Informatique en CPGE (2015-2016)
Corrigé TP 11 : résolution numérique d'un système linéaire : méthode de Gauss

Exercice 1

1. Ecrire une fonction **det2x2(m)** qui prend en argument une matrice de type (2,3), et renvoie le déterminant de la matrice carrée associée, soit $d = a_{0,0}a_{1,1} - a_{1,0}a_{0,1}$.

```
def det2x2(m):
    return m[0][0]*m[1][1]-m[0][1]*m[1][0]
mat=[[2,2,-4],[5,13,7]]
print(det2x2(mat1))
```

2. Ecrire une fonction **transvection(m)** qui prend en entrée une matrice (2,3). La fonction teste si le coefficient $a_{0,0}$ est nul ou pas ; s'il est nul la fonction échange les deux lignes de la matrice et sinon elle remplace la ligne L_1 par la ligne $L_1 - \dfrac{a_{1,0}}{a_{0,0}}L_0$. La fonction renvoie la nouvelle matrice.

```
def transvection(m):
    if m[0][0]==0:
        m[0],m[1]=m[1],m[0]
    else:
        k=m[1][0]/m[0][0]
        for j in range(3):
            m[1][j]=m[1][j]-k*m[0][j]
    return m
mat=[[2,2,-4],[5,13,7]]
print(transvection(mat))
mat=[[0,-3,6],[7,2,10]]
print(transvection(mat))
```

3. A l'aide de la fonction **print** vérifier que les matrices m_1 et m_2 ont été modifiées après l'utilisation de la fonction **transvection**. Si on ne souhaite pas modifier la matrice d'origine, il nous faut créer et utiliser une copie de cette matrice.

   (a) La fonction **matrice**, dont le code suit, permet de créer une matrice de dimension (n, p) dont tous les coefficients sont nuls :

```
def matrice(n,p):
    return [p*[0] for i in range(n)]
```

   Ecrire une fonction **copie(m)** qui prend en argument une matrice m ; dans le corps de la fonction, on détermine les dimensions de m, puis on crée une nouvelle matrice **mat** de même dimensions que m avec la fonction **matrice(n,p)** définie ci-dessus et on copie alors les coefficients de m dans la matrice **mat**. La fonction **copie** renvoie la matrice **mat** qui est donc une copie de m.

```
def copie(m):
    n=len(m)
    p=len(m[0])
    mat=matrice(n,p)
    for i in range(n): # boucle sur les lignes
        mat[i]=m[i][:]
    return mat
```

(b) Modifier le corps de la fonction **transvection(m)** de la manière suivante : on commence par déterminer les dimensions de **m**, puis on crée une matrice **mat** à l'aide de la fonction **copie** ; on modifie alors le code restant afin de travailler sur la matrice **mat**.

```
def transvection(m):
    n=len(m)
    p=len(m[0])
    mat=copie(m)
    if mat[0][0]==0:
        mat[0],mat[1]=mat[1],mat[0]
    else:
        k=mat[1][0]/mat[0][0]
        for j in range(3):
            mat[1][j]=mat[1][j]-k*mat[0][j]
    return mat
```

(c) Ecrire une fonction **solution(m)** qui prend en entrée une matrice modifiée préalablement par la fonction **transvection**, et qui renvoie la solution du système sous la forme d'une liste (**sol=[sol[0],sol[1]**).

Résoudre le système de la question 1.

```
def solution(m):
    sol1=m[1][2]/m[1][1]
    sol0=(m[0][2]-m[0][1]*sol[1])/m[0][0]
    sol=[sol0,sol1]
    return sol
```

(d) Ecrire une fonction **gauss(m)** qui prend en argument la matrice m d'un système et renvoie la solution du système. On fera appel aux fonctions **transvection** et **solution** dans le corps de la fonction **gauss**.

```
def gauss(m):
    mat=transvection(m)
    sol=solution(mat)
    return sol
matrice=[[2,2,-4],[5,13,7]]
print(gauss(matrice))
```

4. Utiliser la fonction **gauss** pour résoudre le système $\begin{cases} 10^{-20}x + y = 5 \\ x + y = 10 \end{cases}$ puis le système $\begin{cases} x + y = 10 \\ 10^{-20}x + y = 5 \end{cases}$ obtenu en échangeant les deux lignes.

Pour le premier système on obtient $(0;5)$ pour le deuxième $(5;5)$.

La solution exacte est $x = \dfrac{5}{1-10^{-20}} \approx 5$ et $y = \dfrac{5-10^{-19}}{1-10^{-20}} \approx 5$

Ceci illustre la nécessité de la recherche du plus grand pivot. Modifier alors le code de la fonction **transvection** afin d'échanger les deux lignes L_0 et L_1 si $|a_{1,0}| > |a_{0,0}|$ et résoudre à nouveau les deux systèmes.

```
def transvection(m):
    n=len(m)
    p=len(m[0])
    mat=copie(m)
    if abs(mat[0][0])<abs(mat[1][0]):
        mat[0],mat[1]=mat[1],mat[0]
    k=mat[1][0]/mat[0][0]
    for j in range(3):
```

```
            mat[1][j]=mat[1][j]-k*mat[0][j]
        return mat
```

Exercice 2

1. Ecrire une fonction **transpose(m)** qui prend en argument une matrice carrée m et renvoie la matrice transposée tm (sans modifier m). Rappel : si m a pour coefficient $a_{i,j}$ alors tm a pour coefficients $b_{i,j} = a_{j,i}$.

```
def transpose(m):
    n=len(m)
    p=len(m[0])
    mat=matrice(p,n) #attention aux indices si matrice non carrée
    for i in range(p):
        for j in range(n):
            mat[i][j]=m[j][i]
    return mat
```

2. Avec la fonction **gauss** de l'exercice 1, (et donc avec les fonctions **matrice**, **copie**, **transvection** et **solution**), résoudre les systèmes $\begin{cases} 3x + 4y = 1 \\ 5x + 7y = 0 \end{cases}$ puis $\begin{cases} 3x + 4y = 0 \\ 5x + 7y = 1 \end{cases}$.

   Quel est le lien entre les deux solutions et l'inverse de la matrice $\begin{pmatrix} 3 & 4 \\ 5 & 7 \end{pmatrix}$?

   Les solutions représentent les colonnes de l'inverse de la matrice. En transposant on obtient donc l'inverse de la matrice.

3. Ecrire une fonction **inverse(m)** qui prend en argument une matrice carrée. La fonction crée une matrice carrée nulle **mat_inv**. Ensuite elle crée la matrice correspondant au premier système précédent, le résout avec la fonction **gauss** et stocke la solution dans la première ligne de la matrice **mat_inv** ; idem pour le deuxième système dont la solution est stockée dans la deuxième ligne de la matrice **mat_inv**. Il n'y a plus qu'à compléter avec un appel à la fonction **transpose** et renvoyer la matrice **mat_inv**.

```
def inverse(m):
    n=len(m)
    mat_inv=matrice(n,n)
    m1=copie(m)
    m1=[m1[0]+[1],m1[1]+[0]]
    mat_inv[0]=gauss(m1)
    m2=copie(m)
    m2=[m2[0]+[0],m2[1]+[1]]
    mat_inv[1]=gauss(m2)
    mat_inv=transpose(mat_inv)
    return mat_inv
```

4. Tester le programme sur différentes matrices.

Exercice 3
Partie 1

```
def matrice(n,p):
    mat=[p*[0] for i in range(n)]
```

```
    return mat

def pivot(m,s):
    n=len(m)
    numpiv=s # numero du pivot provisoire
    for i in range(s+1,n): # boucle sur les lignes
        if abs(m[i][s])>abs(m[numpiv][s]):
            numpiv=i
    return numpiv

""" test
m=[[7,5,-4,4],[0,3,-1,5],[0,6,-3,-4]]
print(pivot(m,1))
"""

def copie(m):
    n=len(m)
    p=len(m[0])
    mat=matrice(n,p)
    for i in range(n): # boucle sur les lignes
        mat[i]=m[i][:]
    return mat

def permute(m,i,j):
    n=len(m)
    p=len(m[0])
    mat=copie(m)
    for k in range (p): # boucle sur les colonnes
        mat[i][k],mat[j][k]=mat[j][k],mat[i][k]
    return mat

def transvection(m,s): # s numero du pivot
    n=len(m)
    p=len(m[0])
    mat=copie(m)
    for i in range(s+1,n): # boucle sur les lignes
        k=m[i][s]/m[s][s]
        for j in range (s,p): # boucle sur les colonnes
            mat[i][j]=mat[i][j]-k*mat[s][j]
    return mat

"""test
m=[[7,5,-4,4],[0,6,-3,-4],[0,3,-1,5]]
print(transvection(m,1))
"""

def solution(m):
    n=len(m)
    p=len(m[0])
    sol=n*[0]
    for i in range(n-1,-1,-1): # boucle sur les lignes
        xi=m[i][p-1] # calcul inconnue numéro i
        for j in range(i+1,p-1):
            xi-=m[i][j]*sol[j]
        sol[i]=xi/m[i][i]
    return sol

"""test
m=[[7, 5, -4, 4], [0, 6, -3, -4], [0, 0.0, 0.5, 7.0]]
```

```
print(solution(m))
"""

def gauss(m):
    n=len(m)
    for s in range(n-1):
        piv=pivot(m,s)
        if piv !=s:
            m=permute(m,s,piv)
        m=transvection(m,s)
    sol=solution(m)
    return sol

""" test
m=[[1,1,1,1,1,5],[1,1,1,1,0,4],[1,1,1,0,0,3],[1,2,3,4,5,15],[0,1,1,1,1,4]]
print(gauss(m))
"""
```

Partie 2

```
from gaussToto import *

fic=open("systemes.txt","r")
n=int(fic.readline().rstrip())
while n!=0:
    m=matrice(n,n+1)
    for i in range(n):
        ligne=list(fic.readline().rstrip().split(" "))
        for j in range(len(ligne)):
            m[i][j]=float(ligne[j])
    print(gauss(m))
    n=int(fic.readline().rstrip())
fic.close()
```