \mathbb{N}, \geq 2$, la matrice $n \times n$ (dite du « Laplacien ») :

$$\Delta_n = \begin{pmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & \ddots & \ddots & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & -1 & 2 & -1 \\ 0 & \dots & 0 & -1 & 2 \end{pmatrix}$$

La diagonale comporte des 2, les sur- et sous-diagonales des -1 et le reste des entrées sont des 0.

On admet (c'est un calcul pas très facile mais abordable) que chacun des vecteurs v_j , pour j variant de 1 à n est vecteur propre de Δ_n où après avoir défini $\theta_j = j \frac{\pi}{n+1}$, on a posé, pour $i \in \{1, \dots, n\}$

$$(v_j)_i = \sin(i.\theta_j)$$

On pose V_n la matrice $n \times n$ dont la colonne numéro j est v_j :

$$V_n = (v_1 | v_2 | \dots | v_n)$$

- Construire une fonction `Delta(n)` qui pour une valeur de n retourne la matrice Δ_n .
- Construire une fonction `V(n)` qui pour une valeur de n retourne la matrice V_n .
- Vérifier que $V_n^{-1}.\Delta_n.V_n$ est diagonale (pour les valeurs de n variant entre 3 et 15. On rappelle que `np.linalg.inv` inverse une matrice.

1. Comment vérifier que deux matrice de `float` sont égales.

3 Implémentation de l'algorithme de GAUSS : Cas CRAMER et autres

L'archive jointe au TP contient le « moule » – à compléter– d'un script nommé `GaussCramer.py` qui implémente une version maison et simplifiée de l'algorithme de GAUSS.

On s'intéresse à l'implémentation informatique d'une version simplifiée de l'algorithme de GAUSS qui ne fonctionnera complètement que dans le cas où la matrice A du système est inversible (et donc comporte n lignes et $p = n$ colonnes). On va donc développer un ensemble de fonctions élémentaires dont l'assemblage dans la fonction `GaussCramer` du script donnera la version simplifiée voulue de l'algorithme de GAUSS.

3.1 Travail demandé

Utiliser le script à remplir `GaussCramer.py`. On pourra, à chaque étape du travail, exécuter le programme et consulter les valeurs calculées pour les exemples.

3.1.1 GAUSS : Echelonnement, la première partie de l'algorithme

Lire la `docstring` de la fonction `GaussCramer` puis observer le texte de cette fonction et repérer (mettre des commentaires descriptifs au dessus de chaque ligne concernée dans le script) :

1. Les lignes où le nombre de lignes, de colonnes de A , de B sont récupérées et assignées à des variables. Quel est le sens des `if:/return None` situés juste après ces assignations? Compléter les `print("Erreur : "???? )` qui précèdent chaque instruction `return None` pour préciser le type d'erreur à l'utilisateur.
2. La ligne où la matrice complète du système est composée, c'est à dire une matrice $(A|B)$ obtenue en juxtaposant la matrice du système avec la matrice représentant le second membre.
3. L'imbrication des deux boucles à l'issue desquelles le système devient échelonné. Quel est le sens du `if:/return None` situé à l'issue de l'échange de lignes? Compléter le `print("Erreur : "???? )` précédent l'instruction `return None` pour préciser le type d'erreur à l'utilisateur.

Pour cette dernière question, on pourra lire à profit les `docstrings` des fonctions `MultiplicationLigneScalaire`, `CombinaisonLigneScalaire`, qui ne sont pas encore écrites (cf. questions suivantes) et celle de la fonction `EchangeLignes`.

3.1.2 GAUSS : Multiplication, Combinaison de deux lignes et échange

1. Ecrire les corps des fonctions `MultiplicationLigneScalaire`, `CombinaisonLigneScalaire`, qui ne sont pas encore écrites (d'où leur manque de corps et leur terminaison sur un simple `return None`) pour qu'elles fassent ce qu'elles annoncent.
On utilisera des `.copy()` comme dans la fonction `EchangeLignes`.
2. Que vaut la matrice C à l'issue de l'étape d'échelonnement lorsque `GaussCramer` est appelée avec les différentes paires matrice A /second membre B présentes en exemples?

3.1.3 GAUSS-JORDAN : résolution –en tableau– d'un système de CRAMER échelonné

Le texte de la fonction `GaussCramer` se termine par deux blocs (signalés par les commentaires `#Bloc 1/2: Début/Fin`)
On suppose qu'à l'issue de l'étape d'échelonnement on a obtenu :

$$C = \begin{pmatrix} 1 & 1 & 1 & 2 & -2 \\ 0 & 2 & 4 & -4 & 6 \\ 0 & 0 & 3 & 9 & -6 \end{pmatrix}$$

1. Lire le `#Bloc 1` et donner la matrice C obtenue à l'issue de l'application des instructions de ce bloc. On vous demande donc de pratiquer « manuellement » les instructions décrites dans ce bloc sur la matrice C .
2. Lire le `#Bloc 2` et donner les matrices C obtenues à l'issue de chaque tour de boucle.
3. Quels sont, sous forme « classique² », les systèmes finaux obtenus à l'issue du `#Bloc 2`, décrits par la matrice C .
4. En s'inspirant de ce qui se passe pour l'exemple décrivez ce qui se passe dans le cas général. Modifier le `return C` final pour qu'il retourne ce qui est décrit en `docstring` de `GaussCramer`.

2. par cela, on entend système sous la forme

$$\begin{cases} a_{11}x + a_{12}y + a_{13}z = b_1 \\ ? a_{21}x + a_{22}y + a_{23}z = b_2 \\ a_{31}x + a_{32}y + a_{33}z = b_3 \end{cases}$$

3.1.4 GAUSS : Recherche de pivot optimal

D'un point de vue calcul numérique, on a intérêt, lors de la première étape d'échelonnement à choisir un pivot (sur la bonne colonne) sur les lignes à partir de la ligne courante qui soit *le plus grand possible* en valeur absolue.

- Remplacer, entre les lignes commentées `#Choix/Pivot`: échange de ligne Début/Fin la recherche du premier pivot non nul par une recherche de ce « meilleur » pivot. On doit aboutir à l'échange de la ligne courante avec la ligne comportant le meilleur pivot.

3.2 Inversion de matrice carrée : la méthode « miroir »

Soit A une matrice carrée de taille $n \times n$. En prenant $B = I_n$, la fonction `GaussCramer` retourne l'inverse de A . Pourquoi ?

- Ecrire une fonction « enveloppante » (un *wrapper*) de `GaussCramer`, nommée `GaussInverse` prenant en argument une matrice carrée A et retournant A^{-1} si A est inversible, `None` sinon.
- Utiliser et comparer le résultat de cette fonction et celui de la fonction `np.linalg.inv`
- Prendre une matrice M « au hasard » comportant n lignes, p colonnes. Calculer $M^T.M$ et l'inverse de celle-ci. Proposer une méthode de résolution de systèmes linéaires comportant plus d'équations que d'inconnues.

3.3 GAUSS : Système et échelonné ou pas ?

Echelonnée vs. triangulaire

La définition d'une matrice (ou d'un système) échelonnée est subtile. Par exemple, il ne faut pas absolument pas confondre matrice échelonnée et matrice triangulaire supérieure.

- Une matrice est « triangulaire supérieure » si les éléments au dessous de la diagonale (on se limite souvent à des matrices carrées) sont nuls. C'est à dire, si T est de taille $n \times p$,

$$\forall (i, j) \in \{1, \dots, n\} \times \{1, \dots, p\}, i > j \Rightarrow [T]_{ij} = 0.$$

- Pour « matrice échelonnée », une définition algorithmique est la suivante : Etant donnée une matrice A comportant n lignes et p colonnes,

- pour chaque ligne i de la matrice A , on repère l'indice de colonne de la première entrée non nulle sur la ligne i , on note cet indice j_i . Si toute la ligne est nulle, on convient que cet indice vaut $p + 1 + i$ (hors de la gamme d'indices de colonnes donc, les indices dans ce cas sont tous distincts par construction)

- La matrice A est échelonnée si et seulement si la suite finie (j_i) est strictement croissante, ce qui se teste aisément

En complément :

- Les indices des inconnues primaires sont les j_i tels que j_i est un numéro de colonne valide,
- les autres indices de colonnes sont ceux des inconnues secondaires
- les indices i tels que j_i n'est pas un numéro de colonne valide sont les lignes du système échelonné que l'on désigne communément par « équations de compatibilité »

- Les matrices $T = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$ et $S = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}$ sont triangulaires supérieures. Elles ne sont pas échelonnées.

Travail demandé

Ecrire le code manquant de la fonction `EstEchelonnee` dont l'entête est

```
EstEchelonnee(A, primaires = [], secondaires = [], compatibilites = [])
```

Elle doit retourner `True` si la matrice A échelonnée et `False` sinon, et fournir, en modifiant *en place* des listes passées sous forme de paramètres nommés avec valeur par défaut les numéros des inconnues primaires, des inconnues secondaires et des lignes de compatibilité.

4 Finir le job (Hors-programme) : Un algorithme de GAUSS complet

Le fichier `corrections/Gauss.py` contient la définition de la fonction `Gauss` résolvant le système linéaire $A.X = B$ dans tous les cas possibles.

- Lire ce code. Comprendre les différences avec `GaussCramer`.
- Comprendre l'utilisation de cette fonction dans la fonction `BaseKer` (qui en est un *wrapper*)
- (Ouverte, sans correction) Ecrire, en extrayant le code adéquat de la fonction `Gauss`, une fonction `EcheLonne(A, B)` qui retourne le couple de matrices E, D , tel que $E.X = D$ est un système échelonné équivalent à $A.X = B$. (partie échelonnement de l'algorithme de GAUSS)
- (Ouverte, sans correction) Comment déterminer une base de l'image d'une matrice A ?

PYTHON AGRO-VETO 2020

Listes

`[]` ----- Créer une liste vide
`[a]*n` ----- Créer une liste avec n fois l'élément a
`L.append(a)` Ajoute l'élément a à la fin de la liste L
`L1 + L2` ----- Concatène les deux listes $L1$ et $L2$
`len(L)` ----- Renvoie le nombre d'éléments de la liste L
`L.pop(k)` --- Renvoie l'élément d'indice k de L et l'enlève de L
`L.remove(a)` Enlève une fois la valeur a de la liste L
`max(L)` ----- Renvoie le plus grand élément de la liste L
`min(L)` ----- Renvoie le plus petit élément de la liste L
`sum(L)` ----- Renvoie la somme de tous les éléments de la liste L

Numpy

`import numpy as np`
`np.array()` ----- Transforme une liste en matrice numpy
`np.linspace(a,b,n)` ----- Crée une matrice ligne de n valeurs uniformément réparties entre a et b (inclus)
`np.zeros([n,m])` ----- Crée la matrice nulle de taille $n \times m$
`np.eye(n)` ----- Crée la matrice identité de taille n
`np.diag(L)` ----- Crée la matrice diagonale dont les termes diagonaux sont les éléments de la liste L
`np.transpose(M)` ----- Renvoie la transposée de M
`np.dot(M,P)` ----- Renvoie le produit matriciel MP
`np.sum(M)` ----- Renvoie la somme de tous les éléments de M
`np.prod(M)` ----- Renvoie le produit de tous les éléments de M
`np.max(M)` ----- Renvoie le plus grand élément de M
`np.min(M)` ----- Renvoie le plus petit élément de M
`np.shape(M)` ----- Renvoie dans un couple le format de la matrice M
`np.size(M)` ----- Renvoie le nombre d'éléments de M

Logique

`a == b` ----- Teste l'égalité « $a = b$ »
`a != b` ----- Teste « $a \neq b$ »
`a < b` ----- Teste « $a < b$ »
`a <= b` ----- Teste « $a \leq b$ »
`a > b` ----- Teste « $a > b$ »
`a >= b` ----- Teste « $a \geq b$ »
`not A` ----- Renvoie la négation de A
`A and B` ----- Renvoie « A et B »
`A or B` ----- Renvoie « A ou B »
`True` ----- Constante booléenne « Vrai »
`False` ----- Constante booléenne « Faux »

Numpy.linalg

`import numpy.linalg as la`
`la.inv(M)` ----- Renvoie l'inverse de la matrice M si elle est inversible
`la.eigvals(M)` ----- Renvoie la liste des valeurs propres de M
`la.eig(M)` ----- Renvoie un couple L, P où L est la liste des valeurs propres de M et P la matrice de passage associée
`la.matrix_rank(M)` ----- Renvoie le rang de M

Random

`import numpy.random as rd`
`rd.random()` ----- Simule une réalisation d'une variable $X \mapsto \mathcal{U}([0,1])$
`rd.randint(a,b)` ----- Simule une réalisation d'une variable $X \mapsto \mathcal{U}([a,b])$
`rd.gauss(0,1)` ----- Simule une réalisation d'une variable $X \mapsto \mathcal{N}(0,1)$
`rd.choice(L)` ----- Choisit aléatoirement un élément de la liste L

Math

`import numpy as np`
`np.atan(x)` ----- Renvoie $\arctan(x)$ `np.sqrt(x)` --- Renvoie $\sqrt{x}$ si $x \geq 0$
`np.floor(x)` ----- Renvoie $\lfloor x \rfloor$ `np.log(x)` --- Renvoie $\ln(x)$ si $x > 0$
`np.factorial(n)` --- Renvoie $n!$ si $n \in \mathbb{N}$ `np.exp(x)` --- Renvoie e^x

Matplotlib.pyplot

`import matplotlib.pyplot as plt`
`plt.plot(X,Y,'+-r')` ----- Génère la courbe des points définis par les listes X et Y (abscisses et ordonnées) avec les options :

- symbole : 'o', 'point', 'h', 'hexagone', '+', 'plus', 'x', 'croix', '*', 'étoile', ...
- ligne : '-', 'trait plein', '--', 'pointillé', '...', 'alterné', ...
- couleur : 'b', 'bleu', 'r', 'rouge', 'g', 'vert', 'c', 'cyan', 'm', 'magenta', 'k', 'noir', ...

`plt.bar(X,Y)` ----- Génère l'histogramme des points définis par les listes X et Y (abscisses et ordonnées)
`plt.axis('equal')` ----- Rend le repère orthonormé
`plt.xlim(xmin,xmax)` ----- Fixe les bornes de l'axe des abscisses
`plt.ylim(ymin,ymax)` ----- Fixe les bornes de l'axe des ordonnées
`plt.show()` ----- Affiche le graphique

Cette liste est non exhaustive. Les candidats sont libres d'utiliser les commandes de leur choix.