TD informatique du chapitre 14 : Représentation des nombres flottants

Exercice 1 : Pivot de Gauss (fin)

Dans cet exercice, on utilisera les commandes de calcul matriciel suivantes :

- **import numpy as np** : importation du module numpy abrégé en np,
- **A=np.array([[**$a_{1,1}$**,. . ., **$a_{1,p}$**],. . .,[**$a_{n,1}$**,. . ., **$a_{n,p}$**]])** : création d'une matrice A,
- **A[i,j]** : terme (i, j) de la matrice A,
- **A[i]** : i-ème ligne de la matrice A,
- **M=np.array(A,copy=True,dtype=float)** : M est une copie de A, les éléments de M étant de type flottant,
- **M=np.array(A,copy=True,dtype=np.float64)** : M est une copie de A, les éléments de M étant de type flottant représentés sur 64 bits,
- **M=np.array(A,copy=True,dtype=np.float32)** : M est une copie de A, les éléments de M étant de type flottant représentés sur 32 bits.

Dans cet exercice, on s'intéresse à la résolution d'un système linéaire de la forme :

$$AX = B,$$

où $A \in GL_n(\mathbb{R})$ et $B \in \mathcal{M}_{n,1}(\mathbb{R})$. Le système admet donc une unique solution et A peut être réduite à I_n en utilisant le pivot de Gauss.

On reprend les fonctions écrites dans le TD précédent :

- La fonction triangularisation qui prend comme arguments deux matrices : $A \in GL_n(\mathbb{R})$ et $B \in \mathcal{M}_{n,1}(\mathbb{R})$ et qui renvoie deux matrices : M matrice triangulaire obtenue à partir de A par le pivot de Gauss et N tel que le système soit équivalent à $MX = N$.

```
def triangularisation(A,B) :
    M=np.array(A,copy=True,dtype=float)
    N=np.array(B,copy=True,dtype=float)
     #l'algorithme du pivot de Gauss effectue des divisions
     donc agit sur des nombres flottants
    n=M.shape[0]
     #nombre de lignes de A qui est aussi le nombre de colonnes
    for k in range(n) : #on se place sur le k-ième colonne
        pivot=k #recherche de la ligne sur laquelle est placé le pivot
        while abs(M[pivot,k])<=10**(-10) :
         #remplace le test d'égalité M[pivot,k])==0
            pivot+=1
         #échange de la ligne pivot et de la ligne k sur M :
        X=np.array(M[k],copy=True,dtype=float)
        M[k]=M[pivot]
        M[pivot]=X
         #échange de la ligne pivot et de la ligne k sur N :
        Y=np.array(N[k],copy=True,dtype=float)
        N[k]=N[pivot]
        N[pivot]=Y
        for i in range(k+1,n) : #transvections sur M et N
            x=M[i,k]/M[k,k]
            M[i]-=x*M[k]
            N[i]-=x*N[k]
    return M,N
```

- La fonction remontee qui prend comme arguments deux matrices : $A \in GL_n(\mathbb{R})$ et $B \in \mathcal{M}_{n,1}(\mathbb{R})$ et qui renvoie deux matrices : M matrice réduite obtenue à partir de A par le pivot de Gauss (c'est-à-dire l'identité car A est inversible et N la solution du système.

```
def remontee(A,B) :
    M,N=triangularisation(A,B)
    n=M.shape[0]
    for k in range(n-1,0,-1) : #transvections à partir de la dernière ligne
        for i in range(k) :
            x=M[i,k]/M[k,k]
            M[i]-=x*M[k]
            N[i]-=x*N[k]
    for j in range(n) : # dilatations
        x=M[j,j]
        M[j]*=1/x
        N[j]*=1/x
    return M,N
```

On testera les fonctions écrites sur les exemples suivants :

$$A1 = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}, \quad B1 = \begin{pmatrix} 2 \\ 2 \\ 1 \end{pmatrix}, \text{ la solution est : } X_1 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

$$A2 = \begin{pmatrix} 1 & 0.1 & -0.1 \\ 3 & 0.3 & 0.3 \\ 3 & 0.1 & -0.3 \end{pmatrix}, \quad B2 = \begin{pmatrix} 0.1 \\ 0.9 \\ 0.1 \end{pmatrix}, \text{ la solution est : } X_2 = \begin{pmatrix} 0.1 \\ 1 \\ 1 \end{pmatrix}$$

$$A3 = \begin{pmatrix} 0.2161 & 0.1441 \\ 1.2969001 & 0.8648 \end{pmatrix}, \quad B3 = \begin{pmatrix} 0.144 \\ 0.8642002 \end{pmatrix}, \text{ la solution est : } X_3 = \begin{pmatrix} 2 \\ -2 \end{pmatrix}.$$

1. (a) Reprendre les fonctions précédentes pour écrire deux fonctions analogues triangularisation64 et remontee64 pour lesquelles on impose une représentation des flottants sur 64 bits.
   Que pensez-vous des résultats obtenus ?
   (b) Reprendre les fonctions précédentes pour écrire deux fonctions analogues triangularisation32 et remontee32 pour lesquelles on impose une représentation des flottants sur 32 bits.
   Que pensez-vous des résultats obtenus ?
2. On a choisit, dans les questions précédentes, de prendre comme pivot le premier terme non nul mais ce choix n'est pas toujours judicieux en raison des erreurs d'arrondi.
   Etudions l'exemple suivant :

On considère les matrices :

$$A4 = \begin{pmatrix} 10^{-9} & 1 \\ 1 & 1 \end{pmatrix} \text{ et } B4 = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \text{ la solution est : } X_4 = \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{10^9}{10^9-1} \\ \frac{10^9-2}{10^9-1} \end{pmatrix}$$

On commence par appliquer l'algorithme du pivot de Gauss en arrondissant le résultat à 7 chiffres significatifs (comme dans le cas d'une représentation sur 32 bits) :

$$\begin{aligned}
AX = B &\Leftrightarrow \begin{cases} 10^{-9}x + y &= 1 \\ x + y &= 2 \end{cases} \\
&\Leftrightarrow \begin{cases} 10^{-9}x + y &= 1 \\ -10^9 y &= -10^9 \end{cases} && L_2 \leftarrow L_2 - 10^9 L_1 \\
&\Leftrightarrow \begin{cases} 10^{-9}x + y &= 1 \\ y &= 1 \end{cases} && L_2 \leftarrow \frac{L_2}{-10^9} \\
&\Leftrightarrow \begin{cases} 10^{-9}x &= 0 \\ y &= 1 \end{cases} && L_1 \leftarrow L_1 - L_2 \\
&\Leftrightarrow \begin{cases} x &= 0 \\ y &= 1 \end{cases}
\end{aligned}$$

La solution obtenue est très éloignée de la solution exacte !
On recommence, en prenant le premier pivot sur la deuxième ligne :

$$\begin{aligned}
AX = B &\Leftrightarrow \begin{cases} 10^{-9}x + y &= 1 \\ x + y &= 2 \end{cases} \\
&\Leftrightarrow \begin{cases} y &= 1 \\ x + y &= 2 \end{cases} && L_1 \leftarrow L_1 - 10^{-9} L_2 \\
&\Leftrightarrow \begin{cases} y &= 1 \\ x &= 1 \end{cases} && L_2 \leftarrow L_2 - L_1
\end{aligned}$$

La solution obtenue est beaucoup plus proche de la solution exacte !
Cette amélioration vient du fait, qu'au lieu de diviser par 10^{-9} (ce qui multiplie l'erreur par 10^9), on a divisé par 10^9 (ce qui multiplie l'erreur par 10^{-9}).
On a donc intérêt, afin d'éviter le cumul des erreurs d'arrondis, à utiliser un pivot le plus grand possible (en valeur absolue).
Cette méthode est appelée méthode du pivot partiel.

(a) Tester la fonction remontee32 sur les matrices A_4 et B_4. Que pensez-vous du résultat obtenu ?
(b) En travaillant avec des représentations sur 32 bits, écrire des fonctions triangularisationMax et remonteeMax qui reprennent l'algorithme du pivot de Gauss mais en considérant, non plus le premier terme non nul comme pivot, mais le plus grand terme en valeur absolue.
(c) Tester la fonction remonteeMax sur les matrices A_4 et B_4. Que pensez-vous du résultat obtenu ?