L3 algèbre effective - Corrigé de l'interrogation du 12 déc. 2017

1. On réécrit la première relation en $u_{n,0} = n! - \sum_{k=1}^{n} u_{n,k}$ (par convention et également dans Python une somme indexée par la liste vide vaut 0). La définition est alors récursive pour la relation $<$sur $\mathbb{N}^2$ définie par $(n,k) < (n',k')$ si $n < n'$ ou si $n = n'$ et $k > 0$ et $k' = 0$. On vérifie que cette relation est bien fondée.

```
def u(n,k):
    if n<0 or k<0:return(0)
    elif k>0:return(binomial(n,k)*u(n-k,0))
    else:return(factorial(n)-sum(u(n,i) for i in range(1,n+1)))
```

```
u(10,3)
```

222480

2.

```
def inv(o):
    n=len(o)
    if n==0:return(o)
    else:
        if o[n-1][0] in [3,4]:op=o[n-1]
        else:op=o[n-1][:3]+[-o[n-1][3]]
        return([op]+inv(o[:n-1]))
```

On a inv(inv(o))=o donc transf(transf(A,inv(o)),o)=transf(transf(A,oi),inv(oi))=A avec oi=inv(o).

On teste la fonction ci-dessus avec la matrice A de l'ex. 3 et la liste d'opération [[1,2,4,2],[3,1,5],[2,1,2,-1],[4,3,4]]

```
o=[[1,2,4,2],[3,1,5],[2,1,2,-1],[4,3,4]]
print inv(o)
```

[[4, 3, 4], [2, 1, 2, 1], [3, 1, 5], [1, 2, 4, -2]]

```
A=matrix([\
[  2  ,  0  ,  1  , -4  ,  0  ,  6],\
[ -4  ,  2  ,  0  ,  8  , -2  ,-12],\
[  2  ,  6  ,  0  ,-10  , -6  ,  6],\
[  0  , -6  ,  0  ,  6  ,  6  ,  0]\
]);show(A)
B=transf(A,o);show(B)
C=transf(B,inv(o));show(C)
print 'A=C ?',A==C
```

$$\begin{pmatrix} 2 & 0 & 1 & -4 & 0 & 6 \\ -4 & 2 & 0 & 8 & -2 & -12 \\ 2 & 6 & 0 & -10 & -6 & 6 \\ 0 & -6 & 0 & 6 & 6 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 2 & -26 & 1 & -12 & 6 & 18 \\ -2 & 18 & 0 & 8 & -4 & -12 \\ 6 & 6 & 0 & 6 & 0 & 0 \\ -6 & -14 & 0 & -10 & 2 & 6 \end{pmatrix}$$

$$\begin{pmatrix} 2 & 0 & 1 & -4 & 0 & 6 \\ -4 & 2 & 0 & 8 & -2 & -12 \\ 2 & 6 & 0 & -10 & -6 & 6 \\ 0 & -6 & 0 & 6 & 6 & 0 \end{pmatrix}$$

A=C ? True

3.

```
o=fsmith(A);print o
print
print showop(o)
```

[[1, 1, 3, -2], [1, 2, 3, 0], [1, 4, 3, 4], [1, 5, 3, 0], [1, 6, 3, -6], [2, 2, 1, 0], [2, 3, 1, 0], [2, 4, 1, 0], [3, 1, 3], [1, 3, 2, 2], [1, 4, 2, -4], [1, 5, 2, 1], [1, 6, 2, 6], [2, 3, 2, -3], [2, 4, 2, 3], [2, 3, 4, 2], [1, 4, 3, 3], [1, 3, 4, -2], [1, 4, 3, -2], [1, 5, 3, 0], [1, 6, 3, -15], [2, 4, 3, 0], [1, 5, 4, 0], [1, 6, 4, -6]]

C1 + -2C3 → C1
C2 + 0C3 → C2
C4 + 4C3 → C4
C5 + 0C3 → C5
C6 + -6C3 → C6
L2 + 0*L1 → L2
L3 + 0*L1 → L3
L4 + 0*L1 → L4
C1 C3
C3 + 2C2 → C3
C4 + -4C2 → C4
C5 + 1C2 → C5
C6 + 6C2 → C6
L3 + -3*L2 → L3
L4 + 3*L2 → L4
L3 + 2*L4 → L3
C4 + 3C3 → C4
C3 + -2C4 → C3
C4 + -2C3 → C4
C5 + 0C3 → C5
C6 + -15C3 → C6
L4 + 0*L3 → L4
C5 + 0C4 → C5
C6 + -6C4 → C6

Les opérations de type 1 et 2 avec 0 comme coefficient sont inutiles : elles ne changent pas la matrice.

On filtre les opérations sur les lignes de celles sur les colonnes (et inversement) et des opérations inutiles :

```
Co=[op for op in o if op[0] in [1,3] and op[-1]!=0];print 'Co=',Co
Lo=[op for op in o if op[0] in [2,4] and op[-1]!=0];print 'Lo=',Lo
P=transf(matrix.identity(A.nrows()),Lo)
Q=transf(matrix.identity(A.ncols()),Co)
show('P=',P,'Q=',Q)
show('PAQ=',P*A*Q)
```

Co= [[1, 1, 3, -2], [1, 4, 3, 4], [1, 6, 3, -6], [3, 1, 3], [1, 3, 2, 2], [1, 4, 2, -4], [1, 5, 2, 1], [1, 6, 2, 6], [1, 4, 3, 3], [1, 3, 4, -2], [1, 4, 3, -2], [1, 6, 3, -15], [1, 6, 4, -6]]

Lo= [[2, 3, 2, -3], [2, 4, 2, 3], [2, 3, 4, 2]]

P= $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 3 & 1 & 2 \\ 0 & 3 & 0 & 1 \end{pmatrix}$ Q= $\begin{pmatrix} 0 & 0 & -5 & 13 & 0 & -3 \\ 0 & 1 & -2 & 6 & 1 & 0 \\ 1 & 0 & 2 & -6 & 0 & 0 \\ 0 & 0 & -2 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$

PAQ= $\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 0 & 0 & 0 \\ 0 & 0 & 0 & -6 & 0 & 0 \end{pmatrix}$

D'après la forme de PAQ, le noyau de PAQ admet pour base les deux derniers vecteurs de la base canonique de $\mathbb{Z}^6$ (ceux correspondant aux colonnes nulles de PAQ), alors le noyau de A admet pour base l'image par Q de ces deux vecteurs, c'est à dire les vecteurs de coordonnées les deux dernières colonnes de Q

```
D=P*A*Q
col=[j for j in range(D.ncols()) if D[:,j]==0];col
show(Q[:,col])
```

[4, 5]

$\begin{pmatrix} 0 & -3 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$

```
X=matrix([3,2,0,0,2,-1]).transpose()
A*X
A*X==0
```

[0]
[0]
[0]
[0]

True

Posons B=Q[:,4:]. On cherche Y tel que B*Y=X. Il existe un unique YY tel que Q*YY=X. Forcément les 4 premières coordonnées de YY sont nulles et Y correspond aux 2 dernières.

On a YY=Q(-1)X ; Q(-1) s'obtient par l'instruction Q(-1) ou par la transformation de la matrice unité par inv(Co), mais on a besoin de moins que Q(-1) :

YY s'obtient en interprétant Q(-1)X comme la transformation de X par des opérations sur les lignes ; lesquelles ?

Rq. On peut aussi passer par une réduction de Smith de B : P1*B*Q1*Q1(-1)*Y=P1*X conduit à *Y=Q1*Y1 où Y1 est solution de P1*B*Q1*Y1=P1*X

```
Qinv=transf(matrix.identity(A.ncols()),inv(Co))
```

```
Q*Qinv
YY=Qinv*X;show('YY=',YY)
Y=YY[col];show('Y=',Y)
show(Q[:,col],'*',Y,'=',Q[:,col]*Y)
```

```
[1 0 0 0 0 0]
[0 1 0 0 0 0]
[0 0 1 0 0 0]
[0 0 0 1 0 0]
[0 0 0 0 1 0]
[0 0 0 0 0 1]
```

$$\text{YY=}\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 2 \\ -1 \end{pmatrix}$$

$$\text{Y=}\begin{pmatrix} 2 \\ -1 \end{pmatrix}$$

$$\begin{pmatrix} 0 & -3 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} * \begin{pmatrix} 2 \\ -1 \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \\ 0 \\ 0 \\ 2 \\ -1 \end{pmatrix}$$

On obtient une base de l'image de A en prenant l'image par P(-1) d'une base de l'image de PAQ, c'est à dire l'image par P(-1) des 4 premières colonnes de PAQ, c'est à dire les 4 premières colonnes de AQ (et les deux dernières sont nulles). Ce n'est pas une base de $\mathbb{Z}^4$ sinon A serait surjective donc PAQ également, ce qui n'est pas.

```
Pinv=transf(matrix.identity(A.nrows()),inv(Lo))
show('P^(-1)=',Pinv)

show('A*Q=',A*Q)
A*Q[:,:4]
```

$$\text{P(-1)=}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 3 & 1 & -2 \\ 0 & -3 & 0 & 1 \end{pmatrix}$$

$$\text{A*Q=}\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 6 & -2 & 12 & 0 & 0 \\ 0 & -6 & 0 & -6 & 0 & 0 \end{pmatrix}$$

```
[ 1  0  0  0]
[ 0  2  0  0]
[ 0  6 -2 12]
[ 0 -6  0 -6]
```

4.

```
M=matrix.diagonal([3,6]);M
```

```
def f(x,y):return(x%3==0 and y%6==0)
f(5,6)
```

[3 0]
[0 6]

False

Pour M,x quelconque on se ramène à M nulle hors de la diagonale en remplaçant M par sa réduite de Smith et x par son transformé suivant les opérations sur les lignes associées à la réduction de Smith de M.

```
def g(M,x):#M=matrice, x=liste
    o=fsmith(M)
    lo=[op for op in o if op[0] in [2,4]]
    D=transf(M,o)
    Y=transf(matrix(x).transpose(),lo)
    return(1==prod([Y[i]==0 or (D[i,i]!=0 and Y[i]%D[i,i]==0) for i \
   in range(len(x))]))
```

```
g(M,[3,3])
g(A,[1,0,0,0])
g(A,[0,1,0,0])
```

False

True

False