Systèmes linéaires en langage Python

Ce TP est à faire **seul·e ou à deux**, au format Jupyter Notebook.

- Ouvrir Firefox et la page suivante : `https://www.jupyter.org/try`.
- Sélectionner `Jupyter Notebook` et renommer le fichier en cliquant sur le titre en haut de la page.
- Enregistrer régulièrement à l'aide de l'icône disquette.
- Si besoin, un tutoriel d'introduction à Python est disponible ici :
  `https://mmorancey.perso.math.cnrs.fr/TutorielPython.html`

Le rendu est obligatoire et se fait dans le dépôt dédié sur AMeTICE. **Un seul étudiant par groupe** effectuera le rendu, en déposant **les trois éléments suivants** :

1. Dans la zone "Texte en ligne", les noms et prénoms des membres du groupe.
2. Le fichier Jupyter Notebook au format `.ipynb`. Pour le récupérer, cliquer sur `Help > Launch Jupyter Notebook File Browser` puis télécharger le fichier à l'aide d'un clic droit.
3. L'impression de votre fichier Jupyter Notebook au format `.pdf`. Pour la récupérer, aller dans le menu de Firefox et sélectionner `Imprimer > Enregistrer au format PDF`.

Tout plagiat ou recours à une IA est interdit et sera très lourdement sanctionné.

1 Implémentation de l'algorithme du pivot de Gauss

On propose de représenter chaque équation linéaire à p inconnues par une liste de longueur $p+1$ contenant ses coefficients dans l'ordre. Par exemple, l'équation

$$-3x_1 + 2x_2 - x_4 = 1$$

sera représentée par la liste

`[-3,2,0,-1,1].`

Un système linéaire à n équations et p inconnues sera représenté par la liste de ses équations. Il s'agit donc d'une liste de listes, qu'on peut voir comme un tableau à deux dimensions correspondant à la matrice augmentée du système dans $\mathcal{M}_{n,p+1}(\mathbb{R})$. Par exemple, le système

$$S = \begin{cases} -3x_1 + 2x_2 - x_4 = 1 \\ -x_2 + 4x_3 = 0 \\ 4x_1 + x_2 + 2x_3 - 2x_4 = 3 \\ x_1 - 2x_3 - x_4 = 2 \end{cases}$$

sera représenté par la liste de listes

`S=[[-3,2,0,-1,1],[0,-1,4,0,0],[4,1,2,-2,3],[1,0,-2,-1,2]].`

1.1 Opérations sur les lignes

On commence par implémenter les opérations élémentaires sur les lignes d'un système linéaire.

1. Ecrire une fonction `permutation(S,i_1,i_2)` qui transforme le système `S` en échangeant les lignes d'indices `i_1` et `i_2`.
2. Ecrire une fonction `dilatation(S,i,x)` qui transforme le système `S` en effectuant une dilatation de facteur `x` sur la ligne `i`.
3. Ecrire une fonction `transvection(S,i_1,i_2,x)` qui transforme le système `S` en effectuant la transvection consistant à modifier la ligne `i_1` par l'ajout de `x` fois la ligne `i_2`.

1.2 Simplification des équations sans inconnues

A chaque itération, l'algorithme du pivot de Gauss doit simplifier les équations sans inconnues : si on obtient une "équation triviale" $0 = 0$ alors on peut la supprimer, et si on obtient une "équation absurde" $0 = a$ avec $a \neq 0$ alors on peut stopper l'algorithme et conclure qu'il n'y a pas de solution. On va donc implémenter ici des fonctions permettant de faire cela.

4. Ecrire une fonction `est_membre_gauche_nul(E)` qui renvoie `True` si le membre gauche de l'équation `E` est égal à 0 et `False` sinon.
5. Ecrire une fonction `ligne_a_simplifier(S)` qui renvoie l'indice d'une équation ayant un membre gauche nul dans le système `S`, ou renvoie `-1` si un tel indice n'existe pas. Attention : on numérote à partir de 0.
   Ex. : `ligne_a_simplifier([[-3,2,0,-1,1],[0,-1,4,0,0],[4,1,2,-2,3],[1,0,-2,-1,2]])` renvoie `-1` mais `ligne_a_simplifier([[3,0,0,0],[0,0,0,4],[-2,1,2,3]])` renvoie `1`.
6. Ecrire une fonction `simplifier(S)` qui fonctionne comme suit :
   - Si le système `S` contient au moins une équation absurde, alors on renvoie `False` ;
   - Sinon, on renvoie `True` et on transforme `S` en y supprimant toutes les équations triviales.

   Attention : la ligne d'indice `i` peut être supprimée avec `del S[i]`, mais vous aurez une erreur si vous utilisez un telle instruction à l'intérieur d'une boucle parcourant la liste `S` !
   Ex. : Si `S=[[-3,2,0,-1,1],[0,0,0,0,0],[4,1,2,-2,3],[1,0,-2,-1,2]]` alors `simplifier(S)` renvoie `True` et après son exécution on a `S=[[-3,2,0,-1,1],[4,1,2,-2,3],[1,0,-2,-1,2]]`, mais si `S=[[3,0,0,0],[0,0,0,4],[-2,1,2,3]]` alors `simplifier(S)` renvoie `False`.

1.3 Triangularisation du système

7. Ecrire une fonction `coordonnees_pivot(S,k)` qui renvoie les coordonnées [ligne,colonne] du pivot à choisir à l'itération `k` (c'est-à-dire en ignorant les `k` premières lignes du système `S`). On ne demande pas ici de privilégier un coefficient pivot égal à 1 ou -1.
   Ex. : Si `S=[[1,-1,1,-1,2],[0,2,1,-3,1],[0,0,0,4,3],[0,0,5,5,-10],[0,0,1,2,2]]` alors `coordonnees_pivot(S,2)` renvoie `[3,2]` car le pivot est sur la ligne 3 et colonne 2 comme encadré ci-après :

$$\left\{\begin{array}{rrrrcr} x_1 & -x_2 & +x_3 & -x_4 & = & 2 \\ & 2x_2 & +x_3 & -3x_4 & = & 1 \\ & & & 4x_4 & = & 3 \\ & & \boxed{5x_3} & +5x_4 & = & -10 \\ & & x_3 & +2x_4 & = & 2 \end{array}\right.$$

8. Ecrire une fonction `pivoter(S,k)` qui calcule les coordonnées [ligne,colonne] du pivot à choisir à l'itération `k` puis transforme le système `S` en effectuant les opérations suivantes : échange de lignes pour placer l'équation pivot en position `k`, puis transvections nécessaires pour que l'inconnue en tête de l'équation pivot n'apparaisse plus dans les équations du dessous.
   Ex. : Si `S=[[1,-1,1,-1,2],[0,2,1,-3,1],[0,0,0,4,3],[0,0,5,5,-10],[0,0,1,2,2]]` (on a repris l'exemple de la question précédente) alors après exécution de l'instruction `pivoter(S,2)` on obtient `S=[[1,-1,1,-1,2],[0,2,1,-3,1],[0,0,5,5,-10],[0,0,0,4,3],[0,0,0,1,4]]` car le système est devenu :

$$\left\{\begin{array}{rrrrcr} x_1 & -x_2 & +x_3 & -x_4 & = & 2 \\ & 2x_2 & +x_3 & -3x_4 & = & 1 \\ & & \boxed{5x_3} & +5x_4 & = & -10 \\ & & & 4x_4 & = & 3 \\ & & & x_4 & = & 4 \end{array}\right.$$

9. Ecrire une fonction `triangulariser(S)` qui fonctionne comme suit :
   - Si le système `S` n'a aucune solution, alors on renvoie `False` ;
   - Sinon, on renvoie `True` et on transforme `S` en le triangularisant.

1.4 Résolution du système

10. Ecrire une fonction `resoudre(S)` qui renvoie la solution du système `S` sous forme d'une liste :
    - Si `S` n'a aucune solution, alors on renvoie la liste vide `[]` ;
    - Si `S` a une unique solution `(x_1,...,x_p)`, alors on renvoie `[[x_1,...,x_p]]` ;
    - Si `S` a une infinité de solutions, par exemple si l'ensemble des solutions est

    $$\begin{pmatrix} 2 \\ -1 \\ 4 \end{pmatrix} + \mathbb{R}\begin{pmatrix} 1 \\ 0 \\ -3 \end{pmatrix} + \mathbb{R}\begin{pmatrix} 5 \\ -2 \\ 6 \end{pmatrix}$$

    alors on renvoie `[[2,-1,4],[1,0,-3],[5,-2,6]]`. Ce troisième cas est plus difficile !

2 Utilisation de bibliothèques existantes

2.1 Résolution de systèmes de Cramer avec NumPy

`NumPy` est une bibliothèque Python destinée à manipuler des matrices ou tableaux multidimensionnels. Elle permet entre autres de résoudre des systèmes de Cramer. Il faut pour cela les définir sous forme matricielle avec la fonction `array`, et les résoudre grâce à la fonction `linalg.solve`.

11. Exécuter les instructions suivantes :

```
import numpy as np
A = np.array([[-2,1,-1,-1],[2,2,0,-2],[-2,-1,-1,-2],[-2,1,0,-1]])
Y = np.array([[-1],[1],[1],[-1]])
X = np.linalg.solve(A,Y)
print(X)
```

On a résolu ici le système $AX = Y$ correspondant au TD2 exercice 4 question (a) :

$$A = \begin{pmatrix} -2 & 1 & -1 & -1 \\ 2 & 2 & 0 & -2 \\ -2 & -1 & -1 & -2 \\ -2 & 1 & 0 & -1 \end{pmatrix}, \qquad Y = \begin{pmatrix} -1 \\ 1 \\ 1 \\ -1 \end{pmatrix}.$$

Cette fonction est limitée, car elle génèrera une erreur si le système n'est pas un système de Cramer.

2.2 Résolution de systèmes quelconques avec SymPy

`SymPy` est une bibliothèque Python consacrée au calcul symbolique. Elle permet donc de manipuler des expressions avec des symboles (créés avec la fonction `symbols`), comme par exemple des équations (créées avec la fonction `Eq`). Grâce à la fonction `solve`, qui prend en argument une liste d'équations et la liste des symboles inconnus, on peut résoudre des systèmes linéaires quelconques. Même dans le cas où l'ensemble des solutions est infini, le calcul symbolique permet de l'écrire sous forme paramétrique.

12. Exécuter les instructions suivantes :

```
import sympy as sp
x1,x2,x3,x4 = sp.symbols('x1 x2 x3 x4')
Eq1 = sp.Eq(3*x1 + 4*x2 + 1*x3 + 2*x4, 3)
Eq2 = sp.Eq(6*x1 + 8*x2 + 2*x3 + 5*x4, 7)
Eq3 = sp.Eq(9*x1 + 12*x2 + 3*x3 + 10*x4, 13)
sp.solve([Eq1,Eq2,Eq3],[x1,x2,x3,x4])
```

On a résolu ici le système du TD2 exercice 4 question (c) :

$$\begin{cases} 3x_1 + 4x_2 + x_3 + 2x_4 = 3 \\ 6x_1 + 8x_2 + 2x_3 + 5x_4 = 7 \\ 9x_1 + 12x_2 + 3x_3 + 10x_4 = 13 \end{cases}.$$