TP 10 : Résolution de systèmes et pivot de Gauss

Devoir à la maison. Le TP est à terminer à la maison. La section 3 est à rendre sur feuille pour le 17/03.

Fichier annexe. Télécharger sur le site web le fichier `annexe_tp10_gauss.py`.

Rappels. Si `M` est une liste de n listes contenant elles-mêmes m éléments, on la considère comme une matrice $n \times m$. `n` est accessible par `len(M)`, `m` est accessible par `len(M[0])`. Pour $0 \leq$ `i` $< n$ et $0 \leq$ `j` $< m$, l'élément à la i-ème ligne et la j-ème colonne est accessible par `M[i][j]`. Un vecteur est représenté comme une matrice $n \times 1$. Dans ce TP, on ne considérera que des matrices carrées, et des vecteurs.

1 Le pivot de Gauss

L'algorithme du pivot de Gauss. Les matrices sont donc représentées comme des listes de listes. Par exemple dans le système suivant :

$$AX = Y \qquad \text{avec} \qquad A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & -1 & 2 \\ 2 & -1 & 1 \end{pmatrix}, \quad X = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} \quad \text{et} \quad Y = \begin{pmatrix} 2 \\ 9 \\ 7 \end{pmatrix}$$

la matrice A et le vecteur Y sont donnés comme suit :

```
A=[[1, 1, 1], [1, -1, 2], [2, -1, 1]]
Y=[[2], [9], [7]]
```

Algorithme 1 : Algorithme du pivot

Entrées : Une matrice A de taille $n \times n$, un vecteur Y de taille $n \times 1$
Sortie : Un vecteur X solution de $AX = Y$
Faire un copie des matrices A et Y ;
pour chaque i *allant de* 0 *à* $n-1$ **faire**
  $j \leftarrow$ indice_pivot(A, i) ;
  si $j \neq i$ **alors**
    echange_ligne(A, i, j) ;
    echange_ligne(Y, i, j)
  pour chaque j *allant de* $i+1$ *à* $n-1$ **faire**
    $\mu \leftarrow -A[j][i]/A[i][i]$;
    transvection(A, j, i, μ) ;
    transvection(A, j, i, μ)
$X \leftarrow$ un vecteur nul de taille $n \times 1$;
pour chaque i *allant de* $n-1$ *à* 0 *par pas de* -1 **faire**
  $X[i][0] \leftarrow Y[i][0] - \frac{1}{A[i][i]} \sum_{j=i+1}^{n-1} A[i][j] \times Y[j][0]$
retourner X

L'algorithme du pivot (algorithme 1) nécessite l'écriture des quatre fonctions de base suivantes :

— `echange_lignes(A,i,j)`, qui réalise l'échange de lignes $L_i \leftrightarrow L_j$ sur la matrice A.
— `transvection(A,i,j,mu)`, qui réalise l'opération de transvection $L_i \leftarrow L_i + \mu L_j$ sur les lignes de la matrice A.
— `indice_pivot(A,i)`, qui cherche et renvoie l'indice de la ligne j tel que le coefficient $a_{j,i}$ soit maximal en valeur absolue parmi les éléments $a_{k,i}$ avec $i \leq k \leq n-1$ où n est le nombre de lignes de la matrice A.
— `copie_matrice(A)`, qui renvoie une copie de la matrice A passée en entrée.

Il faut savoir recoder ces fonctions, mais elles vous sont données pour ce TP, ainsi que le code du pivot (dans l'annexe).

Question 1. Vérifier que tout fonctionne en résolvant le système $AX = Y$ suivant :

```
>>> A=[[1, 1, 1], [1, -1, 2], [2, -1, 1]]
>>> Y=[[2], [9], [7]]
>>> X=gauss(A,Y)
>>> print(X)
[[1.0], [-2.0], [3.0]]
```

Remarque : il est pratique d'utiliser la fonction (donnée dans l'annexe) `affiche(A)` qui affiche la matrice `A`. Pour ce faire, on imprime simplement à l'écran la matrice convertie en tableau numpy (via `np.array(A)`), où numpy a été importé sous le nom `np`.

Question 2. Exécuter le code suivant (donné dans l'annexe) :

```
from random import randint
n=5
A=[[0]*n for _ in range(n)]
for i in range(n):
    for j in range(n):
        A[i][j]=randint(0,50)
Y=[[sum(A[i])] for i in range(n)]
X=gauss(A,Y)
print(X)
```

La solution est sensée être un vecteur colonne constitué de 1. Vérifier que la solution obtenue n'est qu'approchée.

Question 3. *Utilisation de Numpy.* En transformant nos matrices en tableaux Numpy (comme ceux vus dans le TP sur les images), on peut utiliser les fonctions Numpy pour résoudre des systèmes. La syntaxe est la suivante (c'est l'exemple du cours) :

```
import numpy as np
A=np.array([[1, 1, 1], [1, -1, 2], [2, -1, 1]])
Y=np.array([[2], [9], [7]])
X=np.linalg.solve(A,Y)
```

En fait, la conversion en tableaux Numpy se fait automatiquement à l'utilisation de `np.linalg.solve`. Reprendre la question précédente avec Numpy. On utilisera `X.tolist()` pour transformer les entrées d'un tableau Numpy `X` en liste (Numpy à tendance à écourter l'affichage, on a l'impression qu'il calcule exactement mais non !)

Remarque : Numpy est souple d'utilisation : si on lui donne le second membre `Y` comme une liste (ou un tableau Numpy à une dimension), il renvoie un tableau Numpy à 1 dimension. Si on lui donne une matrice colonne (de dimension 2, comme ci-dessus), il renvoie la solution sous cette forme également.

On va maintenant travailler avec la famille des matrices H_n qui sont connues pour être *très mal conditionnées*, c'est-à-dire que les résultats fournis par l'algorithme du pivot sont très vite très éloignés de la réalité, et on verra que Numpy ne fait pas mieux que nous.

Question 4. Écrire une fonction `matrice_hilbert(n)` prenant en entrée un entier naturel strictement positif, et retournant la matrice H_n de taille $n \times n$, dont le coefficient en case (i, j) est $\frac{1}{i+j+1}$ (lignes et colonnes sont indexées à partir de 0).

Question 5. On considère ici $n = 5$. Résoudre le système $H_5X = Y$, avec pour Y les deux vecteurs ci-desous (donnés dans l'annexe) :

$$\begin{pmatrix} -7.7 \\ -6 \\ -2.1 \\ -0.5 \\ 0 \end{pmatrix} \quad \text{et} \quad \begin{pmatrix} -7.7 \\ -6 \\ -2.1 \\ -0.4 \\ 0 \end{pmatrix}$$

On voit donc qu'une petite variation sur Y peut induire une grande variation sur X : cela est du au mauvais conditionnement de H_5. La notion de conditionnement est trop dure à définir pour nous en première année, mais elle ressemble à celle vue en cours pour l'étude des suites de la forme $u_{n+1} = f(u_n)$: plus le conditionnement est élevé, plus l'écart entre la solution de $AX = Y$ et celle de $AX = Y'$ (avec Y' « proche » de Y) est important.

Module fractions et résolution sur $\mathbb{Q}$. En travaillant sur les rationnels, on n'introduit pas d'erreurs d'approximation, et on résout nos systèmes de manière *exacte*. L'algorithme du pivot fonctionne aussi sur les rationnels, s'il est correctement écrit. Pour travailler sur $\mathbb{Q}$, il faut importer la fonction `Fraction` du module `fractions`. Pour construire un rationnel, on donne son numérateur et son dénominateur de la façon suivante :

```
from fractions import Fraction
r=Fraction(p,q)
```

Les opérations `+`, `-`, `*`, `/`,... s'appliquent ensuite sur des rationnels comme sur des flottants. Si l'un des argument est un entier et l'autre un rationnel, le résultat sera rationnel.

Question 6. Écrire une fonction `matrice_hilbert_Q(n)` créant aussi une matrice de Hilbert, mais dont les éléments sont sous forme de fractions.

Question 7. Résoudre, pour quelques n entre 2 et 20, le système $H_nX = Y$, où H_n est la matrice de Hilbert d'ordre n et Y le vecteur ayant un 1 dans sa dernière composante, et des 0 ailleurs. On affichera à l'écran le vecteur X obtenu. On fera trois résolutions : sur les rationnels avec notre algorithme du pivot [1] (pour l'affichage, on pourra utiliser la fonction `affiche_float(M)` fournie, qui affiche à l'écran un vecteur / une matrice de rationnels sous forme de flottants), sur les flottants avec notre algorithme, et sur les flottants avec Numpy. Vérifier qu'à partir de $n = 12$ environ, les deux résolutions flottantes s'éloignent drastiquement de la vraie solution (calculée de manière exacte avec l'algorithme de Gauss sur $\mathbb{Q}$).

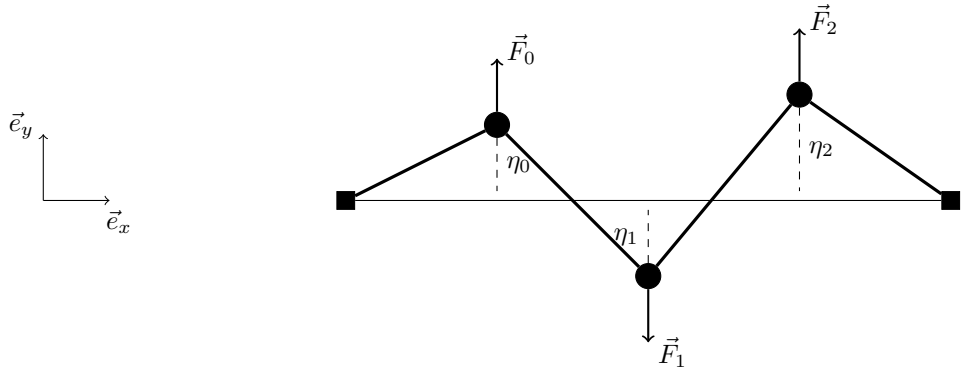
Question 8. Pour mesurer le temps d'exécution d'un processus, on peut utiliser la fonction `process_time` du module `time` (déja importée dans l'annexe). Cette fonction (sans argument) mesure le temps processeur depuis le début de l'exécution. Pour mesurer le temps d'exécution d'un script, il suffit de faire un appel à `process_time` avant et après, la soustraction des deux temps obtenus donne le résultat, comme suit

```
t1 = process_time ()
[instructions dont on veut mesurer le temps]
t2 = process_time ()
# t2 - t1 donne le temps d'exécution.
```

Comparer le temps d'exécution pour des matrices de taille $\simeq 100$ du pivot avec des rationnels et du pivot sur flottants sur la matrice H_n. Le temps d'exécution de l'algorithme avec des rationnels n'est plus en $O(n^3)$, pourquoi ?

2 Une application physique : système de masses sur un fil tendu

On considère un fil horizontal, attachés aux extrémités, sur lequel sont régulièrement réparties N masses, numérotées de 0 à $N-1$. On applique sur chaque masse une force perpendiculaire au fil, également dans le plan horizontal, comme le montre le schéma suivant ($N = 3$ ici).



On cherche à étudier les petits déplacements η_i des masses (largement exagérés sur la figure ci-dessus) dans la direction perpendiculaire à l'axe du fil. Pour de petits déplacements, la longueur du fil entre deux masses reste

1. Il fonctionne sans changement sur les rationnels !

constante égale à ℓ, et la tension du fil est également constante. Au premier ordre, l'équation du mouvement de la masse numéro i dans la direction $\vec{e}_y$ s'écrit :

$$m_i\ddot{\eta_i} = f_i - \frac{T}{\ell}(\eta_i - \eta_{i-1}) + \frac{T}{\ell}(\eta_{i+1} - \eta_i)$$

où ℓ est la longueur du fil entre deux masses (ou une masse et l'extrémité du fil), T est la tension du fil et $\vec{F_i} = f_i\vec{e}_y$, avec la convention $\eta_{-1} = \eta_N = 0$.

En convenant que $\frac{T}{\ell} = 1$ N.m^{-1} pour simplifier, le système linéaire vérifié par $(\eta_0, \dots, \eta_{N-1})$ lorsque le système est à l'équilibre et que les forces $\vec{F_i}$ sont de plus supposées constantes est donc :

$$\begin{pmatrix} 2 & -1 & 0 & 0 & 0 & \cdots & 0 \\ -1 & 2 & -1 & 0 & 0 & \cdots & 0 \\ 0 & -1 & 2 & -1 & 0 & \cdots & 0 \\ \ddots & \ddots & \ddots & \ddots & \cdots & \vdots & \vdots \\ 0 & \cdots & \cdots & -1 & 2 & -1 & 0 \\ 0 & \cdots & \cdots & 0 & -1 & 2 & -1 \\ 0 & \cdots & \cdots & 0 & 0 & -1 & 2 \end{pmatrix} \times \begin{pmatrix} \eta_0 \\ \eta_1 \\ \eta_2 \\ \vdots \\ \vdots \\ \eta_{n-1} \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ \vdots \\ f_{n-1} \end{pmatrix}$$

Ce genre de système *tridiagonal*[2] intervient fréquemment en mécanique.

Question 9. Écrire une fonction `tridiag(n)` prenant en entrée un entier $n \geq 3$, et renvoyant la matrice précédente (sous forme de liste de listes).

Question 10. Reprendre l'étude de la section 1 avec les matrices obtenues pour ce système, et le vecteur Y dont les composantes sont toutes égales à 1. Vérifier que l'algorithme du pivot se passe plutôt bien pour ces matrices là.

3 Polynôme de Lagrange

Vous avez vu en mathématiques que si on se donne un n-uplet $(a_0, \dots, a_{n-1})$ de réels distincts, et un autre n-uplet $(b_0, \dots, b_{n-1})$, alors il existe un unique polynôme $\mathcal{L}$ de degré au plus n tel que $\mathcal{L}(a_i) = b_i$ pour tout $i \in [\![0, n-1]\!]$. Une expression de ce polynôme est :

$$\mathcal{L}(X) = \sum_{i=0}^{n-1} b_i \prod_{j=0, j\neq i}^{n-1} \frac{X - a_j}{a_i - a_j}$$

On représente un polynôme en Python par la liste de ses coefficients, par degré croissant : le polynôme $\sum_{k=0}^{n-1} c_k X^k$ est représenté par la liste `[`c_0`,`c_1`, ..., `c_{n-1}`]`. Notez que la liste vide est associée au polynôme nul. On cherche à calculer précisément les coefficients du polynôme de Lagrange.

3.1 Calcul explicite

Avertissement. Dans les questions qui suivent, ne modifiez pas les listes `P` et `Q` passées en paramètre des fonctions (pas de `append`...). Ou alors, faites une copie préalable avec `P=P[:]`, par exemple. Une première méthode consiste à utiliser la formule ci-dessus pour calculer les coefficients de $\mathcal{L}$. Il faut au préalable écrire des fonctions de sommation et de multiplication de polynômes.

Question 11. Écrire une fonction `somme(P,Q)` faisant l'addition de deux polynômes, de degré possiblement distincts. On renverra une liste dont la taille est le maximum entre celles de `P` et de `Q` (attention aux dépassements d'indice !)

Question 12. Écrire une fonction `mult_scal(P,a)` multipliant le polynôme `P` par le scalaire `a`.

Question 13. Écrire une fonction `prod(P,Q)` effectuant le produit des deux polynômes `P` et `Q`. Deux solutions :

- par calcul direct : on crée une liste de taille `len(P)+len(Q)-1` (cela suffit dès que l'une des deux listes est non vide), que l'on remplit en utilisant $(\sum_{i=0}^{p-1} p_i X^i) \times (\sum_{j=0}^{q-1} q_j X^j) = \sum_{i=0}^{p-1} \sum_{j=0}^{q-1} p_i q_j X^{i+j}$.
- en utilisant les fonctions précédentes : on écrit au préalable une fonction `prod_mon(P, i)` renvoyant la liste associée à $X^i P$ (il suffit de concaténer une liste de `i` zéros à `P`), puis de partir d'une liste vide (polynôme nul), et d'utiliser l'expression $P \times \left(\sum_{j=0}^{q-1} q_j X^j\right) = \sum_{j=0}^{q-1} q_j X^j P$.

2. c'est-à-dire que les seuls coefficients non nuls sont sur la diagonale principale et les deux diagonales situées en dessous et au dessus.

Donner la complexité de `prod(P,Q)` en fonction des tailles de `P` et `Q`.

Question 14. Déduire des questions précédentes une fonction `Lagrange(A,B)` prenant en entrée deux listes de mêmes tailles (les éléments de `A` sont distincts), et renvoyant les coefficients du polynôme de Lagrange associé à `A` et `B` (la formule est rappelée plus haut). Estimer la complexité de votre fonction.

```
>>> Lagrange([0, 1, 2], [-1, 2, 3])
[-1.0, 4.0, -1.0]
```

3.2 Calcul du polynôme interpolateur via une résolution de système

Un autre moyen de calculer le polynôme $\mathcal{L}$ est de résoudre un système linéaire : il suffit d'écrire que $\mathcal{L}(X) = \sum_{k=0}^{n-1} c_k X^k$ vérifie $\mathcal{L}(a_i) = b_i$ pour tout $i \in [\![0, n-1]\!]$: les coefficients c_i cherchés forment alors un vecteur solution d'un système $M \times C = B$, où le second membre B est donné par les b_i.

Question 15. La matrice M associée au système dépend des a_i. Donner cette matrice, et écrire une fonction `Lagrange2(A,B)` résolvant le système après avoir calculé la matrice.

Remarque : il existe des schémas de calcul du polynôme interpolateur de Lagrange meilleurs (plus efficaces en complexité comme en précision) que les deux précédents. En particulier, la matrice M de la question précédente n'est, comme la matrice de Hilbert, pas bien conditionnée. Vous pouvez vérifier qu'à partir de $n = 30$ environ, on obtient des résultats très différents avec les deux méthodes.