Algèbre

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

Exercice 1

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

def charpoly(n):
    return np.poly(A(n))
```

Le script qui suit permet de conjecturer que $\chi_n = X^n - \frac{1}{n}\sum_{k=0}^{n-1} X^k$:

```
for n in range(2, 9):
    print(charpoly(n))
```

Pour afficher les modules des racines de χ_n, le plus simple est de se souvenir qu'il s'agit des valeurs propres de A_n :

```
for n in range(2, 9):
    for z in alg.eigvals(A(n)):
        print(abs(z), end=" ")
    print()
```

On observe que 1 est valeur propre de A_n et que les autres valeurs propres ont un module strictement inférieur à 1.

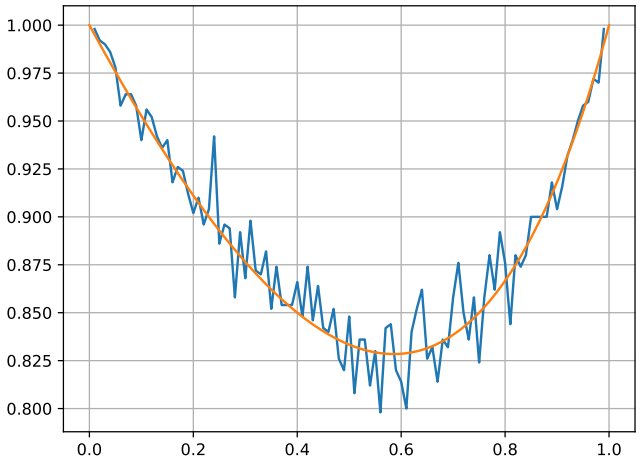
Exercice 2

```
def ecart(a, b):
    M = np.array([[3 * a - 2 * b, -6 * a + 6 * b + 3], [a - b, -2 * a + 3 * b + 1]])
    v1, v2 = alg.eigvals(M)
    e = abs(v1 - v2)
    return round(e, 2)
```

```
import numpy.random as rd

def hasard(p):
    s = 0
    for _ in range(500):
        a, b = rd.geometric(p, 2)
        if ecart(a, b) >= 1e-1:
            s += 1
    return s
```

Pour les tracés demandés, on réalise le script suivant :

```
X, Y = [], []
for k in range(1, 100):
    p = k / 100
    X.append(p)
    Y.append(hasard(p) / 500)
plt.plot(X, Y)

P = np.linspace(0, 1, 256)
F = [(2 - 2 * p + p ** 2) / (2 - p) for p in P]
plt.plot(P, F)

plt.grid()
plt.show()
```



Ceci laisse conjecturer que $\mathbb{P}(\lambda_1 \neq \lambda_2) = \dfrac{2-2p+p^2}{2-p}$.

Exercice 3

```
def A(n, a):
    M = np.zeros((n, n))
    for i in range(n - 1):
        M[i, i + 1] = 1 / a
        M[i + 1, i] = a
    return M
```

On réalise le script suivant :

```
for n in range(3, 9):
    for a in (-2, -1, 1, 2, 3):
        print(alg.eigvals(A(n, a)).round(2))
```

qui permet de conjecturer que la matrice $A_{n,a}$ possède n valeurs propres distinctes *et indépendantes de a*. En particulier, on peut penser que cette matrice est diagonalisable.
On calcule les polynômes $P_1, \dots, P_8$ à l'aide du script :

```
p = [None] * 9
p[1] = Polynomial([0, 1])
p[2] = Polynomial([-1, 0, 1])
for n in range(3, 9):
    p[n] = p[1] * p[n - 1] - p[n - 2]
```

Le script suivant permet de conjecturer que les racines de P_n sont les valeurs propres de $A_{n,a}$, autrement dit que P_n est le polynôme caractéristique de cette matrice :

```
for n in range(3, 9):
    print(p[n].roots().round(2))
```

Exercice 4 On utilise la fonction quad du module `scipy.integrate` pour calculer des intégrales.

```
from scipy.integrate import quad
```

On commence par définir le produit scalaire :

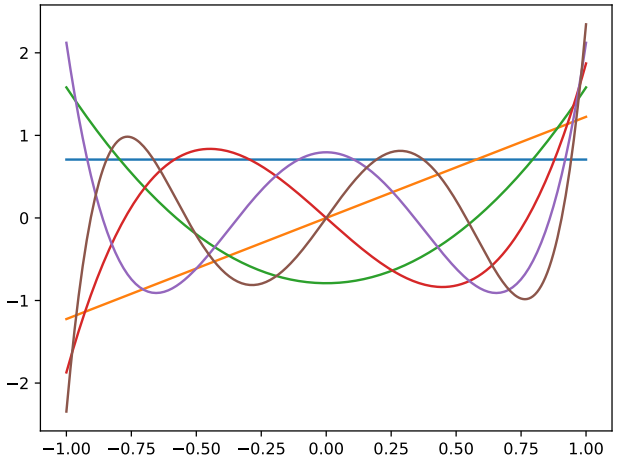
```
def scal(P, Q):
    def f(t):
        return P(t) * Q(t)

    return quad(f, -1, 1)[0]
```

puis on applique le procédé de Schmidt :

```
E = [Polynomial(1 / np.sqrt(2))]        # on définit E[0]
for k in range(1, 6):
    B = Polynomial([0] * k + [1])       # B = X^k
    U = B
    for i in range(k):
        U = U - scal(E[i], B) * E[i]    # U = X^k - son projeté orthogonal
    E.append(U / np.sqrt(scal(U, U)))   # on normalise U
```

Il reste à faire le tracé :

```
X = np.linspace(-1, 1, 256)
for k in range(6):
    Y = [E[k](x) for x in X]
    plt.plot(X, Y)
plt.show()
```



On constate que $\|E_i\|$ semble toujours être atteinte en ± 1.