TD 20 - Résolution de systèmes linéaires par pivot de Gauss

Informatique
MPSI-PCSI - Lycée Thiers

Exercice 1.

Exercice 1. *Pivot de Gauss pour une matrice à petite bande.*

Définissons une matrice à petite bande comme une matrice carrée de la forme suivante :

$$\begin{pmatrix} 1 & 0 & \cdots & \cdots & \cdots & 0 \\ a_1 & 1 & \ddots & & & \vdots \\ 0 & a_2 & 1 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & a_{n-1} & 1 & 0 \\ 0 & \cdots & \cdots & 0 & a_n & 1 \end{pmatrix}$$

Remarquons qu'une telle matrice est toujours inversible (car triangulaire inférieure sans aucun 0 sur sa diagonale).

1. Ecrire une fonction `MatBande(L)` qui prend en paramètre une liste `L=[a1,a2,...an]` de longueur quelconque et qui retourne la matrice `numpy` à petite bande comme définie ci-dessus.
2. Ecrire une fonction `GaussBande(L,B)` prenant en paramètre une liste `L=[a1,a2,...an]` et un vecteur `B = np.array([b0,b1,...,bn])` qui appelle la fonction `MatBande(L)` puis lui applique un pivot de Gauss (simplifié) pour retourner l'unique solution du système linéaire $A.X = B$ où A est la matrice à petite bande retournée par `MatBande(L)`.
3. Quelle est sa complexité en temps en fonction du nombre d'équations ?
4. Pourriez-vous écrire une version de meilleure complexité ?

Exercice 1.

1.

```
import numpy as np
def MatBande(L):
    n = len(L)
    M = np.zeros((n+1,n+1))
    M[0,0] = 1
    for i in range(1,n+1):
        M[i,i] = 1
        M[i,i-1] = L[i-1]
    return M
```

2. On aura besoin des 2 fonctions :

```
def Matrice(A,B):
    n = len(A)
    M = np.empty((n,n+1))
    M[:,:n] = A
    M[:,n] = B
    return M
def transvection(M,k,i,mu): # Lk <- Lk - mu*Li
    M[k,:] -= mu * M[i,:]
```

Exercice 1. Corrigé

```
def GaussBande(L,B):
    n = len(L)
    A = MatBande(L)
    M = Matrice(A,B)
    for i in range(n):
        mu = L[i]
        transvection(M,i+1,i,mu)
    return M[:,n+1]
```

3. De complexité quadratique (car `Matrice` est de complexité quadratique et `transvection` de complexité linéaire).

4. On peut écrire une version de complexité linéaire de l'algorithme :

```
def GaussBande1(L,B):
    B = B[:]
    n = len(L)
    for k in range(1,n+1):
        B[k] -= L[k-1]*B[k-1]
    return B
```

Exercice 2 : Enoncé

Exercice 2. *Pivot de Gauss pour une matrice* 3×3 *triangulaire supérieure.*

On souhaite écrire une fonction de résolution par la méthode du pivot de Gauss d'un système linéaire de 3 équations à 3 inconnues x, y, z, échelonné, c'est à dire de la forme suivante :

$$(*) \quad \begin{cases} a \cdot x + b \cdot y + c \cdot z = g \\ d \cdot y + e \cdot z = h \\ f \cdot z = i \end{cases}$$

Rappelons que l'instruction `np.zeros((3,3))` crée une matrice `numpy` carrée de taille $(3) \times (3)$ remplie de zéros.

1. Ecrire une fonction `Mat(L)` qui prend en paramètre une liste `L=[a,b,c,d,e,f]` de 6 nombres et qui retourne la matrice `numpy` de la matrice associé au système $(*)$.
2. Ecrire la fonction `Gauss(L,B)` prenant en paramètre la liste `L=[a,b,c,d,e,f]` et la liste `B = [g,h,i]` et qui applique (la dernière étape de) la méthode du pivot de Gauss pour retourner, si elle existe, l'unique solution du système linéaire $(*)$.

   Lorsque le système $(*)$ n'est pas de Cramer la fonction s'arrêtera en retournant un message d'erreur.

Exercice 2. Corrigé

1.

```
def Mat(L):
    A = np.zeros((3,3))
    A[0,0],A[0,1],A[0,2]=L[0],L[1],L[2]
    A[1,1],A[1,2]=L[3],L[4]
    A[2,2] = L[5]
    return A
```

2.

```
def Gauss(L,B):
    A = Mat(L)
    M = Matrice(A,B)
    M[1,:] -= M[2,:]*M[1,2]/M[2,2]
    M[0,:] -= M[2,:]*M[0,2]/M[2,2]
    M[0,:] -= M[1,:]*M[0,1]/M[1,1]
    return [M[0,3]/M[0,0], M[1,3]/M[1,1], M[2,3]/M[2,2]]
```