Oral TD7 : quelques exemples de codes d'algèbre bilinéaire

On importe certains modules

```
from scipy.integrate import quad # calcul d'intégrale
from numpy.polynomial import Polynomial # calculs avec des polynômes
from numpy import sqrt
import numpy as np
import numpy.linalg as alg # commandes d'algèbre linéaire
```

1. Procédé d'orthonormalisation de Gram-Schmidt

On définit le produit scalaire et la norme : ici pour des polynômes $(P|Q) = \int_0^1 P(t)Q(t)\,\mathrm{d}t$

```
def ps(P,Q) :
    return quad(P*Q,0,1)[0]

def norm(P) :
    return sqrt(ps(P,P))
```

Puis on orthonormalise une famille de vecteurs libres (une base en général), représentée par une liste de vecteurs :

```
def GramSchmidt(B) :
    GS = []
    V = B[0]/norm(B[0])
    GS.append(V)
    for k in range(1,len(B)) :
        W = B[k]
        for i in range(k) :
            W -= ps(GS[i],B[k])*GS[i]
        W = W/norm(W)
        GS.append(W)
    return GS
```

Lorsqu'on utilise un autre produit scalaire, seule la fonction `ps` est à modifier

On peut définir la base canonique de $\mathbb{R}_4[X]$ puis l'orthonormaliser

```
Bc = []
for k in range(5) :
    L = [0]*5
    L[k] = 1
    P = Polynomial(L)
    Bc.append(P)
GS = GramSchmidt(Bc)
```

2. Calculs de projection orthogonale

Écrire la matrice dans la base canonique de $\mathbb{R}^4$, canoniquement euclidien, de la projection orthogonale sur le sous-espace vectoriel $F = \{(x,y,z,t) \in \mathbb{R}^4, x+y+z+t = x+2y+3z+4t = 0\}$:

$$v = \pi_F(u) \Leftrightarrow \begin{cases} v \in F \\ u - v \in F^\perp \end{cases}$$

On a $F = \mathrm{Vect}\{(1,-2,1,0),(2,-3,0,1)\}$ donc si on pose $u = (x,y,z,t)$ et $v = \pi_F(u) = (X,Y,Z,T)$, on a

$$v \in F \Leftrightarrow \begin{cases} X+Y+Z+T = 0 \\ X+2Y+3Z+4T = 0 \end{cases}$$

et

$$u - v \in F^\perp \Leftrightarrow \begin{cases} (X-x) - 2(Y-y) + (Z-z) = 0 \\ 2(X-x) - 3(Y-y) + (T-t) = 0 \end{cases}$$

On utilise Python pour résoudre ce système, d'inconnues (X,Y,Z,T) :

```
A = np.array([[1,1,1,1],[1,2,3,4],[1,-2,1,0],[2,-3,0,1]]) # la matrice du syst
    ème

def ProjOrtho(V) :
    B = np.linalg.inv(A)
    x,y,z,t = V
    W = np.array([0,0,x-2*y+z,2*x-3*y+t])
    return(np.dot(B,W.T))    # on met W en colonne pour faire le produit
```

On peut alors écrire la matrice demandée en calculant les projections orthogonales des vecteurs de $\mathcal{B}_c$:

```
Bc = []
for k in range(4) :
    L = [0]*4
    L[k] = 1
    Bc.append(np.array(L))
M = np.array([ProjOrtho(Bc[k]) for k in range(4)])
```

On devrait normalement transposer M (elle est replie en ligne ici) mais la matrice dans une base orthonormale d'un projecteur orthogonal est symétrique.

3. Le théorème spectral

La fonction `alg.eig` permet d'obtenir les valeurs propres et la matrice de passage P pour diagonaliser une matrice mais dans le cas d'une matrice symétrique réelle, cette matrice de passage n'est à priori pas orthogonale. On peut obtenir une matrice orthogonale en orthonormalisant la base des vecteurs colonnes de P
Le produit scalaire canonique de $\mathbb{R}^n$

```
def ps(U,V) :
    return np.vdot(U,V)
```

La matrice de $\mathcal{M}_n(\mathbb{R})$ telle que $a_{i,j} = \begin{cases} 4 & \text{si } i=j \\ 1 & \text{sinon} \end{cases}$

```
def A(n) :
    M = np.zeros((n,n))
    for i in range(n) :
        for j in range(n) :
            M[i,j] = 1
        M[i,i] = 4
    return M
```

Le code pour une matrice de passage orthogonale :

```
n = 5
M = A(n)
P = alg.eig(M)[1]                  # la matrice de passage, non orthogonale
B = [P[:,j] for j in range(n)]     # la liste des vecteurs colonnes
B = GramSchmidt(B)                 # on les orthonormalise
Q = np.array(B)                    # la matrice de passage orthogonale
```