T.P. 14 : résolution de systèmes linéaires

1 Le codage des matrices : PYTHON PUR vs NUMPY

1.1 En python pur : on code une matrice par 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]`. Attention les indices commencent à zéro.

1.2 Attention au préformatage pour les matrices en python pur

Un mauvais gag du préformage des listes doubles

On pourrait avoir envie de préformater une matrice 3×3 de zéros de la façon suivante (**essayer ce code !**) :

```
L=[0,0,0]
A=[]# matrice vide
for i in range(3):
    A.append(L)
print(A) # jusqu'ici tout va bien
##
A[0][0]=17
print(A)
```

Que constatez-vous ?

Comment contourner ce mauvais gag ?

a) D'abord en comprenant d'où il vient. Avec `A[0][0]=17` on fait `L[0]=17` et `L` est copié dans chaque ligne de `A`.

b) Le même gag se produit avec :

```
A=[[0]*3]*3
```

car le `*3` le plus à droite va faire des copies non autonomes (alias) de la liste `[0]*3`.

c) Il y a bien sûr bien des façons de faire cela plus correctement : la principe est qu'à chaque tour de la boucle intérieure on doit créer une nouvelle liste de zéros.

Question 1. Ecrire une fonction `MatriceNulle(m,n)` qui renvoie une liste double codant une matrice à m lignes et n colonnes, toute remplie de zéros, que l'on peut ensuite modifier correctement !

d) Le cas écheant, si on veut vraiment copier une ligne d'une matrice dans une autre ligne, avec une copie autonome, on peut utiliser la fonction `deepcopy` du module `copy`.

Question 2. Modifier le code du début du paragraphe (celui du mauvais gag), en rajoutant dans la boucle quelque chose comme :

```
M=deepcopy(L)
 A.append(M)
```

e) Dans ce qui suit, par commodité, on pourra se servir plutôt des `np.array` de `numpy` qui ont des fonctions d'initialisations intégrées comme `np.zeros((n,n))` qui initialise un `np.array` rempli de 0. On a aussi une fonction de copie dans `numpy` qui remplace `deepcopy` qui est simplement `B=np.copy(A)`.

1.3 Les tableaux NUMPY : np.array

Dans toute la suite, on considère qu'on a importé `numpy` via :

```
import numpy as np
```

La syntaxe pour déclarer un tableau NUMPY codant la matrice donnée au début du cours est très proche :

```
A=np.array([[1,2],[3,4]])
```

Cette fois, on a un affichage en deux dimensions :

```
array([[1, 2],
       [3, 4]])
```

On accède aux entrées de deux manières : `A[i][j]` ou `A[i,j]`

Pour initialiser une matrice 3×2 remplies de zéros :

```
A=np.zeros((3,2)) # noter les doubles parenthèses.
print(A)
array([[ 0.,  0.],
       [ 0.,  0.],
       [ 0.,  0.]])
```

Noter que par défaut, ces zéros sont des flottants mais :

```
A=np.zeros((3,2),dtype=int) # dtype pour data type
```

fabriquera une matrice dont les entrées sont des entiers.

> Attention, pour les tableaux `numpy` le type des entrées est fixé au début : un tableau d'entiers ne pourra pas devenir un tableau de flottants, et les divisions par exemple seront de quotients de divisions euclidiennes., ce qui provoquera des erreurs de calculs si vous attendiez des flottants.

1.4 Une alternative : les `np.matrix`

La déclaration est semblable. L'accès aux entrées se fait cette fois exclusivement avec `[i,j]`.

```
B=np.matrix([[1,2],[3,4]])
print(B)
print(B[1,1]) # et pas  B[1][1]
```

> A part cela, la principale différence entre np.array et np.matrix est la fonction pour calculer les produits de matrices : il s'écrit simplement `*` pour le type `matrix` alors qu'on doit utiliser `np.dot(A,B)` pour le produit matriciel de deux `np.array` `A` et `B`.

On peut facilement passer d'un type à un autre :

```
B=np.matrix([[1,2],[3,4]])
C=np.array(B)
```

2 La résolution des systèmes triangulaires

On considère un système $TX = Y$ 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*[1]

> Par commodité, dans ce qui suit, on numérote les lignes comme PYTHON de 0 à $n-1$.

A la ligne L_{n-1}, on va seulement faire un quotient pour trouver $x_{n-1} = y_{n-1}/t_{n-1,n-1}$.

Mais ensuite à la ligne L_i, on obtiendra x_i par la formule :

$$x_i = \frac{1}{t_{i,i}}(y_i - \sum_{j>i} t_{i,j}x_j).$$

Travail à faire : écrire une fonction `resoutTriangle` qui recoit en argument une matrice `T` (supposée T.S. pas besoin de le vérifier) et un vecteur Y (codé au choix par une liste ou un tableau np unidimensionnel) et renvoie le vecteur X tel que $TX = Y$.

Indication – Pour parcourir les lignes de bas en haut, on pourra utiliser un `for i in range(n-1,-1,-1)`, le troisième argument est le pas de -1, le second dit qu'on s'arrête avant que `i` ne soit égal à -1 donc à zéro.

1. De même, un système $BX = Y$ avec B triangulaire inférieure se résout de haut en bas.

3 Codage matriciel de la résolution d'un système : matrice augmentée

On a défini la matrice augmentée d'un système dans le cours.

Travail à faire : Ecrire une fonction `augmente(A,Y)` qui prend comme arguments une matrice A et une vecteur Y et renvoie la matrice augmentée (A, Y).

Ecrire une fonction `Extrait(M)` où `M` est une matrice $n \times (n+1)$ qui renvoie la matrice `A` et la colonne `Y` telles que `M=(A,Y)`.

N.B. Indication : le plus simple est de préformater les tableaux à la bonne taille avec des `np.zeros` puis de les remplir. Pour que la matrice fabriquée avec `np.zeros` ait le même type que les données dans `A` (et `Y`) (flottant ici mais plus loin dans le T.P. fractions par exemple), on utilisera l'argument optionnel `dtype` de `np.zeros`.

4 Résolution d'un système par la méthode du pivot

4.1 Les deux temps de la résolution d'un système par cette méthode

- Premier temps : on ramène notre système à un système triangulaire
- Deuxième temps : la résolution du système triangulaire

On a déjà un algorithme pour la deuxième étape. On va s'occuper ici de la première. La méthode du pivot a été décrite dans le cours : consultez vos notes. Rappelons le pseudo- code du cours :

```
pour i de 0 à n-1 exclu :
        trouver j >=i  tel que |A(j,i)| soit maximum
        échanger L_i et L_j
        pour k de i+1 à n-1 inclus
             L_k <- L_k - mu_k L_i # où mu_k est le quotient A(k,i)/A(i,i)
```

Travail à faire : Implementer cet algorithme dans une fonction `devientTriangle(M)` qui fait le pivot sur les lignes de la matrice augmentée `M=(A,Y)` et renvoie un couple `(T,Z)` avec `T` triangulaire supérieure et `Z` vecteur tels que le système `AX=Y` soit équivalent à `TX=Z`.

En déduire une fonction `resoutSystème` qui reçoit une matrice carrée inversible `A` et un vecteur `Y` et renvoie le vecteur `X` tel que `AX=Y`.

Remarque : Pour vérifier votre résultat, vous pourrez utiliser le produit matriciel dans `numpy`. Avec `np.dot(A,X)` on a le produit matriciel de `A` par `X`.

5 Cas des matrices à coefficients entiers ou rationnels

Question : Comment utiliser les algorithmes précédents pour faire du calcul exact sur les rationnels ?

Réponse : Il suffit de déclarer des matrices `A` et `Y` à coefficients rationnels, mais il faut aussi faire attention à vos fonctions `augmente` et `Extrait` pour qu'elles fabriquent aussi des matrices à coefficients rationnels.

On utilisera le module `fractions` comme suit :

```
>>> from fractions import * # ce qui suit est un exemple de calcul :
>>> Fraction(1,3)+Fraction(1,2)
Fraction(5, 6)
```

a) Vérifier que vos fonctions précédentes s'appliquent bien à des matrices dont les entrées sont des `Fractions` en renvoyant des `Fractions`.

b) **(Par commodité pour la suite) :** écrire une fonction `convert_ratio` qui prend une matrice `A` rectangulaire quelconque dont les coefficients sont supposés entiers et renvoie *une autre matrice* où on a converti chaque entrée `A[i][j]` en la fraction `Fraction(A[i][j],1)`.

Ecrire aussi `convert_float` qui fait l'inverse et convertit les entrées en flottant.

6 Calcul exact sur les rationnels vs. calcul sur les flottants : la matrice de Hilbert

On considère la matrice dite de Hilbert $H_n = \left(\frac{1}{i+j+1}\right)_{(i,j)\in[\![0,n-1]\!]}$.

Cette matrice (qui intervient notamment dans certains problèmes d'analyse) est très connue notamment pour :

- être inversible : son inverse est même à coefficients *entiers* et on peut donner une formule explicite pour cet inverse avec des binomiaux...
- avoir un calcul d'inverse très *instable* du point de vue du calcul numérique.

L'exercice qui suit va illustrer ces affirmations :

a) Ecrire une fonction `hilbert(n)` qui fabrique la matrice de Hilbert H_n en PYTHON comme une matrice à coefficients de type `Fraction`.

b) Appliquer la fonction `devientTriangle` du paragraphe 4.1 à H_n pour $n = 20$: l'entrée la plus petite de H_n est $1/(2n-1)$. Commentez la taille des dénominateurs obtenus pour `T=devientTriangle(hilbert(20))`

c) A l'aide de `convert_float(T)`, que peut-on dire des entrées diagonales de la matrice `T` ? Que penser *a priori* des risques de l'algorithme de résolution appliqué à un système $TX = Y$?

d) On veut expliciter la première colonne de la matrice H_n^{-1}. Pour cela il suffit de résoudre le système $H_nX = E_1$ où E_1 est la première colonne canonique.

Résoudre ce système à l'aide de la fonction `resoutSysteme` du § 4.1 avec H_n codée avec des `Fractions`.

e) Résoudre ce système à l'aide de la fonction `resoutSysteme` du § 4.1 avec H_n codée avec des `floats`.

f) Que dire des résultats obtenus ?

7 Le problème de la sensibilité des solutions aux données dans le second membre

On considère la matrice à coefficients entiers $A = \begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix}$.

On considère les deux colonnes $Y_1 = \begin{pmatrix} 32 \\ 23 \\ 33 \\ 31 \end{pmatrix}$ et $Y_2 = \begin{pmatrix} 32.1 \\ 22.9 \\ 33.1 \\ 30.9 \end{pmatrix}$. Les deux matrices Y_1 et Y_2 sont proches.

Par exemple en notant $||Y||_\infty = \max_{i\in[\![1,4]\!]} |y_{i,1}|$, on a $||Y_1 - Y_2||_\infty = 0.1$.

a) Résoudre, à l 'aide de votre algorithme du pivot les deux systèmes suivants $AX_1 = Y_1$ et $AX_2 = Y_2$ d'inconnues respectives X_1 et X_2 dans $M_{4,1}(\mathbb{Q})$.

b) Que dire de l'écart trouvé $||X_1 - X_2||_\infty$?

c) La grosse différence trouvée peut laisser planer un doute sur la validité des solutions solutions. Vérifier que les solutions trouvées sont bien « justes » en travaillant comme au § 5.

Moralité : pour la matrice A ci-dessus (qui n'a pas été choisie au hasard !) le système $AX = Y$ est très sensible aux variations du second membre !