Polynômes orthogonaux

Sur $E = C^0([0,1],\mathbb{R})$ on définit un produit scalaire par

$$\langle f, g\rangle = \int_0^1 f.g$$

On identifie polynômes et fonctions polynomiales ; on note

$$E_n = \mathbb{R}_n[X]$$

Pour $n \geq 1$, E_{n-1} est un hyperplan de E_n ; notons D_n la normale de E_{n-1} dans E_n.

D_n contient un unique polynôme L_n qui vérifie

$$L_n(0) = 1$$

On se propose de le calculer.

1e méthode

L_n est orthogonal à $1, X, X^2, ..., X^{n-1}$, et $L_n(0) = 1$; d'où un système linéaire (système de Cramer) de taille $n+1$ dont la solution représente L_n.

On pourra utiliser *numpy.linalg.solve*

2e méthode

Utiliser l'algorithme d'orthonormalisation de Schmidt.

Programmes

```python
import numpy as np
import matplotlib.pyplot as plt
import numpy.linalg as alg
from numpy.polynomial import Polynomial

nb_points = 101
x = np.linspace(0, 1, nb_points)
y0 = np.zeros(nb_points)
plt.plot(x,y0)                  # tracé de l'axe


# calcul par résolution d'un
# système de Cramer

def legendre(n):
    a = np.zeros((n+1, n+1))
    for i in range(n):
        for j in range(n+1):
            a[i, j] = 1/(i + j + 1)
    a[n, 0] = 1
    b = np.zeros(n+1)
    b[n] = 1
    return Polynomial(alg.solve(a,b))

def tracer(n):
    y = (legendre(n))(x)
    plt.plot(x, y)

for k in range(6,12):
    tracer(k)
```

```
import numpy as np
import matplotlib.pyplot as plt
import numpy.linalg as alg
from numpy.polynomial import Polynomial

nb_points = 101
x = np.linspace(0, 1, nb_points)
y0 = np.zeros(nb_points)
plt.plot(x,y0)                  # tracé de l'axe

def produit_scalaire(p, q):
    return (p*q).integ()(1)



# Calcul des n premiers par récurrence :
# orthogonalisation (et non orthonormalisation)

def legendre(n):
    un = Polynomial([1])
    resultat = [un]
    carre_normes = [produit_scalaire(un, un)]
    X = Polynomial([0, 1])
    for k in range(1, n):
        P = X**k
        Q = P
        for j in range(k):
            vj = resultat[j]
            vj2 = carre_normes[j]
            P = P - (produit_scalaire(Q, vj) /
vj2) * vj
        P = P / P(0)
        resultat.append(P)
        carre_normes.append(produit_scalaire(P,
P))
    return resultat
```

Résultat