TD 21- Proposition de solutions

Solution 1 Echange de ligne

Echange de lignes

```
import numpy as np
A=np.array([[1,2,1,0],[0,0,2,-1],
 [-1,1,-1,2],[1,0,0,-1]])
def echligne(M,i,j):
    B=1*M
    B[i,:],B[j,:]=1*M[j,:],1*M[i,:]
    return B
```

```
-->B=echligne(A,1,2)
-->B=echligne(B,2,3)
-->print(B)
[[ 1  2  1  0]
 [-1  1 -1  2]
 [ 1  0  0 -1]
 [ 0  0  2 -1]]
```

Solution 2 Recherche d'un pivot

Recherche d'un pivot

```
def pivot(M,i):
    n,j=len(M),i
    for k in range(i+1,n):
        if abs(M[k,i])>abs(M[j,i]):
            j=k
    return j
```

```
--> print(pivot(A,1))
2
```

Solution 3 Elimination d'un coefficient

Elimination d'un coefficient

```
def elimine(M,i,j):
    B=1*M
    B[j]=M[i,i]*M[j]-M[j,i]*M[i]
    return B
```

```
--> B=elimine(A,0,2)
--> B=elimine(B,0,3)
--> print(B)
[[ 1  2  1  0]
 [ 0  0  2 -1]
 [ 0  3  0  2]
 [ 0 -2 -1 -1]]
```

Solution 4 Réduction du pivot de Gauss

pivot de Gauss

```
def Gauss(M,B=False):
# verification que M est carree
    t=np.shape(M)
    assert len(t)==2 and t[0]==t[1],
        'matrice non carree'
    n=t[0]
# si B==False prendre B=In -> calcul de l'inverse
    if type(B)==bool:
        B=np.identity(n)
# verification que B est de la bonne taille
    tb=np.shape(B)
    assert tb[0]==n,"B n'est pas compatible"
# construction de M|B
    T=np.zeros((n,n+tb[1]))
    T[:,:n]=1*M
    T[:,n:]=1*B
# travail sur les lignes
    for i in range(n):
        p=pivot(T,i)
# detecter si M est inversible
        assert abs(T[p,i])>1e-14,
                'matrice non inversible'
    # positionnement du pivot
        if p!=i:
            T=echligne(T,i,p)
    # elimination sur la colonne
        for j in range(n):
            if j!=i:
                T=elimine(T,i,j)
    # normalisation des coeff diagonaux
    T=normalise(T)
    return T
```

Lors de l'exécution de `Gauss(A)`, la matrice `B` n'étant pas renseignée, elle vaut par défaut I_n. Les colonnes de `T` à partir de la 5ième sont une copie de la matrice `B`. La matrice `T` est de la forme :

1. après la ligne 16 :

$$T = \begin{pmatrix} 1 & 2 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & -1 & 0 & 1 & 0 & 0 \\ -1 & 1 & -1 & 2 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

2. après la ligne 25, pour `i=1` :

$$T=\begin{pmatrix}1 & X & X & X & X & X & X & X\\ 0 & \boxed{X} & X & X & X & X & X & X\\ 0 & X & X & X & X & X & X & X\\ 0 & X & X & X & X & X & X & X\end{pmatrix}$$

3. après la ligne 29, pour `i=1` :

$$T=\begin{pmatrix}1 & 0 & X & X & X & X & X & X\\ 0 & X & X & X & X & X & X & X\\ 0 & 0 & X & X & X & X & X & X\\ 0 & 0 & X & X & X & X & X & X\end{pmatrix}$$

4. après la ligne 31 :

$$T=\begin{pmatrix}1 & 0 & 0 & 0 & X & X & X & X\\ 0 & 1 & 0 & 0 & X & X & X & X\\ 0 & 0 & 1 & 0 & X & X & X & X\\ 0 & 0 & 0 & 1 & X & X & X & X\end{pmatrix}$$

Solution 5 Inverse d'une matrice

Inverse d'une matrice

```
1 def inverse(M):
2     n=len(M)
3     T=Gauss(M)
4     return T[:,n:]
```

Vérification :

```
--> print(inverse(A))
[[ 2. -3. -4. -5.]
 [-1.  2.  3.  4.]
 [ 1. -1. -2. -3.]
 [ 2. -3. -4. -6.]]
--> print(np.dot(A,inverse(A)))
[[  1.0e+00   2.2e-16   4.4e-16   4.4e-16]
 [  0.0e+00   1.0e+00   0.0e+00   0.0e+00]
 [  4.4e-16   0.0e+00   1.0e+00   0.0e+00]
 [ -2.2e-16   0.0e+00   4.4e-16   1.0e+00]]
```

Solution 6 Résolution de systèmes

Systèmes linéaire

```
--> Y=np.array([[1,0],[0,0],[2,0],[-1,1]])
--> print('Y = ',Y)
Y =  [[ 1  0]
 [ 0  0]
 [ 2  0]
 [-1  1]]
--> T=Gauss(A,B=Y)
--> print('Les solutions sont : X= ')
Les solutions sont : X=
--> print(T[:,4:])
[[-1. -5.]
 [ 1.  4.]
 [ 0. -3.]
 [-0. -6.]]
--> print('Verification : AX = ')
Verification : AX =
--> print(np.dot(A,T[:,4:]))
[[  1.00000000e+00   4.44089210e-16]
 [  0.00000000e+00   0.00000000e+00]
 [  2.00000000e+00   0.00000000e+00]
 [ -1.00000000e+00   1.00000000e+00]]
```

Solution 7 Application au calcul du déterminant

Il convient de penser à deux adaptations :

- ne pas faire la phase de normalisation des coefficients diagonaux
- multiplier par (-1) à chaque échange de ligne

```
def det(M):
    t=np.shape(M)
    assert len(t)==2 and t[0]==t[1],
                'matrice non carree'
    n,sg=t[0],1
    for i in range(n):
        p=pivot(M,i)
        if abs(M[p,i])<1e-14:
            return 0
        if p!=i:
            M=echligne(M,i,p)
            sg=-sg
        for j in range(n):
            if j!=i:
                M=elimine(M,i,j)
    return sg*np.prod([M[i,i] for i in range(n)])
```