

Solutions

Exercice 2

a) On peut utiliser la relation $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$ lorsque $1 \leq k \leq n-1$. Cela donne

$$B_{k,n} = \binom{n-1}{k} X^k (1-X)^{n-k} + \binom{n-1}{k-1} X^k (1-X)^{n-k} = (1-X)B_{k,n-1} + XB_{k-1,n-1}$$

Si $k=0$, on a $B_{0,n} = (1-X)^n = (1-X)B_{0,n-1}$ et si $k=n$, alors $B_{n,n} = X^n = XB_{n-1,n-1}$

b) Si on veut une fonction récursive (ce qui n'est pas très intelligent) qui détermine les polynômes, on a :

```
from numpy.polynomial import Polynomial

def B(k,n):
    if k==0 and n==0:
        return Polynomial([1])
    X = Polynomial([0,1])
    if k==0:
        return (1-X)*B(0,n-1)
    if k==n:
        return X*B(n-1,n-1)
    return X*B(k-1,n-1)+(1-X)*B(k,n-1)
```

mais si on veut répondre à la question, on faut la liste de tous les polynômes (on peut utiliser la fonction précédente et en faire la liste mais ...)

```
def liste_B(n):
    if n==0:
        return [Polynomial([1])]

    X = Polynomial([0,1])
    L_a = liste_B(n-1)
    L = [0]*(n+1)
    L[0] = (1-X)*L_a[0]
    L[n] = X*L_a[n-1]
    for k in range(1,n):
        L[k] = (1-X)*L_a[k]+X*L_a[k-1]
    return L
```

c) On a évidemment les valeurs simples lorsque $k=0$ ou $k=n$:

$$\int_0^1 B_{n,n}(t)dt = \int_0^1 t^n dt = \frac{1}{n+1} \text{ et } \int_0^1 B_{0,n}(t)dt = \int_0^1 (1-t)^n dt = \frac{1}{n+1}$$

On essaie de déterminer des relations de récurrence pour le calcul. Comme $B_{k,n}$ admet 0 et 1 comme racine lorsque $1 \leq k \leq n-1$, on obtient que $\int_0^1 B'_{k,n}(t)dt = 0$ pour ces valeurs de k . On calcule alors $B'_{k,n}$. On a

$$B'_{k,n} = k \binom{n}{k} X^{k-1} (1-X)^{n-k} - (n-k) X^k (1-X)^{n-k-1} = n \binom{n-1}{k-1} X^{k-1} (1-X)^{(n-1)-(k-1)} - n \binom{n-1}{k} X^k (1-X)^{n-1-k}$$

ce qui donne $B'_{k,n} = n(B_{k-1,n-1} - B_{k,n-1})$. En intégrant de 0 à 1 - et en notant $I_{k,n} = \int_0^1 B_{k,n}(t)dt$, on obtient

$$\forall k \in \llbracket 1; n-1 \rrbracket, n(I_{k-1,n-1} - I_{k,n-1}) = 0$$

et ainsi $I_{k,n-1} = I_{k-1,n-1}$. La suite est constante et, pour tout $k \in \llbracket 0; n \rrbracket$, $\int_0^1 B_{k,n}(t)dt = \frac{1}{n+1}$.

d) On vérifie que, pour $n \in \mathbb{N}$, la famille $(B_{k,n})_{k \in \llbracket 0; n \rrbracket}$ est une base de $\mathbb{R}_n[X]$. En effet, on considère une combinaison linéaire nulle non triviale $\sum_{k=0}^n \alpha_k B_{k,n}$, on s'intéresse au plus petit entier k_0 tel que $\alpha_k = 0$, ce qui donne un polynôme Q tel que

$$\alpha_{k_0} X^{k_0} (1 - X)^{n-k_0} + X^{k_0+1} Q = 0$$

Ainsi $\alpha_{k_0} (1 - X)^{n-k_0} + X^Q = 0$ et en évaluant en 0, $\alpha_{k_0} = 0$. Tout polynôme de $\mathbb{R}_n[X]$ est alors une combinaison linéaire de $B_{0,n}, \dots, B_{n,n}$ sous la forme $\sum_{k=0}^n \alpha_k B_{k,n}$. Il existe alors une fonction de classe \mathcal{C}^2 sur $[0, 1]$ telle que, pour tout $k \in \llbracket 0; n \rrbracket$, $f\left(\frac{k}{n}\right) = \alpha_k$ (par exemple en prenant un polynôme d'interpolation de Lagrange).

Exercice 3

a) On vérifie rapidement la linéarité. On calcule l'image de la base canonique. On a

$$f(1) = -nX \text{ et, pour tout } k \in \llbracket 1; n \rrbracket, f(X^k) = k(X-a)(X-b)X^{k-1} - nX^{k+1} = (k-n)X^{k+1} - k(a+b)X^k + kabX^{k-1}$$

On remarque que toutes ces images restent de degré au plus n . Cela permet d'obtenir que f est un endomorphisme et d'écrire sa matrice dans la base canonique.

b) on remplit les cases...

```
import numpy as np

def matrice(n,a,b):
    A = np.zeros((n+1,n+1))
    for k in range(n+1):
        if k>=1:
            A[k-1,k] = k*a*b
        if k<n:
            A[k+1,k] = k-n
        A[k,k] = -k*(a+b)
    return A
```

```
>>> matrice(4,1,2)
array([[ 0.,  2.,  0.,  0.,  0.],
       [-4., -3.,  4.,  0.,  0.],
       [ 0., -3., -6.,  6.,  0.],
       [ 0.,  0., -2., -9.,  8.],
       [ 0.,  0.,  0., -1., -12.]])
```

et pour les éléments propres :

```
def propre(n,a,b):
    A = matrice(n,a,b)
    return np.linalg.eig(A)
```

c) on doit le faire à la main mais on peut commencer par regarder ce qu'on obtient :

```
>>> matrice(2,1,0)
array([[ 0.,  0.,  0.],
       [-2., -1.,  0.],
       [ 0., -1., -2.]])
>>> propre(2,1,0)[1]
array([[ 0.          ,  0.          ,  0.40824829],
       [ 0.          ,  0.70710678, -0.81649658],
       [ 1.          , -0.70710678,  0.40824829]])
```

cela donne des valeurs propres $-2, -1, 0$ et les vecteurs propres associés (en colonne). On fait alors les calculs pendant l'oral...

On fait de même pour la seconde matrice :

```
>>> matrice(2,2,1)
array([[ 0.,  2.,  0.],
       [-2., -3.,  4.],
       [ 0., -1., -6.]])
>>> propre(2,2,1)[0]
array([-2., -3., -4.])
>>> propre(2,1,0)[1]
array([[ -0.69631062,  0.53452248,  0.40824829],
       [ 0.69631062, -0.80178373, -0.81649658],
       [-0.17407766,  0.26726124,  0.40824829]])
```

d) Le meilleur moyen est de résoudre l'équation différentielle associée, par exemple sur l'intervalle $] \max(a, b), +\infty[$ et de chercher λ pour que l'équation différentielle admette des solutions polynomiales. En effectuant les calculs, on trouve au moins $n+1$ valeurs distinctes et des polynômes associés (de la forme $(X-a)^k(X-b)^{n-k}$ pour la valeur propre $-((n-k)a+kb)$) - voir exercice correspondant sur la feuille d'exercices « réduction 1 ».

Exercice 4

a) Chaque ensemble est borné et fermé (une suite de matrices de l'un de ces ensembles qui converge a sa limite dans l'ensemble (coefficients par coefficients). Les ensembles sont donc compacts - en dimension finie - et la fonction déterminant est continue sur $M_n(\mathbb{R})$ donc elle est bornée et atteint ses bornes sur H_n et sur \widehat{H}_n .

b) On programme la fonction demandée :

i) on commence avec les matrices à coefficients dans $\{-1, 1\}$.

```
import numpy as np
import numpy.random as rnd

def H(n):
    return rnd.randint(0,2,(n,n))*2-1

def MaxiDetAlea(n):
    maxi, Mat = 0, 0
    for i in range(1000):
        M = H(n)
        d = np.linalg.det(M)
        if d > maxi:
            maxi = d
            Mat = M
    return maxi, M
```

et on essaie

```
>>> MaxiDetAlea(4)
(np.float64(15.999999999999998),
 array([[ 1,  1,  1,  1],
        [-1, -1,  1,  1],
        [-1,  1,  1, -1],
        [ 1,  1,  1,  1]]))
```

ou

```
>>> MaxiDetAlea(6)
(np.float64(127.99999999999997),
 array([[ 1, -1, -1, -1, -1,  1],
        [-1,  1, -1,  1, -1,  1],
        [ 1,  1, -1, -1, -1, -1],
        [ 1, -1, -1,  1, -1,  1],
        [-1, -1,  1,  1,  1,  1],
        [ 1,  1, -1, -1, -1,  1]]))
```

Pour $n = 5$, on obtient parfois 32 parfois 48... pour $n = 7$, c'est pire.

ii) Pour \widehat{M}_n , on a simplement à changer le choix des coefficients avec `rnd.random((n,n))*2-1`. Les résultats sont beaucoup moins concluants.

iii) on détermine toutes les matrices (n^2 coefficients qu'on reformate en matrice (n, n)) :

```
def M(n):
    maxi, Mat = 0, 0
    for m in product([-1, 1], repeat=n*n):
        mm = np.array(m).reshape((n, n))
        d = np.linalg.det(mm)
        if d > maxi:
            maxi = d
            Mat = mm
    return maxi, Mat
```

On trouve 4 pour $n = 3$, 16 pour $n = 4$ et 48 pour $n = 5$ (après les calculs deviennent très longs.. on a 2^{36} matrices à tester)

- c) i) B est symétriques réelles. Si X est un vecteur propre pour la valeur propre λ de B , alors $X^T B X = (A X)^T (A X) = \|A X\|_2^2 \geq 0$ et $X^T B X = \lambda \|X\|_2^2$ donc $\lambda \geq 0$.

On note $\lambda_1, \dots, \lambda_n$ les valeurs propres de B . On a $\det B = \prod_{k=1}^n \lambda_k$ et $\operatorname{tr}(B) = \sum_{k=1}^n \lambda_k$. L'inégalité est vraie si l'une des valeurs propres est nulle. Sinon, en utilisant la concavité de la fonction logarithme sur \mathbb{R}_+^* , on a

$$\ln \left(\frac{1}{n} \sum_{k=1}^n \lambda_k \right) \geq \frac{1}{n} \sum_{k=1}^n \ln \lambda_k$$

Par composition par la fonction exponentielle qui est croissante, on obtient

$$\frac{1}{n} \operatorname{tr} B \geq (\det B)^{1/n} \text{ soit } \det B \leq \left(\frac{\operatorname{tr} B}{n} \right)^n$$

- ii) Chaque coefficient diagonal de B vaut n (la norme au carré d'une colonne de A). On a $(\det A^T A) = (\det A)^2 = \det B \leq n^n$ et $|\det A| \leq n^{n/2}$.
- d) Si $\frac{1}{\sqrt{n}} A$ est orthogonale alors $|\det A| \cdot \frac{1}{n^{n/2}} = 1$ donc $|\det A| = n^{n/2}$. Réciproquement, si le déterminant vaut $n^{n/2}$, alors on est dans le cas d'égalité de l'inégalité de convexité; elle n'a lieu que si toutes les valeurs propres sont égales. Ainsi B est une matrice scalaire (elle est diagonalisable avec une seule valeur propre) et plus précisément $B = nI_n$. En posant $C = \frac{1}{\sqrt{n}} A$, on a $C^T C = \frac{1}{n} B = I_n$.

Exercice 5

- a) i) le plus simple puisque la question est posée ainsi est de réaliser un crible d'Ératosthène (mais une méthode plus basique passerait également) :

```
def premier(n):
    L = [0,0]+[1]*(n-1)
    for j in range(2,n+1):
        if L[j]==1: # le nombre est premier - on met ses multiples à 0
            for k in range(2,1+n//j):
                L[k*j]=0
    return L
```

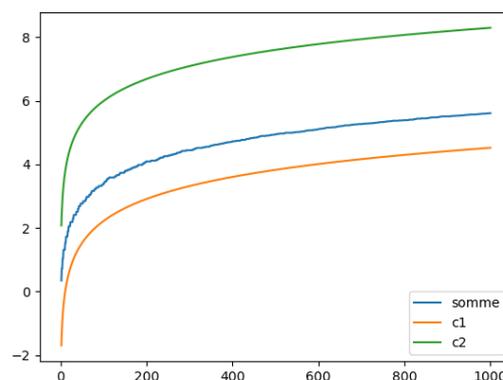
```
>>> premier(20)
[0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0]
```

- ii) on ne cherche pas à optimiser (on pourrait calculer la somme de proche en proche afin de ne pas refaire tous les calculs)

```
def somme(n):
    S = 0
    P = premier(n)
    for k in range(2,n+1):
        if P[k]:
            S += np.log(k)/k
    return S
```

ce qui donne

```
N = 1000
A = list(range(2,N+1))
B = [somme(k) for k in A]
C = [np.log(k)-np.log(4)-1 for k in A]
D = [np.log(k)+np.log(4) for k in A]
plt.plot(A,B,A,C,A,D)
plt.legend(["somme", "c1", "c2"])
plt.show()
```



On peut conjecturer l'encadrement de la somme par les deux valeurs proposées.

- b) i) Entre 1 et n , il y a $\left\lfloor \frac{n}{p^k} \right\rfloor$ multiples de p^k . Parmi ces multiples, il faut retirer les multiples de p^{k+1} . Il y a donc exactement $\left\lfloor \frac{n}{p^k} \right\rfloor - \left\lfloor \frac{n}{p^{k+1}} \right\rfloor$ entiers entre 1 et n qui ont une valuation p -adique de k .
- ii) On remarque tout d'abord que la série proposée est bien convergente puisque les termes sont nuls à partir d'un certain rang.

Si on note n_k le cardinal précédent, on a (chaque somme qui apparaît est finie) :

$$\begin{aligned} v_p(n!) &= \sum_{k=0}^{+\infty} k n_k = \sum_{k=0}^{+\infty} k \left(\left\lfloor \frac{n}{p^k} \right\rfloor - \left\lfloor \frac{n}{p^{k+1}} \right\rfloor \right) \\ &= \sum_{k=0}^{+\infty} k \left\lfloor \frac{n}{p^k} \right\rfloor - \sum_{k=0}^{+\infty} k \left\lfloor \frac{n}{p^{k+1}} \right\rfloor \\ &= \sum_{k=0}^{+\infty} k \left\lfloor \frac{n}{p^k} \right\rfloor - \sum_{k=1}^{+\infty} (k-1) \left\lfloor \frac{n}{p^k} \right\rfloor = \sum_{k=1}^{+\infty} k \left\lfloor \frac{n}{p^k} \right\rfloor - \sum_{k=1}^{+\infty} (k-1) \left\lfloor \frac{n}{p^k} \right\rfloor \\ &= \sum_{k=1}^{+\infty} \left\lfloor \frac{n}{p^k} \right\rfloor \end{aligned}$$

iii) Puisque on a $x-1 < \lfloor x \rfloor \leq x$, on obtient

$$v_p(n!) \leq \sum_{k=1}^{+\infty} \frac{n}{p^k} = \frac{n}{p} \frac{1}{1-1/p} = \frac{n}{p-1}$$

Pour la minoration, on a $v_p(n!) \geq \left\lfloor \frac{n}{p} \right\rfloor \geq \frac{n}{p} - 1$.

iv) Dans les entiers de 1 à n , il n'y a que des facteurs premiers inférieurs à n . On a

$$n! = \prod_{p \in \mathcal{P}_n} p^{v_p(n!)} \text{ et } \ln(n!) = \sum_{p \in \mathcal{P}_n} v_p(n!) \cdot \ln(p)$$

Cela donne l'encadrement

$$n \left(\sum_{p \in \mathcal{P}_n} \frac{\ln p}{p} \right) - \sum_{p \in \mathcal{P}_n} \ln p \leq \ln(n!) \leq n \sum_{p \in \mathcal{P}_n} \frac{\ln p}{p-1}$$

c) i) On a pour tout $x \in]-1, 1[$, $\frac{1}{1-x} = \sum_{k=0}^{+\infty} x^k$ et $\frac{1}{(1-x)^2} = \sum_{k=1}^{+\infty} kx^{k-1}$. En évaluant en $1/2$, on obtient $4 = \sum_{k=1}^{+\infty} \frac{k}{2^{k-1}}$, ce qui donne $\sum_{k=1}^{+\infty} \frac{k}{2^k} = 2$.

ii) La série de terme général $\frac{\ln n}{n(n+1)}$ est convergente à termes positifs, donc la famille $\left(\frac{\ln n}{n(n+1)} \right)_{n \in \mathbb{N}^*}$ est sommable. En tant que sous-famille, la famille $\left(\frac{\ln n}{n(n+1)} \right)_{n \in \mathcal{P}}$ est sommable.

iii) On note $v_n = \ln(n!) - n \ln n + n - 1$. On a $v_1 = 0$. De plus

$$v_{k+1} - v_k = \ln(k+1) - (k+1) \ln(k+1) + (k) \ln(k) + 1 = 1 - k \ln \frac{k+1}{k} = 1 - k \ln \left(1 + \frac{1}{k} \right)$$

En utilisant le théorème des séries alternées sur $\ln(1+x) = \sum_{k=1}^{+\infty} (-1)^{k-1} \frac{x^k}{k}$ pour $x \in [0, 1[$, on obtient

$$x - \frac{x^2}{2} \leq \ln(1+x) \leq x$$

Cela donne $\frac{1}{k} - \frac{1}{2k^2} \leq \ln\left(1 + \frac{1}{k}\right) \leq \frac{1}{k}$ et $1 - \frac{1}{2k} \leq k \ln\left(1 + \frac{1}{k}\right) \leq 1$, et finalement

$$0 \leq 1 - k \ln\left(1 + \frac{1}{k}\right) \leq \frac{1}{2k}$$

On utilise enfin que $v_n - v_1 = v_n = \sum_{k=1}^{n-1} (v_{k+1} - v_k)$ ce qui donne $0 \leq v_n \leq \frac{1}{2} \sum_{k=1}^{n-1} \frac{1}{k}$. Par comparaison avec une intégrale, on vérifie que la somme est inférieure à $\ln n$

remarque : il est fortement possible qu'on puisse trouver un peu plus simple

iv) Question difficile (on peut même dire infaisable)...

• si $m \in \mathbb{N}$, alors $w_m = \prod_{\substack{p \in \mathcal{P} \\ m+1 < p \leq 2m+1}} p \leq \binom{2m+1}{m} = \binom{2m+1}{m+1}$. En effet

$$\binom{2m+1}{m} = \frac{1}{m!} (2m+1)(2m) \dots (m+1)$$

le produit $(2m+1)(2m)\dots(m+1)$ peut se factoriser par les différents nombres premiers qui sont entre $m+1$ et $2m+1$, ce qui donne

$$m! \binom{2m+1}{m} = K \cdot w_n$$

où K est un entier. Or $m!$ est premier avec le produit des nombres premiers entre $m+1$ et $2m+1$ (aucun facteur premier en commun). On en déduit bien que w_m divise $\binom{2m+1}{m}$ donc est inférieur.

- On a $2^{2m+1} = (1+1)^{2m+1} = \sum_{k=0}^{2m+1} \binom{2m+1}{k} = \binom{2m+1}{m} + \binom{2m+1}{m+1} + \dots$ (termes positifs). On a donc $2 \binom{2m+1}{m} \leq 2^{2m+1}$, ce qui donne $\binom{2m+1}{m} \leq 4^m$.

- on montre enfin le résultat demandé par récurrence sur n . On le vérifie aisément pour les premières valeurs de n . Si la majoration est vraie jusqu'au rang n . Si $n+1$ n'est pas premier, alors $\prod_{p \in \mathcal{P}_{n+1}} p = \prod_{p \in \mathcal{P}_n} p \leq 4^n \leq 4^{n+1}$. Si $n+1 = 2m+1$ est premier, alors

$$\prod_{p \in \mathcal{P}_{2m+1}} p = \prod_{p \in \mathcal{P}_{m+1}} p \cdot w_m \leq 4^{m+1} 4^m = 4^{2m+1}$$

On reprend la première inégalité - on note dans la suite $s_n = \sum_{p \in \mathcal{P}_n} \frac{\ln p}{p}$:

$$n s_n - \sum_{p \in \mathcal{P}_n} \ln p \leq \ln n! \leq n \ln n - n + 1 + \ln n$$

soit

$$n s_n \leq \sum_{p \in \mathcal{P}_n} \ln p + n \ln n - n + 1 + \ln n$$

Or $\sum_{p \in \mathcal{P}_n} \ln p \leq \ln(4^n) = n \ln 4$ d'après le début de la question. On a donc

$$s_n \leq \ln 4 + \ln n \frac{1}{n} (\ln n - (n-1))$$

Puisque $\ln(1+u) \leq u$, $\ln n \leq n-1$ et enfin $s_n \leq \ln n + \ln 4$.

Pour l'autre inégalité... on peut écrire $\sum_{p \in \mathcal{P}_n} \frac{\ln p}{p-1} = s_n + \sum_{p \in \mathcal{P}_n} \frac{\ln p}{p-1} - \frac{\ln p}{p}$ et utiliser $\ln n! \geq n \ln n - n + 1 \dots$ à finir mais de toute façon la demi-heure est plus que passée.

Exercice 6

a)

i)

```
def pgcd(a, b):
    if b==0:
        return a
    return pgcd(b, a%b)
```

ii) On a $H_k = H_{k-1} + \frac{1}{k} = \frac{A_{k-1}}{B_{k-1}} + \frac{1}{k} = \frac{kA_{k-1} + B_{k-1}}{kB_{k-1}}$. Si on note $d = (kA_{k-1} + B_{k-1}) \wedge (kB_{k-1})$, alors

$$A_k = \frac{kA_{k-1} + B_{k-1}}{d} \text{ et } B_k = \frac{kB_{k-1}}{d}.$$

iii)

```
def AB(n):
    a, b = 1, 1
    for k in range(2, n+1):
        num = k*a+b
        den = k*b
        d = pgcd(num, den)
        a, b = num//d, den//d
    return a, b

def A(n):
    return (AB(n))[0]
```

On peut évidemment directement retourner a dans la première fonction (mais pour tester c'est pas mal ainsi). On obtient $A_{16} = 2436559$

iv)

```
def test():
    for p in sympy.primerange(500):
        if A(p-1)%(p*p)==0:
            print(p, 'OK')
        else:
            print(p, 'Non !!!')
```

La propriété est vérifiée mais seulement à partir de $p = 5!!!$ (pas pour $p = 2$ ou $p = 3$)

b) si p n'est pas premier, alors il existe q et r dans $\llbracket 1; p-1 \rrbracket$ tels que $qr = p$. On a alors $(p-1)!$ qui a $qr = p$ en facteurs et $(p-1)! \equiv 0[p]$. Si p est premier, tout élément de $(\mathbb{Z}/p\mathbb{Z}^*, \times)$ admet un symétrique. Les éléments qui sont leur propre symétrique vérifient $x^2 = 1$ dans le corps $\mathbb{Z}/p\mathbb{Z}$ donc sont 1 et -1 . En regroupant les facteurs symétriques deux à deux dans $\overline{(p-1)!}$, sauf $\overline{1}$ et $\overline{-1}$, il reste $\overline{1} \times \overline{-1}$ et le produit vaut finalement $\overline{-1}$. On a bien $(p-1)! \equiv -1[p]$.

c) On a $\frac{1}{k(p-k)} = \frac{1}{p} \left(\frac{1}{k} + \frac{1}{p-k} \right)$. Ainsi

$$\sum_{k=1}^{p-1} a_k = (p-1)! \frac{1}{p} \sum_{k=1}^{p-1} \left(\frac{1}{k} + \frac{1}{p-k} \right) = \frac{2(p-1)!}{p} H_{p-1}.$$

Cela donne $p \sum_{k=1}^{p-1} a_k = 2(p-1)! H_{p-1}$ d'où $p B_{p-1} \sum_{k=1}^{p-1} a_k = 2(p-1)! A_{p-1}$. Puisque p est premier différent de 2, p est premier avec 2 et $(p-1)!$ donc p divise A_{p-1} .

d) On a enfin $k(p-k)a_k = (p-1)!$. Puisque $p-k \equiv -k[p]$, on a $k(p-k)a_k \equiv -k^2 a_k[p]$. En utilisant le théorème de Wilson, $p^2 a_k \equiv 1[p]$ pour tout $k \in \llbracket 1; p-1 \rrbracket$. L'application $x \mapsto x^{-1}$ est une permutation de $\mathbb{Z}/p\mathbb{Z}^*$. On a ainsi, dans $\mathbb{Z}/p\mathbb{Z}$,

$$\sum_{k=1}^{p-1} \overline{a_k} = \sum_{k=1}^{p-1} (\overline{k-1})^2 = \sum_{j=1}^{p-1} \overline{j^{-2}} = \overline{\sum_{j=1}^{p-1} j^2}$$

Or $\sum_{j=1}^{p-1} j^2 = \frac{p(p-1)(2p-1)}{6}$. Si p est premier différent de 2 et 3 alors c'est $(p-1)(2p-1)$ qui est divisible par 2×3 et ainsi la somme est divisible par p . On a bien $\sum_{k=1}^{p-1} a_k \equiv 0[p]$ et p divise $\sum_{k=1}^{p-1} a_k$. On peut écrire cette somme $\sum_{k=1}^{p-1} a_k = p \cdot q$. Alors la question précédente donne $p^2 \cdot q = 2(p-1)!H_{p-1}$. Le raisonnement précédent donne $p^2 | A_{p-1}$.

Exercice 8

a) se fait sans problème

b)

```

from numpy.polynomial import Polynomial
import scipy.integrate as integ
import numpy as np
import matplotlib.pyplot as plt

X = Polynomial([0,1])

def phi(P,Q):
    def f(t):
        return np.abs(t)*P(t)*Q(t)

    f = lambda t:np.abs(t)*P(t)*Q(t)

    I,eps = integ.quad(f,-1,1)
    return I

def norme(P):
    return np.sqrt(phi(P,P))

```

c) L'existence vient de la méthode d'orthonormalisation de Gram-Schmidt de la base canonique : elle donne notamment le degré k pour P_k : on a $\text{Vect}(P_0, \dots, P_{k-1}) = \mathbb{R}_{k-1}[X]$ et $\text{Vect}(P_0, \dots, P_k) = \mathbb{R}_k[X]$ avec notamment $P_k = \lambda_k (X^k - \sum_{i=0}^{k-1} \langle X^k, P_i, P \rangle_i)$ où λ_k est l'inverse de la norme du polynôme qui suit la constante. L'unicité vient du choix positif de la constante : P_k est dans l'orthogonal de $\mathbb{R}_{k-1}[X]$ dans $\mathbb{R}_k[X]$ - c'est une droite, il y a deux polynômes unitaires sur cette droite, qui sont opposés. Seul l'un a un coefficient dominant positif.

d)

```

def P(n):

    P0 = Polynomial([1])
    P0 = P0/norme(P0)
    L = [P0]
    for k in range(1,n+1):
        A = X**k
        for i in range(k):
            A = A - phi(X**k,L[i])*L[i]
        A = A/norme(A)
        L.append(A)
    return L[n]

x = np.linspace(-1,1,201)
for i in range(5):
    y = P(i)(x)
    plt.plot(x,y,label="P_"+str(i))
plt.legend()
plt.show()

```

e) Raisonnement habituel pour ce genre de polynômes orthonormalisés. On note x_1, \dots, x_p les racines de multiplicité impaire de P_n dans $] -1, 1[$. Si p est différent de n , on considère $Q = \prod_{i=1}^p (X - x_i)$ ($Q = 1$ s'il n'y a pas de telles racines). Le polynôme $Q.P_n$ garde un signe constant sur $] -1, 1[$ et si $p < n$ alors $\langle Q, P_n \rangle = 0$ ce qui est impossible avec la considération de signe. On en déduit que $p = n$ et que P_n , qui est de degré n , admet exactement n racines distinctes dans $] -1, 1[$ (il est donc scindé à racines simples).

f) C'est plus compliqué... on le fait en plusieurs étapes.

- comme toute suite de polynômes orthogonaux obtenus ainsi, on a une relation de récurrence simple. Le polynôme XP_n de degré $n+1$ se décompose $XP_n = \sum_{i=0}^{n+1} a_i P_i$ avec $a_i = \langle XP_n, P_i \rangle = \langle P_n, XP_i \rangle$. Ce produit scalaire est nul si $\deg(XP_i) < n$ donc si $i < n-1$. Il vient une relation

$$XP_n = \alpha_n P_{n+1} + \beta_n P_n + \gamma_n P_{n-1}.$$

- la constante α_n est strictement positive en examinant les termes de plus haut degré. On a $\gamma_n = \langle XP_n, P_{n-1} \rangle = \langle P_n, XP_{n-1} \rangle$. On décompose $XP_{n-1} = k_n P_n + \sum_{i=0}^{n-1} \lambda_i P_i$. De nouveau (termes de plus haut degré) $k_n > 0$. Par linéarité (et avec l'orthogonalité des polynômes), on a $\gamma_n = k_n \langle P_n, P_n \rangle > 0$.
- on effectue une récurrence (l'initialisation se fait bien). On note $x_1 < \dots < x_{n-1}$ les racines de P_{n-1} et $y_0 < y_1 < \dots < y_{n-1}$ celles de P_n . L'hypothèse de récurrence donne

$$y_0 < x_1 < y_1 < x_2 < \dots < x_{n-1} < y_{n-1}$$

Puisque les racines de P_{n-1} sont simples et que $\lim_{x \rightarrow +\infty} P_{n-1}(x) = +\infty$, on en déduit que $P_{n-1}(y_{n-1}) > 0$, $P_{n-1}(y_{n-2}) < 0$... (alternance des signes) En évaluant la relation de récurrence en y_k , on a $P_{n+1}(y_k) = -\frac{\gamma_n}{\alpha_n} P_{n-1}(y_k)$. On a de nouveau alternance de signes (mais dans l'autre sens par rapport à P_{n-1}). Cela donne $n-1$ racines qui s'intercalent entre les racines de P_n . Puisque $P_{n+1}(y_{n-1}) < 0$ et que $\lim_{x \rightarrow +\infty} P_{n+1}(x) = +\infty$, on a une nouvelle racine strictement supérieure à y_{n-1} . De même il existe une racine pour P_{n+1} strictement inférieure à y_0 . On a obtenu les $n+1$ racines de P_{n+1} avec celles de P_n qui s'intercalent.

Exercice 9

a)

```
def premier(n):
    i = 2
    while i*i<=n:
        if n%i == 0: return False
        i += 1
    return True
```

b)

```
from math import sqrt

def carre(n):
    a = int(sqrt(n)+0.01)
    # au cas où... avec les erreurs d'arrondis
    return(n==a**2)

def test():
    T = []
    for p in range(3,1000):
        if premier(p) and carre(p-2):
            T.append(p)
    return T
```

ce qui donne

```
>>> test()
[3, 11, 83, 227, 443]
```

c)

```
def solutions(p,n):
    L = []
    i = 0
    x = 1
    while i<n:
        if ((x**2-1)%p==0) and carre((x**2-1)//p):
            y = int(sqrt((x**2-1)//p)+0.01)
            L.append([x,y])
            i+=1
        x += 1
    return L
```

On obtient

```
>>> solutions(3,10)
[[1, 0], [2, 1], [7, 4], [26, 15], [97, 56], [362, 209], [1351, 780], [5042,
2911], [18817,10864], [70226, 40545]]
```

pour les entiers trouvés dans la question précédente :

```
def q3():
    L = test()
    for p in L:
        print(p, " : ", solutions(p,3))
```

```
>>> q3()
3 : [[1, 0], [2, 1], [7, 4]]
11 : [[1, 0], [10, 3], [199, 60]]
83 : [[1, 0], [82, 9], [13447, 1476]]
227 : [[1, 0], [226, 15], [102151, 6780]]
443 : [[1, 0], [442, 21], [390727, 18564]]
```

- d) i) On a $z_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, $z_1 = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$ et $z_2 = \begin{pmatrix} 7 \\ 4 \end{pmatrix}$. On cherche une matrice $M = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$. La première relation donne $a = 2$ et $c = 1$. La seconde donne $b = 3$ et $d = 2$. On trouve ainsi $M = \begin{pmatrix} 2 & 3 \\ 1 & 2 \end{pmatrix}$.

```
ii) def suite(n):
    L = [[1, 0]]
    x, y = 1, 0
    for _ in range(n-1):
        x, y = 2*x+3*y, x+2*y
        L.append([x, y])
    return L
```

ce qui donne

```
>>> suite(10)
[[1, 0], [2, 1], [7, 4], [26, 15], [97, 56], [362, 209], [1351, 780], [5042, 2911], [18817, 10864], [70226, 40545]]
```

On retrouve exactement les solutions trouvées précédemment.

- iii) Cela revient à déterminer les matrices M^n . On peut le faire en diagonalisant où à l'aide d'un polynôme annulateur. Le polynôme caractéristique de M est $X^2 - 4X + 1$ et c'est le polynôme minimal (puisque M n'est pas scalaire, il est de degré au moins 2). On a donc, par division euclidienne

$$X^n = (X^2 - 4X + 1).Q + a_n X + b_n$$

en évaluant sur les racines $2 \pm \sqrt{3}$, on a le système :

$$\begin{cases} a_n(2 + \sqrt{3}) + b_n &= (2 + \sqrt{3})^n \\ a_n(2 - \sqrt{3}) + b_n &= (2 - \sqrt{3})^n \end{cases}$$

On résout, on reporte, on trouve $M^n = a_n M + b_n I_2$ et finalement $z_n = \frac{1}{2} \left(\frac{(2 + \sqrt{3})^n + (2 - \sqrt{3})^n}{\sqrt{3}} \begin{pmatrix} 2 + \sqrt{3} \\ 1 \end{pmatrix} + \frac{(2 + \sqrt{3})^n - (2 - \sqrt{3})^n}{\sqrt{3}} \begin{pmatrix} 2 - \sqrt{3} \\ 1 \end{pmatrix} \right)$

- e) Avec $x = p - 1$, on a $(p - 1)^2 - py^2 = 1$ donne $y^2 = p - 2$. D'où une solution $(p - 1, \sqrt{p - 2})$. Réciproquement si on considère une solution différente de $(1, 0)$. On a $x^2 - 1 = py^2$. Puisque p est premier, p divise $(x - 1)(x + 1)$ donc l'un des deux. Soit $x - 1 = 0$, c'est-à-dire $x = 1$, soit $x + 1 = 0$ - exclu. Sinon $x - 1$ ou $x + 1$ est supérieur ou égal à p . On a donc $x \geq p - 1$.

Exercice 10

- a) Soient T_1 et T_2 deux polynômes et $\lambda \in \mathbb{C}$. On écrit les divisions euclidiennes $T_1 P' = Q_1 P + R_1$ et $T_2 P' = Q_2 P + R_2$ avec R_1 et R_2 de degré au plus $n-1$. On a alors $(T_1 + \lambda T_2) P' = (Q_1 + \lambda Q_2) P + (R_1 + \lambda R_2)$ avec $\deg(R_1 + \lambda R_2) \leq n-1$. On a ainsi $\Phi_P(T_1 + \lambda T_2) = R_1 + \lambda R_2$ et par définition, $R_1 = \Phi_P(T_1)$ et $R_2 = \Phi_P(T_2)$. On a bien $\Phi_P(T_1 + \lambda T_2) = \Phi_P(T_1) + \lambda \Phi_P(T_2)$. Évidemment, Φ_P est à valeurs dans $\mathbb{C}_{n-1}[X]$ et Φ_P est un endomorphisme de $\mathbb{C}_{n-1}[X]$.

```
b) from numpy.polynomial import Polynomial
import numpy as np

def det_matrice(P):
    n = P.degree()
    X = Polynomial([0, 1])
    Pprime = P.deriv()
    Xn = 1
    A = np.zeros((n, n))
    for i in range(n):
        Q = (Xn*Pprime) % P
        A[:, len(Q), i] = Q.coef
        Xn = Xn*X # permet d'avoir les puissances de X
    return np.linalg.det(A)
```

- c) Différentes options : effectuer la division euclidienne en évaluant en les différents λ_i mais cela semble compliqué; on peut chercher à diagonaliser en résolvant $TP' \% P = \lambda T$, c'est à dire trouver T tel que $T(P' - \lambda)$ soit multiple de P (pour des raisons de degré, on se rend compte que l'une des racines de P doit être racine de $P' - \lambda$, ce qui donne $\lambda = P'(\lambda_i)$ et on trouve alors un polynôme propre pour chacune des n valeurs propres $P'(\lambda_i)$. On peut enfin effectuer les calculs dans la base la plus raisonnable qui soit vu le problème, c'est-à-dire la base de Lagrange.

Soit (L_1, \dots, L_n) la base de Lagrange associée aux racines $\lambda_1, \dots, \lambda_n$. Pour $i \in \llbracket 1; n \rrbracket$, on a $L_i P' = Q_i P + R_i$ où $R_i = \sum_{j=1}^n \alpha_j^{(i)} L_j$.

En évaluant en λ_k , on obtient $L_i(\lambda_k) P'(\lambda_k) = \alpha_k^{(i)}$. Finalement les $\alpha_j^{(i)}$ sont tous nuls sauf $\alpha_i^{(i)} = P'(\lambda_i)$. On a finalement obtenu $\Phi_P(L_i) = P'(\lambda_i) L_i$ (on retrouve les résultats envisagés au dessus). L'application Φ_P est diagonalisable et son déterminant est $d = \prod_{k=1}^n P'(\lambda_k)$.

- d) Si P est à racines simples alors Φ_P est inversible. Supposons que P admette une racine double λ : on a $P = (X - \lambda)^2 Q_1$ et $P' = (X - \lambda) Q_2$. On cherche T tel que $\Phi_P(T) = 0$ soit T de degré au plus $n-1$ et tel que TP' soit divisible par P . On choisit $T = (X - \lambda) Q_1$, il est de degré $n-1$ et $TP' = (X - \lambda)^2 Q_1 Q_2 = P Q_2$ donc $\Phi_P(T) = 0$ sans que T soit nul. L'application Φ_P est inversible si et seulement si P est à racines simples.

Exercice 11

```

a) from numpy.polynomial import Polynomial
import numpy as np

def Mat(n):
    A = np.zeros((n+1, n+1))
    X = Polynomial([0, 1])
    for i in range(n+1):
        P = (1-X)**i*(1+X)**(n-i)
        A[:, i]=P.coef
    return A

for n in range(2, 10):
    M = Mat(n)
    print(n, np.dot(M, M))

```

Les calculs semblent montrer que $M_n^2 = 2^n I_n$.

- b) On peut évidemment effectuer les calculs brutalement, ça passe sans soucis : le terme en ligne i de $M_n U_n(x_0)$ (avec i allant de 0 à n) est

$$\sum_{j=0}^n \beta_{n,i,j} x_0^j = P_{n,i}(x_0) = (1-x_0)^i (1+x_0)^{n-i} = (1+x_0)^n \left(\frac{1-x_0}{1+x_0} \right)^i = (1+x_0)^n y_0^i.$$

On a donc bien la relation demandée. On pose alors

$$z_0 = \frac{1-y_0}{1+y_0} = \frac{1+x_0-1+x_0}{1+x_0+1-x_0} = x_0,$$

et $M_n^2 U_n(x_0) = (1+x_0)^n M_n U_n(y_0) = (1+x_0)^n (1+y_0)^n U_n(z_0) = (1+x_0)^n \left(\frac{2}{1+x_0} \right)^n U_n(x_0) = 2^n U_n(x_0)$. Ainsi $M_n^2 X$ et $2^n I_{n+1} X$ sont égaux pour tout vecteur X du type $U_n(x_0)$. En prenant $n+1$ valeurs 2 à 2 distinctes x_0, \dots, x_n , on obtient une base de vecteurs (la matrice de cette famille est une matrice de Vandermonde). On en déduit que M_n^2 et $2^n I_{n+1}$ sont égales. Le polynôme $X^2 - 2^n$ est annulateur de M_n , scindé à racines simples donc M_n est diagonalisable avec des valeurs propres égales à $\pm 2^{n/2}$.

- c) On a $y_i - y_j = \frac{1+j-i-ij-1-i+j+ij}{(1+i)(1+j)} \frac{2(j-i)}{(1+i)(1+j)} = -\frac{x_i - x_j}{(1+i)(1+j)}$. On calcule alors le déterminant de Vandermonde (la taille est $n+1$, le nombre de terme est $\frac{(n+1)n}{2}$):

$$\det V_n(y) = \prod_{i>j} (y_i - y_j) = \prod_{i>j} \left(-\frac{x_i - x_j}{(1+i)(1+j)} \right) = (-2)^{(n(n+1)/2)} \frac{1}{\prod_{i>j} (1+i)(1+j)},$$

avec

$$\prod_{i>j} (1+i)(1+j) = \prod_{i=1}^n \prod_{j=0}^{i-1} (1+i)(1+j) = \prod_{i=1}^n ((1+i)^i i!)$$

qui n'est pas immédiat. Pour des raisons de symétrie, on a $\prod_{n \geq i > j \geq 0} (1+i)(1+j) = \prod_{n \geq j > i \geq 0} (1+i)(1+j) = P$. On calcule alors

$$\prod_{i=0}^n \prod_{j=0}^n (1+i)(1+j) = \prod_{i=0}^n ((1+i)^{n+1} (n+1)!) = (n+1)!^{n+1} (n+1)!^{n+1} = ((n+1)!)^{2n+2},$$

et ce terme est aussi $P^2 \prod_{i=0}^n (1+i)(1+i) = (n+1)!^2 P^2$. On a $P^2 = ((n+1)!)^{2n}$, et puisque $P > 0$, on en déduit que $P = ((n+1)!)^n$. Cela donne la relation.

- d) On calcule $M_n V_n(x) = W_n$. La colonne i (entre 0 et n) de W_n est $(1+x_i)^n = (1+i)^n$ fois la colonne i de $V_n(y)$. On en déduit alors

$$\det(M_n V_n(x)) = \det(M_n) \det(V_n(x)) = 1^n 2^n \dots (n+1)^n \det(V_n(y)) = ((n+1)!)^n \det(V_n(y)),$$

et avec la question précédente (puisque $V_n(x)$ est de déterminant non nul), on obtient $\det M_n = (-2)^{n(n+1)/2}$. Les deux réels $2^{n/2}$ et $-2^{n/2}$ sont valeurs propres (et la parité du nombre de $-2^{n/2}$ est celle de $(-1)^{n(n+1)/2}$).

Exercice 12

- a) On choisit Ω une matrice de rotation : $\Omega = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}$. On effectue le produit ${}^t\Omega A \Omega$. L'égalité des éléments diagonaux donne la relation

$$a \cos^2 \theta + (b+c) \sin \theta \cos \theta + d \sin^2 \theta = a \sin^2 \theta - (b+c) \sin \theta \cos \theta + d \cos^2 \theta,$$

ce qui peut s'écrire $a \cos(2\theta) + (b+c) \sin(2\theta) - d \cos(2\theta) = 0$ ou encore $(a-d) \cos(2\theta) + (b+c) \sin(2\theta) = 0$.

- lorsque $a = d$ la matrice a déjà sa diagonale constante, on peut prendre $\theta = 0$, c'est-à-dire $\Omega = I_2$.
- lorsque $a \neq d$, on obtient $\cotan(2\theta) = \frac{b+c}{d-a}$ (si on préfère \tan , on obtient en général, $\theta = \frac{1}{2} \arctan\left(\frac{d-a}{b+c}\right)$ sauf si $b+c=0$).

b)

```
import numpy as np

A = np.array([[0, 2], [2, 1]])
theta = np.arctan(1/4)/2
Omega = np.array([[np.cos(theta), -np.sin(theta)], [np.sin(theta), np.cos(theta)]])
B = np.dot(Omega.T, np.dot(A, Omega))
# ou (Omega.T@A@Omega) dans les dernières versions de numpy
```

donne

```
>>> B
array([[ 0.5,  2.06155281],
       [ 2.06155281,  0.5]])
```

- c) On note $\varphi : \Omega \mapsto {}^t\Omega A \Omega$. Cette application est continue sur $M_n(\mathbb{R})$ (produit, transposition). Puisque $O_n(\mathbb{R})$ est compact, $\Gamma = \varphi(O_n(\mathbb{R}))$ est compact.
- d) On commence par montrer que f est continue sur $M_n(\mathbb{R})$. Pour deux réels $\sup(x, y) = \frac{x+y+|x-y|}{2}$ si bien que \sup est continue sur \mathbb{R}^2 . Par récurrence, en utilisant le fait que $\sup(x_1, \dots, x_{n+1}) = \sup(\sup(x_1, \dots, x_n), x_{n+1})$, on montre que $(x_1, \dots, x_n) \mapsto \sup(x_1, \dots, x_n)$ est continue sur \mathbb{R}^n . L'application $M \mapsto (|M_{ii} - M_{jj}|)_{i,j \in \llbracket 1;n \rrbracket}$ est continue sur $M_n(\mathbb{R})$ dans \mathbb{R}^{n^2} . Finalement par composition f est continue sur $M_n(\mathbb{R})$. Puisque Γ est compact, f admet un minimum sur Γ et celui ci est atteint sur Γ .
- e) Soit M une matrice en laquelle le minimum de f est atteint. On suppose que $f(M) > 0$. Il existe alors $i \neq j$ tel que $M_{ii} \neq M_{jj}$. On considère alors Ω la matrice de $O_n(\mathbb{R})$ qui correspond à la matrice dans la base canonique (e_1, \dots, e_n) de \mathbb{R}^n de la rotation d'angle θ dans le plan (e_i, e_j) (et qui laisse les autres vecteurs de la base). Les calculs de ${}^t\Omega M \Omega$ se font comme dans la question 1. Pour s'en convaincre, on peut effectuer les calculs dans la base (e_i, e_j, \dots) . On a

$$\Omega' = \left(\begin{array}{c|c} R_\theta & (0) \\ \hline (0) & I_{n-2} \end{array} \right), A' = \left(\begin{array}{c|c} A & B \\ \hline C & D \end{array} \right), \text{ et } {}^t\Omega' A' \Omega' = \left(\begin{array}{c|c} {}^tR_\theta A \theta & {}^tR_\theta B \\ \hline C R_\theta & D \end{array} \right)$$

En revenant à la base de départ, on permute les éléments diagonaux mais sans changer leurs valeurs (en revanche pour les autres coefficients, c'est autre chose). En choisissant θ comme dans la question 1, on peut trouver θ de sorte que les termes de $M' = {}^t\Omega M \Omega$ soient égaux.

On croit avoir terminé... mais non... on a est parti de M avec une certaine diagonale, on l'a transformé en M' qui permet de changer deux éléments diagonaux (et les égaux) mais on a changé plein d'autres termes $|M_{kk} - M_{ii}|$. On va légèrement changer le départ : on ordonne les éléments diagonaux de M : $\alpha_1 \leq \alpha_2 \leq \dots \leq \alpha_n$ avec $\alpha_1 < \alpha_n$. On effectue alors le travail précédent. On a fait augmenter strictement α_1 et diminuer strictement α_n . Il se pourrait qu'il y ait plusieurs fois la valeur α_1 et/ou α_n mais un nombre fini de fois. On réitère alors cela (moins de $n/2$ fois afin d'avoir des termes diagonaux tous strictement entre α_1 et α_n). On obtient ainsi une nouvelle matrice N de Γ telle que $f(N) < f(M)$ et enfin une contradiction.

Exercice 13

a)

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize as resol

def R(n, x):
    S = 0
    for k in range(n+1):
        S += 1/(x-k)
    return S

def trace(n):
    X = np.linspace(-4, 8, num=1001)
    Y = [R(n,x) for x in X]
    plt.plot(X, Y)
    plt.ylim = [-5, 5]
    plt.grid()
    plt.show()
```

b) i) Puisque $R'_n(x) = -\sum_{i=0}^n \frac{1}{(x-i)^2} < 0$ sur $]k, k+1[$, R_n est strictement décroissante et comme $\lim_{x \rightarrow k^+} R_n(x) = +\infty$, et $\lim_{x \rightarrow (k+1)^-} R_n(x) = -\infty$, elle admet un unique zéro $a_{n,k}$ dans cet intervalle.

ii) On procède par dichotomie pour trouver une valeur approchée de $a_{n,k}$:

```
def A(n, k, eps):
    """ utilisation de dichotomie """
    g = k
    d = k + 1
    m = (g + d) / 2
    while (d-g) > eps:
        if R(n, m) > 0:
            g = m
        else:
            d = m
        m = (g+d)/2
    return m
```

On peut également utiliser la fonction `fsolve` du module `scipy`, qui ne marche que pour une fonction d'une variable :

```
def a(n, k):
    def Rn(x):
        return R(n, x)
    a = resol.fsolve(Rn, k + 0.5)
    return a[0]
```

iii)

c) Puisque $R_n(x) - R_{n-1}(x) = \frac{1}{x-n}$, $R_n(a_{n,k}) - R_{n-1}(a_{n,k}) = \frac{1}{a_{n,k}-n} < 0$, donc $R_{n-1}(a_{n,k}) > 0$ et par conséquent $a_{n,k} \leq a_{n-1,k}$. La suite $(a_{n,k})$ est donc décroissante et minorée par k , elle converge.

d) Puisque $\frac{1}{a_{n,k}-k} = -\underbrace{\sum_{i=0}^{k-1} \frac{1}{a_{n,k}-i}}_{A_n} + \underbrace{\sum_{i=k+1}^n \frac{1}{i-a_{n,k}}}_{B_n}$ et que $a_{n,k} \rightarrow k$, $A_n \rightarrow \sum_{i=0}^{k-1} \frac{1}{k-i} = C$ et puisque $k \leq a_{n,k} \leq k+1$,

$$\sum_{i=k+1}^n \frac{1}{i-k} \leq B_n \leq \frac{1}{k+1-a_{n,k}} + \sum_{i=k+2}^n \frac{1}{i-(k+1)}.$$

on peut encadrer B_n à l'aide des nombres harmoniques : $H_{n-k} \leq B_n \leq H_{n-k-1} + \frac{1}{k+1-a_{n,k}}$, et comme $H_n = \ln n + \gamma + o(1)$, on montre que $\frac{1}{a_{n,k}-k} \sim \frac{1}{\ln n}$ d'où $a_{n,k} = k + \frac{1}{\ln n} + o\left(\frac{1}{\ln n}\right)$.

Exercice 14

- a) On utilise la relation $a \wedge b = b \wedge r$ si r est le reste de la division euclidienne de a par b avec $b \wedge r = b$ lorsque $r = 0$ (dernier reste non nul). On a le choix entre une version récursive ou non

```
def pgcd(a, b):
    while b > 0:
        a, b = b, a % b
    return a

def pgcd(a, b):
    if b == 0:
        return a
    else:
        return pgcd(b, a % b)
```

- b) • Fonction φ : on peut le faire à partir du lemme chinois comme dans le cours ($\varphi(n)$ est le nombre d'inversibles dans $\mathbb{Z}/n\mathbb{Z}$). L'application

$$\theta : \begin{cases} \mathbb{Z}/(mn)\mathbb{Z} & \rightarrow \mathbb{Z}/n\mathbb{Z} \times \mathbb{Z}/m\mathbb{Z} \\ \bar{x}^{mn} & \mapsto (\bar{x}^n, \bar{x}^m) \end{cases}$$

où \bar{x}^p désigne la classe de x dans $\mathbb{Z}/p\mathbb{Z}$, est un **isomorphisme** d'anneaux (le second est l'anneau produit avec les lois composante par composante) : elle est bien définie, on a $\theta(x+y) = \theta(x) + \theta(y)$ et $\theta(xy) = \theta(x)\theta(y)$ ainsi que $\theta(1) = (1, 1)$ unité de l'anneau produit. Par le lemme de Gauss, l'application est injective (si $m \wedge n = 1$) et donc bijective avec les cardinaux. Si $x \in \llbracket 1; mn \rrbracket$, alors x est premier avec mn si et seulement si \bar{x} est inversible dans $\mathbb{Z}/(mn)\mathbb{Z}$. Ces inversibles sont en bijection avec les couples (y, z) où y inversible dans $\mathbb{Z}/n\mathbb{Z}$ et z dans $\mathbb{Z}/m\mathbb{Z}$. On a bien $\varphi(mn) = \varphi(m)\varphi(n)$ si $m \wedge n = 1$.

- Fonction τ : on peut le faire avec une décomposition en facteurs premiers : $m = \prod_{i=1}^k p_i^{\alpha_i}$, les diviseurs de m sont les nombres

$\prod_{i=1}^k p_i^{\beta_i}$ avec $\beta_i \in \llbracket 1; \alpha_i \rrbracket$ et il y en a $\prod_{i=1}^k (1 + \alpha_i)$. Idem avec $n = \prod_{j=1}^l q_j^{\gamma_j}$. Puisque m et n sont premiers entre eux, x divise mn si et seulement si x s'écrit

$$x = \prod_{i=1}^k p_i^{\alpha'_i} \prod_{j=1}^l q_j^{\gamma'_j},$$

avec les conditions précédentes. On obtient $\prod_{i=1}^k (1 + \alpha_i) \prod_{j=1}^l (1 + \gamma_j)$ diviseurs. Cela donne $\tau(mn) = \tau(m)\tau(n)$ si $m \wedge n = 1$.

On peut également considérer $x \in \llbracket 1; mn \rrbracket$ diviseur de mn et vérifier que $x = pq$ avec $p \wedge q = 1$ et p et q diviseurs quelconques de m et n sans passer par une décomposition en facteurs premiers. On suppose que $x|(mn)$ et on note $g = m \wedge x$. On a alors $x = gx'$ et $m = gm'$ avec $m' \wedge x' = 1$, ainsi que $x'|(m'n)$ puisque x' et m' sont premiers entre eux, x' divise n . On a bien $x = gx'$ avec g diviseur de m et d' diviseur de n . Puisque m et n sont premiers entre eux, g et x' le sont aussi. Réciproquement toutes ces décompositions conviennent. Cela redonne le résultat.

- Fonction σ . On note D_n l'ensemble des diviseurs de n . La question précédente montre que $d|(mn)$ si et seulement si $d = d_1 d_2$ avec $d_1|m$ et $d_2|n$, ou, dans l'autre sens,

$$D_{mn} = \{d_1 d_2, (d_1, d_2) \in D_m \times D_n\}.$$

Cela donne $\sigma(mn) = \sum_{(d_1, d_2) \in D_m \times D_n} d_1 d_2 = \left(\sum_{d_1 \in D_m} d_1 \right) \left(\sum_{d_2 \in D_n} d_2 \right) = \sigma(m)\sigma(n)$.

- Fonction μ : m et n n'ont aucun facteur premier commun ainsi mn a un facteur premier multiple si et seulement m ou n en a. Ainsi $\mu(mn) = 0$ si et seulement si $\mu(m) = 0$ ou $\mu(n) = 0$. Lorsque $\mu(m)$ et $\mu(n)$ sont non nuls, alors mn est produit de $\mu(m) + \mu(n)$ facteurs premiers distincts et $\mu(mn) = \mu(m)\mu(n)$. On obtient bien $\mu(mn) = \mu(m)\mu(n)$ dans toutes les situations.

- c) C'est assez simple pour les 3 premières

```
def phi(n):
    k = 0
    for x in range(1, n):
        if pgcd(x, n) == 1:
            k += 1
    return k

def tau(n):
```

```

k = 0
for x in range(1, n+1):
    if n%x==0:
        k+=1
return k

def sigma(n):
    k = 0
    for x in range(1, n+1):
        if n%x==0:
            k+=x
    return k

```

et un peu plus subtil si on ne veut pas trop calculer pour la quatrième

```

def mu(n):
    if n==1:
        return 1
    k = 1
    for x in range(2, n+1):
        # on parcourt tous les entiers
        if n%x==0:
            # x est un diviseur de n
            # c'est forcément un facteur premier car on a simplifié par les facteurs
            # premiers simples précédents et retourné 0 si on a rencontré un facteur multiple
            n = n//x
            if n%x==0:
                # facteur double, on renvoie 0
                return 0
            else:
                # facteur simple, on multiplie k par -1
                k = -k
    return k

```

- d) On a $f * e(n) = \sum_{d|n} f(d)e(n/d)$. Le seule terme restant est pour $d = n$ afin d'avoir $e(1) = 1$ et ainsi $f * e(n) = f(n)$ pour tout entier n . On a bien $f * e = f$. On a également

$$1 * \mu(n) = \mu * 1(n) = \sum_{d|n} \mu(d)1(n/d) = \sum_{d|n} \mu(d),$$

On a $(1 * \mu)(1) = 1$ (si on prend la convention que $\mu(1) = 1$). Soit n différent de 1 et sa décomposition en facteurs premiers

$n = \prod_{i=1}^k p_i^{\alpha_i}$. On considère uniquement les diviseurs de n avec des facteurs simples : $d = \prod_{i=1}^k p_i^{\varepsilon_i}$ avec $\varepsilon_i = \pm 1$. On note alors

$d_1 = \prod_{i=1}^k p_i$ et $d_2 = \prod_{i=1}^{k-1} p_i$ avec $d_2 = 1$ si $k = 1$. On a ainsi

$$\sum_{d|n} \mu(d) = \sum_{d|d_1} \mu(d) = \sum_{d|d_2} \mu(d) + \sum_{d|d_2} \mu(d.p_k) = \sum_{d|d_2} \mu(d) + \sum_{d|d_2} \mu(d)\mu(p_k) = \sum_{d|d_2} \mu(d) - \sum_{d|d_2} \mu(d) = 0.$$

Finalemnt $1 * \mu(n) = \delta_{1n} = e(n)$ pour tout $n \in \mathbb{N}^*$.

De façon plus simple, on peut utiliser le fait que la fonction est multiplicative et ainsi la calculer sur p^k où k est un entier et

p premier. En effet, si on connaît ces valeurs alors, pour n se décomposant en $n = \prod_{i=1}^d p_i^{\alpha_i}$, on a $f(n) = \prod_{i=1}^d f(p_i^{\alpha_i})$. Soit p un

nombre premier et $k \in \mathbb{N}^*$. On prend $n = p^k$. Les diviseurs de n sont les p^j avec $j \in \llbracket 0; k \rrbracket$. On a alors $(1 * \mu)(p^k) = \sum_{j=0}^k \mu(p^j) =$

$\mu(1) + \mu(p) + 0 = 1 - 1 = 0$. Ainsi $(1 * \mu)$ est nulle sur tous les p^k avec $k \neq 0$. On en déduit que pour tout $n \in \mathbb{N}^*$, $(1 * \mu)(n) = 0$.

- e) On a $h(n) = (1 * H)(n)$ d'où $h = 1 * H$ et ainsi $\mu * h = \mu * 1 * H = e * H = H$. Avec $H = \varphi$, on obtient $h(n) = \sum_{d|n} \varphi(d) = \varphi * 1$. Comme

précédemment, on calcule H en p^k . On a (voir cours), $\varphi(p^j) = p^j - p^{j-1}$ si $j \geq 1$ et $\varphi(1) = 1$. Ainsi

$$h(p^k) = \sum_{j=0}^k \varphi(p^j) = 1 + \sum_{j=1}^k (p^j - p^{j-1}) = p^k.$$

On en déduit (par décomposition en facteurs premiers et multiplicativité) que $h(n) = n$ pour tout $n \in \mathbb{N}^*$. Cela donne $\varphi * \mathbf{1} = \text{Id}$ et ainsi $\varphi = \text{Id} * \mu = \mu * \text{Id}$.

f) On calcule brutalement :

$$({}^t ADA)_{i,j} = \sum_{k=1}^n ({}^t A)_{ik} (DA)_{kj} = \sum_{k=1}^n A_{ki} F(k) A_{kj}$$

le terme A_{ki} est non nul si et seulement si $k|i$ et de même pour A_{kj} . Il ne reste donc que $\sum_{k|1, k|j} F(k)$ c'est-à-dire $\sum_{k|(i \wedge j)} F(k)$. Cette somme est

$$(\mathbf{1}) * F(i \wedge j) = ((\mathbf{1}) * \mu * f)(i \wedge j) = (e * f)(i \wedge j) = f(i \wedge j).$$

Finalement ${}^t ADA = M$. La matrice A est triangulaire avec des 1 sur la diagonale donc $\det A = 1$ et ainsi $\det M = \det D = \prod_{k=1}^n F(k)$.

- lorsque $f = \text{Id}$: on a $\mu * f = \varphi$ et $\det M = \prod_{k=1}^n \varphi(k)$,
- lorsque $f = \sigma$: on calcule $F = \mu * \sigma$ de nouveau sur les p^k avec p premier et $k \in \mathbb{N}^*$. Il reste $F(p^k) = \mu(1)\sigma(p^k) + \mu(p)\sigma(p^{k-1}) = \sigma(p^k) - \sigma(p^{k-1})$. De plus

$$\sigma(p^k) = \sum_{j=0}^k p^j = \frac{p^{k+1} - 1}{p - 1},$$

cela donne $F(p^k) = \frac{p^{k+1} - 1}{p - 1} - \frac{p^k - 1}{p - 1} = p^k$. Cette fois on a $\mu * \sigma = \text{Id}$. On obtient $\det M = n!$.

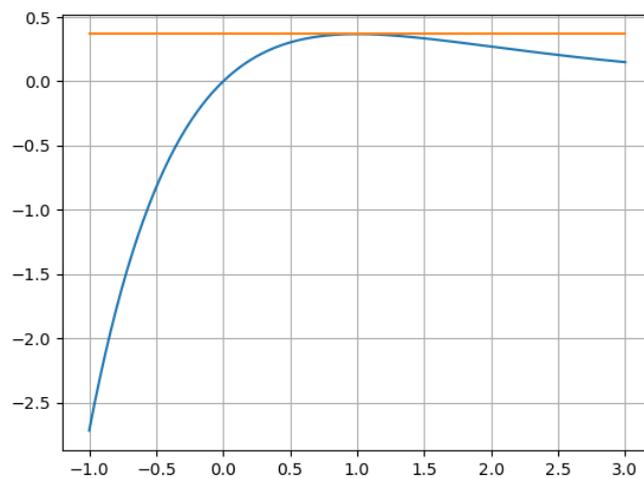
- lorsque $f = \tau$: on pose $T = \mu * \tau$. On a $T(p^k) = \tau(p^k) - \tau(p^{k-1})$. On a $\tau(p^k) = k + 1$ (les diviseurs sont les p^j pour j allant de 0 à k). Finalement $T(p^k) = 1$ et par produit $T = \mathbf{1}$. On en déduit que $\det M = 1$.

Exercice 16

- a) Les solutions de l'équation homogène sont $x \mapsto \alpha e^{-x}$ et une solution particulière est $x \mapsto \frac{1}{2}e^x$ (ou bien était-ce $y' + y = e^{-x}$ qu'il fallait résoudre?)
- b) avec

```
x = np.linspace(-1,3,300)
y = x*np.exp(-x)
y2 = np.array([np.exp(-1) for _ in x])
plt.plot(x,y,x,y2)
plt.grid()
plt.show()
```

on obtient



On voit que e^{-1} est obtenu pour $x = 1$ et pour e^{-1} , c'est autour de -0.2785 .

- c) On note $f(x) = xe^{-x}$. Pour tout $x \in \mathbb{R}$, $f'(x) = (x-1)e^x$. La fonction f est croissante sur $]-\infty, 1]$ avec une limite $-\infty$ en $-\infty$. Elle réalise une bijection de $]-\infty, 1]$ sur $]-\infty, e^{-1}]$.
- d) On note $u_n = \frac{n^{n-1}}{n!}$. Ce terme est strictement positif et

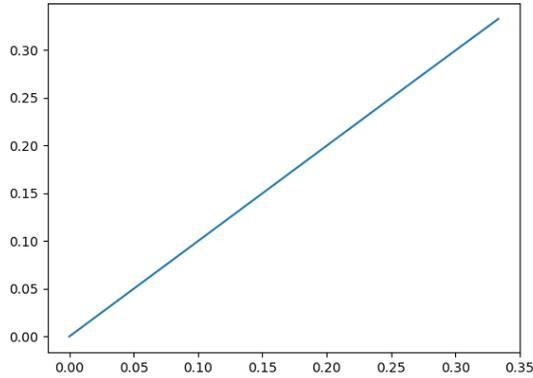
$$\frac{u_{n+1}}{u_n} = \frac{(n+1)^n}{n^{n-1}} \frac{1}{n+1} = \left(\frac{n+1}{n}\right)^{n-1} = \left(1 + \frac{1}{n}\right)^{n-1} \xrightarrow{n \rightarrow +\infty} e$$

Le rayon de convergence de la série entière est $\frac{1}{e}$.

```
def p(x):
    return x*np.exp(-x)

def S(N,x):
    s = 0
    fact = 1
    for n in range(1,N+1):
        fact *= n
        s += n**(n-1)*(x**n)/fact
    return s

X = np.linspace(0,1/3,200)
Y = S(8,p(X))
plt.plot(X,Y)
plt.show()
```



Il semblerait que les fonctions soient réciproques (sur certains intervalles).

e) i) On résout la seconde équation, ce qui donne $y(t) = y_0 e^{\lambda t}$. On reporte dans la première et on résout et on trouve $x(t) = x_0 e^{\lambda t} + y_0 t e^{\lambda t}$.

ii)

f) i) On note $g(t) = f(t) + \frac{(-1)^m}{m!}$. On a $f^{(m-1)}(0) = g^{(m-1)}(0)$ si $m \geq 2$. Or $g(t) \sim \frac{t^m}{m!}$; g est \mathcal{C}^∞ sur \mathbb{R} et admet un DL en 0 à tout ordre. L'équivalent précédent donne que toutes les dérivées de g en 0 jusqu'à l'ordre $m-1$ sont nulles. On a donc $f^{(m-1)}(0) = 0$ lorsque $m \geq 2$. Pour $m = 1$, on a $f(t) = e^t$ et $f^{(0)}(0) = 1$.

Puisque cela servira certainement après, on calcule cette valeur d'une seconde façon : on développe, on dérive et on évalue en 0. On a pour $m \geq 2$:

$$f(t) = \frac{1}{m!} \left(\sum_{k=0}^m \binom{m}{k} (-1)^{m-k} e^{kt} + (-1)^{m-1} \right)$$

et

$$f^{(m-1)}(0) = \sum_{k=0}^m \frac{1}{k!(m-k)!} (-1)^{m-k} k^{m-1}$$

ii) Les solutions sur \mathbb{R}_+^* et \mathbb{R}_-^* sont les fonctions $x \mapsto \alpha x$. Pour avoir une solution dérivable en 0, il faut les mêmes dérivées à droite et à gauche en 0 (ou un taux d'accroissement avec une limite finie), ce qui donne $x \mapsto \alpha x$.

iii) On effectue le calcul de plusieurs façons

$$\sum_{n=1}^{+\infty} \left(\sum_{k=0}^{+\infty} \frac{(-1)^k (nx)^k}{k!} \right) n^{n-1} \frac{x^n}{n!} = \sum_{n=1}^{+\infty} n^{n-1} e^{-nx} \frac{x^n}{n!} = \sum_{n=1}^{+\infty} \frac{n^{n-1}}{n!} (xe^{-x})^n = S(p(x))$$

Au passage le même calcul en valeur absolue donne

$$\sum_{n=1}^{+\infty} \left(\sum_{k=0}^{+\infty} \frac{(n|x|)^k}{k!} \right) n^{n-1} \frac{|x|^n}{n!} = \sum_{n=1}^{+\infty} \frac{n^{n-1}}{n!} (|x|e^{|x|})^n$$

qui existe dès que $|x|$ est suffisamment petit, ce qui garantit la sommabilité de la famille dès que x est assez proche de 0.

On somme suivant les paquets recommandés :

$$\begin{aligned} \sum_{(k,n) \in \mathbb{N} \times \mathbb{N}^*} (-1)^k \frac{1}{k!n!} n^{n+k-1} x^{n+k} &= \sum_{m=1}^{+\infty} \left(\sum_{(k,n) \in I_m} (-1)^k \frac{1}{k!n!} n^{n+k-1} x^{n+k} \right) \\ &= \sum_{m=1}^{+\infty} \left(\sum_{(k,n) \in I_m} (-1)^{m-n} \frac{1}{(m-n)!n!} n^{m-1} x^m \right) \\ &= \sum_{m=1}^{+\infty} \left(\sum_{n=1}^m (-1)^{m-n} \frac{1}{(m-n)!n!} n^{m-1} x^m \right) \\ &= x + \sum_{m=2}^{+\infty} \left(\sum_{n=1}^m (-1)^{m-n} \frac{1}{(m-n)!n!} n^{m-1} x^m \right) \end{aligned}$$

On peut ajouter le terme $n = 0$ dans la somme lorsque $m \geq 2$ et cette somme vaut alors 0 (c'est la question précédente avec un choix de variables vraiment judicieux pour embrouiller). Finalement, pour x proche de 0, on a $S(p(x)) = x$.

Exercice 19

- a) L'application $P \mapsto \|P\|_A$ est bien définie et à valeurs positives ou nulles. Si $\|P\|_A = 0$ alors P s'annule en $n + 1$ valeurs distinctes; il est donc nul puisque de degré au plus n . L'homogénéité s'écrit facilement. Si P et Q sont deux polynômes,

$$\text{pour tout } k \in \llbracket 0; n \rrbracket, |(P + Q)(a_k)| \leq |P(a_k)| + |Q(a_k)| \leq \|P\|_A + \|Q\|_A,$$

si bien que $\|P + Q\|_A \leq \|P\|_A + \|Q\|_A$. L'application est bien une norme sur $E = \mathbb{R}_n[X]$.

- b) • premier calcul : on cherche $d_{\{0,1\},1} = \inf\{\|X - a\|_A, a \in \mathbb{R}\}$ avec $A = \{0, 1\}$, soit

$$\min(\max(|a|, |1 - a|), a \in \mathbb{R})$$

```
import numpy as np
import scipy.optimize as opti

def f(a):
    return max(abs(a), abs(1-a))

opti.fmin(f, 0)
```

renvoie la valeur 0.5 (évidemment les calculs se font aussi à la main)

- second calcul : le polynôme pour lequel on calcule la norme s'écrit sous la forme $X^2 - (aX + b)$. On écrit des fonctions qui ont donc deux variables a et b . La fonction `fmin` optimise une fonction d'une seule variable vectorielle (il faut aller lire l'aide et la comprendre) :

```
def norme2(X):
    # renvoie la norme pour la polynôme X**2-(aX+b)
    # avec les points d'évaluation 0,1 et 2

    a,b = X
    def P(x):
        return abs(x*x-a*x-b)

    return max(P(0), P(1), P(2))
```

puis on minimise :

```
opti.fmin(norme2, np.array([0,0]))
```

renvoie un minimum valant 0.5 au point $(2, -0.5)$ (numériquement)

Exercice 20

- a) On vérifie qu'on a la relation de récurrence $a_{n+1} = \frac{2n+1}{2n+2}a_n$ ce qui permet de calculer les termes de proches en proche :

```
def a(n):
    s = 1
    for i in range(n):
        s = s*(2*i+1)/(2*i+2)
    return s
```

ou, puisque la liste des termes est intéressante pour la suite, autant la calculer directement plutôt que de refaire tous les calculs à chaque fois

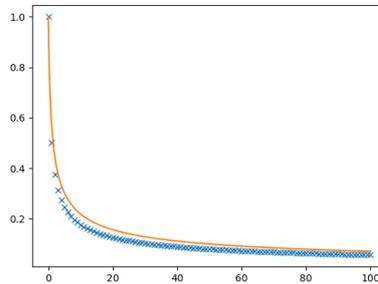
```
def liste_a(n):
    s = 1
    L = [1]
    for i in range(n):
        s = s*(2*i+1)/(2*i+2)
        L.append(s)
    return L
```

Quelques tests :

```
>>> L = [100, 1000, 10000, 100000, 1000000]
>>> [1/(n*a(n)**2) for n in L]
[3.1494564280813027, 3.1423781499034242, 3.1416711943878313,
3.1416005075813365, 3.1415934389888664]
```

On peut conjecturer que $\lim_{n \rightarrow +\infty} \frac{1}{na_n^2} = \pi$.

```
b) L = list(range(101))
A = liste_a(100)
X = np.linspace(0, 100, 1001)
Y = 1/np.sqrt(2*X+1)
plt.plot(L, A, "x")
plt.plot(X, Y)
plt.show()
```



Il semblerait que $a_n \leq \frac{1}{\sqrt{2n+1}}$ pour tout $n \in \mathbb{N}$. Cela revient à montrer que $(2n+1)a_n^2 \leq 1$. On pose $u_n = (2n+1)a_n^2$. On a $u_0 = a_0^2 = 1$.

Pour tout $n \in \mathbb{N}$,

$$\frac{u_{n+1}}{u_n} = \frac{2n+3}{2n+1} \left(\frac{a_{n+1}}{a_n} \right)^2 = \frac{2n+3}{2n+1} \frac{(2n+1)^2}{(2n+2)^2} = \frac{(2n+1)(2n+3)}{(2n+2)^2} = \frac{4n^2+8n+3}{4n^2+8n+4} < 1.$$

La suite (u_n) est donc décroissante positive et majorée par son premier terme qui vaut 1. Cela donne le bon résultat.

c) On peut appliquer un critère de d'Alembert puisque a_n n'est jamais nul. De plus $\lim_{n \rightarrow +\infty} \frac{na_{n+1}}{(n+1)a_n} = 1$ si bien que le rayon de convergence de la série entière vaut 1. On étudie aux deux extrémités. On a montré que $0 \leq \frac{a_n}{n} \leq \frac{1}{n\sqrt{2n+1}} \underset{n \rightarrow +\infty}{\sim} \frac{1}{\sqrt{2}n^{3/2}}$ donc la série converge absolument en 1 et -1 . Finalement l'ensemble de définition de f est $[-1, 1]$. La fonction f est continue sur $[-1, 1]$ puisque la série de fonctions associée converge normalement sur $[-1, 1]$.

d) Soit on reconnaît un DSE connu (avec $(1-u)^a$), soit on le retrouve. On note $h(u) = \sum_{n=0}^{+\infty} a_n u^n$. Pour tout $n \in \mathbb{N}$, on a

$$(2n+2)a_{n+1} = 2(n+1)a_{n+1} = (2n+1)a_n$$

On multiplie par u^n et on effectue la somme de $n=0$ à $+\infty$. Cela donne, pour tout $u \in]-1; 1[$,

$$2 \sum_{n=0}^{+\infty} (n+1)a_{n+1}u^n = 2 \sum_{n=0}^{+\infty} na_n u^n + \sum_{n=0}^{+\infty} a_n u^n$$

c'est-à-dire, $2h'(u) = 2uh'(u) + h(u)$ ou encore $2(1-u)f'(u) = f(u)$. On résout cette équation différentielle, qui donne $f(u) = A \exp\left(-\frac{1}{2} \ln(1-u)\right)$. La valeur $h(0) = a_0 = 1$ donne $h(u) = \frac{1}{\sqrt{1-u}}$ et finalement $\sum_{n=0}^{+\infty} a_n x^{2n} = \frac{1}{\sqrt{1-x^2}}$.

e) Si $w(x) = 1 + \sqrt{1-x^2}$ alors $w'(x) = -\frac{x}{\sqrt{1-x^2}}$. Soit $u(x) = \frac{1}{x\sqrt{1-x^2}} - \frac{1}{x}$. On remarque que u est continue en 0 car

$$u(x) = \frac{1 - \sqrt{1-x^2}}{x\sqrt{1-x^2}} \underset{x \rightarrow 0}{\sim} \frac{x^2}{2x} = \frac{x}{2}$$

donc u admet une limite finie nulle en 0. On pose pour la suite $u(0) = 0$.

On a

$$u(x) = \frac{1 - \sqrt{1-x^2}}{x\sqrt{1-x^2}} = \frac{(1 - \sqrt{1-x^2})(1 + \sqrt{1-x^2})}{x\sqrt{1-x^2}(1 + \sqrt{1-x^2})} = \frac{x^2}{x\sqrt{1-x^2}(1 + \sqrt{1-x^2})} = \frac{x}{\sqrt{1-x^2}} \frac{1}{1 + \sqrt{1-x^2}}$$

d'où une primitive

$$U(x) = -\ln\left(1 + \sqrt{1-x^2}\right)$$

où plutôt, afin d'avoir une primitive nulle en 0, $U(x) = \ln 2 - \ln\left(1 + \sqrt{1-x^2}\right)$

- f) On a, pour $x \in]-1, 1[$, $f'(x) = 2 \sum_{n=1}^{+\infty} a_n x^{2n-1} = 2 \frac{1}{x} \left(\sum_{n=0}^{+\infty} a_n x^{2n} - a_0 \right) = 2u(x)$. Puisque $f(0) = 0$, on a besoin de la primitive de u qui s'annule en 0. On a donc $f(x) = 2 \ln 2 - 2 \ln\left(1 + \sqrt{1-x^2}\right)$. En justifiant qu'on peut calculer $\sum_{n=1}^{+\infty} \frac{a_n}{n} u^n$ en remplaçant x^2 par u de chaque côté et en évaluant en -1 (possible par le théorème d'Abel radial), on trouve

$$\sum_{n=1}^{+\infty} \frac{(-1)^n a_n}{n} = 2 \ln 2 - 2 \ln(1 + \sqrt{2}) = \ln(12 - 8\sqrt{2})$$

Exercice 23

a) le programme :

```
import numpy as np

def somme(A, m):
    S = 0
    P = A.copy()
    for k in range(1, m+1):
        S += np.trace(P)/k
        P = np.dot(P, A)
    return S
```

et un petit essai :

```
>>> A = np.array([[ 0.25,  0.5 ], [ 1. , -0.5 ]])
>>> 1/np.linalg.det(np.eye(2)-A)
1.6000000000000001
>>> np.exp(somme(A, 500))
1.5999999999999999
```

b) La matrice A est trigonalisable. Notons $\lambda_1, \dots, \lambda_n$ ses valeurs propres (pas forcément distinctes). Elles sont complexes, de module inférieur à 1 et différentes de 1. On a alors

$$\chi_A = \prod_{i=1}^n (X - \lambda_i), \operatorname{tr}(A^k) = \sum_{i=1}^n \lambda_i^k$$

On a également $P_A(t) = \prod_{i=1}^n (1 - \lambda_i t)$. Par un calcul de dérivée d'un produit (on dérive chacun des facteurs en $-\lambda_i$ et on a la somme avec tous les termes sauf un qui vaut ce $-\lambda_i$), on obtient

$$\frac{P'_A(t)}{P_A(t)} = - \sum_{i=1}^n \frac{\lambda_i}{1 - \lambda_i t} = - \sum_{i=1}^n \lambda_i \left(\sum_{p=0}^{+\infty} \lambda_i^p t^p \right) = - \sum_{i=1}^n \left(\sum_{p=0}^{+\infty} \lambda_i^{p+1} t^p \right).$$

Sans justifier, si tout se passe bien (permutations, convergences...), on obtiendrait

$$\int_0^1 \frac{P'_A(t)}{P_A(t)} dt = - \sum_{i=1}^n \left(\sum_{p=0}^{+\infty} \lambda_i^{p+1} \int_0^1 t^p dt \right) = - \sum_{i=1}^n \left(\sum_{p=0}^{+\infty} \frac{\lambda_i^{p+1}}{p+1} \right) = - \sum_{p=0}^{+\infty} \frac{1}{p+1} \left(\sum_{i=1}^n \lambda_i^{p+1} \right) = - \sum_{p=0}^{+\infty} \frac{1}{p+1} \operatorname{tr}(A^{p+1}) = - \sum_{p=1}^{+\infty} \frac{\operatorname{tr}(A^p)}{p}.$$

Par linéarité, il suffit de prouver que, pour λ complexe de module inférieur à 1 et différent de 1, on a

$$\int_0^1 \frac{\lambda}{1 - \lambda t} dt = \sum_{p=1}^{+\infty} \frac{\lambda^p}{p}.$$

- si $|\lambda| < 1$. On a alors, pour tout $t \in [0, 1]$, $\frac{\lambda}{1 - \lambda t} = \lambda \sum_{p=0}^{+\infty} \lambda^p t^p = \sum_{p=0}^{+\infty} \lambda^{p+1} t^p$. On note $f_p(t) = \lambda^{p+1} t^p$, la fonction f_p est continue sur $[0, 1]$ et $|f_p(t)| \leq |\lambda|^{p+1}$. On a donc convergence normale de $\sum f_p$ sur $[0, 1]$ et ainsi uniforme. On peut permuter intégrale et somme. Cela donne

$$\int_0^1 \frac{\lambda}{1 - \lambda t} dt = \sum_{p=0}^{+\infty} \int_0^1 f_p(t) dt = \sum_{p=0}^{+\infty} \frac{\lambda^{p+1}}{p+1} = \sum_{p=1}^{+\infty} \frac{\lambda^p}{p}.$$

- si $|\lambda| = 1$ avec $\lambda \neq 1$. On utilise le théorème de convergence dominée sur la suite des sommes partielles. On note $S_m(t) = \sum_{p=0}^m f_p(t) = \sum_{p=0}^m \lambda^{p+1} t^p$. La suite converge simplement sur $[0, 1[$ et $\lim_{m \rightarrow +\infty} S_m(t) dt = \frac{\lambda}{1 - \lambda t} = S(t)$ si $t \in [0, 1[$ (attention de bien exclure le 1). La fonction S est continue sur $[0, 1[$. On a également, pour $t \in [0, 1[$, $S_m(t) = \lambda \frac{1 - \lambda^{m+1} t^{m+1}}{1 - \lambda t}$ et $|S_m(t)| \leq \frac{2}{|1 - \lambda t|}$. Or la fonction $t \mapsto \frac{1}{1 - \lambda t}$ est continue sur $[0, 1[$ ($1 - \lambda t$ ne s'annule pas lorsque t décrit $[0, 1[$). Elle est donc intégrable sur $[0, 1[$ et $[0, 1[$. Le théorème de convergence dominée permet d'assurer la permutation et ainsi

$$\lim_{m \rightarrow +\infty} \int_0^1 S_m(t) dt = \int_0^1 S(t) dt = \int_0^1 \frac{\lambda}{1 - \lambda t} dt$$

avec

$$\int_0^1 S_m(t) dt = \sum_{p=0}^m \frac{\lambda^{p+1}}{p+1} = \sum_{p=1}^{m+1} \frac{\lambda^p}{p}.$$

On obtient de nouveau le résultat.

On peut remarquer au passage qu'on a prouvé la convergence de $\sum \frac{\lambda^p}{p}$ lorsque $|\lambda| = 1$ et $\lambda \neq 1$.

c) On le fait en deux étapes :

- si on suppose que $\rho(A) > 1$: on montre que $\text{tr}(A^k)$ ne tend pas vers 0. Ce n'est pas immédiat car il pourrait y avoir plusieurs valeurs propres de module $\rho(A)$ et il faut montrer que leur somme des puissances k -ièmes ne tend pas vers 0...

- on a ainsi $\exp\left(\sum_{p=1}^{+\infty} \frac{\text{tr}(A^p)}{p}\right) = \exp\left(-\int_0^1 \frac{P'_A(t)}{P_A(t)} dt\right)$. On s'attend plus ou moins à ce que $\int_0^1 \frac{P'_A(t)}{P_A(t)} dt = [\ln(P_A(t))]_0^1 = \ln P_A(1)$

sauf qu'on est en complexe... on s'en inspire quand même en généralisant un peu. On va montrer que, pour tout $x \in [0, 1]$,

$\exp\left(-\int_0^x \frac{P'_A(t)}{P_A(t)} dt\right) = \frac{1}{P_A(x)}$. Soit $\varphi(x) = P_A(x) \exp\left(-\int_0^x \frac{P'_A(t)}{P_A(t)} dt\right)$. La fonction $t \mapsto \frac{P'_A(t)}{P_A(t)}$ est continue sur $[0, 1]$ donc φ est de classe \mathcal{C}^1 sur $[0, 1]$ et

$$\varphi'(x) = \left(P'_A(x) - P_A(x) \cdot \frac{P'_A(x)}{P_A(x)}\right) \exp\left(-\int_0^x \frac{P'_A(t)}{P_A(t)} dt\right) = 0.$$

La fonction φ est constante et $\varphi(0) = P_A(0)$. Le terme constant de P_A est le terme de plus haut degré de χ_A donc 1. On a bien

le résultat et par continuité sur $[0, 1]$ des différentes fonctions, on a $\exp\left(-\int_0^1 \frac{P'_A(t)}{P_A(t)} dt\right) = \frac{1}{P_A(1)} = \frac{1}{\chi_A(1)} = \frac{1}{\det(I_n - A)}$.

Exercice 24

- a) L'étude directe des variations ne semblent pas donner grand chose. On peut réécrire la somme. On voit que 1 n'est pas racine de Q_n ($Q_n(1) = 1 - n < 0$) si $n \geq 2$. On se place dans le cas où $n \geq 2$. On a

$$Q_n = X^n - \frac{X^n - 1}{X - 1} = \frac{X^{n+1} - 2X^n + 1}{X - 1}$$

ainsi Q_n s'annule sur $\mathbb{R}^+ \setminus \{1\}$ si et seulement si $P_n = X^{n+1} - 2X^n + 1$ s'annule $\mathbb{R}^+ \setminus \{1\}$. On a $P'_n = (n+1)X^n - 2nX^{n-1} = X^{n-1}((n+1)X - 2n)$. La fonction $x \mapsto P_n(x)$ est décroissante sur $[0, \frac{2n}{n+1}]$ puis croissante - on a $\frac{2n}{n+1} \in]1, 2[$. On a $P_n(0) = 1, P_n(1) = 0, P_n(2) = 1$. Ainsi P_n s'annule deux fois sur \mathbb{R}^+ : en 1 et quelque part en x_n entre $\frac{2n}{n+1}$ et 2 et Q_n s'annule uniquement en x_n (sur \mathbb{R}^+).

- b) On peut programmer une dichotomie entre 1 et 2 :

```
import numpy as np

def x(n):
    def f(x):
        p = 1
        s = 0
        # on calcule raisonnablement la somme
        for k in range(n):
            s += p
            p = p*x
        # en sortie p=x**n
        return p-s

    a=1
    b=2
    while b-a>1e-10:
        c = (a+b)/2
        if f(c)==0:
            return c
        if f(a)*f(c)<0:
            b = c
        else:
            a = c
    return c
```

```
>>> [x(n) for n in range(2,11)]
[1.618033988692332,
 1.839286755246576,
 1.9275619754916988,
 1.9659482366987504,
 1.9835828433861025,
 1.9919641966116615,
 1.9960311797331087,
 1.9980294702691026,
 1.99901863274863]
```

La suite à l'air croissante vers 2.

On peut également utiliser les fonctions sur les polynômes

```
def y(n):
    X = Polynomial([0,1])
    P = 1
    S = 0
    for k in range(n):
        S = S + P
        P = P * X
    # P contient X**n en sortie
```

```

Q = P-S
racines = Q.roots()
# on retourne la première racine positive qu'on voit
for r in racines:
    if r.real>0 and np.abs(r.imag)<1e-12:
        return r.real

```

La méthode semble quand même moins satisfaisante (mais fonctionne sans problème si on ne fait pas exploser le degré).

- c) On a montré que $\frac{2n}{n+1} = 2 - \frac{2}{n+1} < x_n < 2$. On en déduit donc que (x_n) converge vers 2.
- d) On écrit ce qu'on sait : $Q_n(x_n) = 0$ donne $x_n^{n+1} - 2x_n^n + 1 = 0$ où encore $x_n^n(2 - x_n) = 1$. On a également $x_n = 2 - h_n$ avec $0 < h_n < \frac{2}{n+1}$. On obtient

$$(*) \quad h_n(2 - h_n)^n = 1 \Leftrightarrow h_n = \frac{1}{(2 - h_n)^n} = \frac{1}{2^n(1 - h_n/2)^n} = \frac{1}{2^n} \exp(-n \ln(1 - h_n/2)).$$

La question est de connaître le comportement de $\ln(1 - h_n/2)$. On a $0 < h_n < \frac{2}{n+1}$ et $0 < \frac{h_n}{2} < \frac{1}{n+1}$. On en déduit que $0 < -n \ln(1 - \frac{h_n}{2}) < -n \ln(1 - \frac{1}{n+1})$. Le terme de droite admet une limite finie donc $-n \ln(1 - h_n/2)$ est bornée. En reprenant (*), cela permet d'obtenir $h_n = O_{n \rightarrow +\infty} \left(\frac{1}{2^n} \right)$. En réutilisant ce comportement plus précis pour h_n , on a alors $-n \ln(1 - \frac{h_n}{2}) \sim \frac{nh_n}{2}$ de limite nulle lorsque n tend vers $+\infty$ (car nh_n tend vers 0 maintenant que $h_n = O_{n \rightarrow +\infty} \left(\frac{1}{2^n} \right)$). En reprenant (*), on en déduit que $h_n \underset{n \rightarrow +\infty}{\sim} \frac{1}{2^n}$. On a donc $x_n = 2 - h_n$ soit $2 - x_n = h_n \underset{n \rightarrow +\infty}{\sim} \frac{1}{2^n}$ ou encore $2^n(2 - x_n)$ de limite 1.

- e) On a $x_n = 2 - \frac{1}{2^n} + y_n$ avec $y_n = o_{n \rightarrow +\infty} \left(\frac{1}{2^n} \right)$. On a $x_n^n(2 - x_n) = 1$ soit $n \ln(x_n) + \ln(2 - x_n) = 0$. On reporte le développement précédent

$$n \ln\left(2 - \frac{1}{2^n} + y_n\right) + \ln\left(\frac{1}{2^n} - y_n\right) = 0 = n \ln 2 + n \ln\left(1 - \frac{1}{2^{n+1}} + \frac{y_n}{2}\right) - n \ln 2 + \ln(1 - 2^n y_n) = 0,$$

ce qui donne après simplifications

$$n \ln\left(1 - \frac{1}{2^{n+1}} + \frac{y_n}{2}\right) = -\ln(1 - 2^n y_n)$$

en prenant un équivalent de chaque membre (avec y_n négligeable devant $1/2^n$), on obtient $-\frac{n}{2^{n+1}} \sim 2^n y_n$ et $y_n \sim -\frac{n}{2^{2n+1}}$. On en déduit $x_n = 2 - \frac{1}{2^n} - \frac{n}{2^{2n+1}} + z_n$ avec z_n négligeable devant le dernier terme. On pourrait poursuivre ainsi...

Exercice 25

- a) On peut montrer facilement par récurrence que pour tout $n \in \mathbb{N}$, on a x_n et y_n dans $[0, 7]$ (par exemple... c'est pour ne pas avoir une racine d'un nombre négatif).
- b) On peut le faire récursivement :

```
import numpy as np
def x(n):
    if n==0:
        return 0
    else:
        return np.sqrt(7-y(n-1))

def y(n):
    if n==0:
        return 0
    else:
        return np.sqrt(7+x(n-1))
```

On obtient

n	x_n	y_n
0	0	0
1	2.64575131106	2.64575131106
2	2.08668365809	3.10576098743
3	1.97338263207	3.01441265557
4	1.99639358455	2.99556048713
5	2.00110957043	2.99939887053
6	2.00015027672	3.00018492271
7	1.99995376879	3.00002504602
8	1.99999373849	2.99999229479
9	2.0000019263	2.99999895641

- c) Si ℓ_1 et ℓ_2 sont les limites éventuelles de (x_n) et (y_n) (respectivement), on doit avoir

$$\ell_1 = \sqrt{7 - \ell_2} \text{ et } \ell_2 = \sqrt{7 + \ell_1}.$$

Puisque les limites sont positives (et dans $[0, 7]$), on a $\ell_1^2 = 7 - \ell_2$ et $\ell_2^2 = 7 + \ell_1$. La différence donne $\ell_2^2 - \ell_1^2 = \ell_1 + \ell_2$ donc $(\ell_2 + \ell_1)(\ell_2 - \ell_1) = (\ell_2 + \ell_1)$. On ne peut pas avoir $\ell_1 + \ell_2 = 0$ sinon on aurait $\ell_1 = \ell_2 = 0$ ce qui ne convient pas avec les deux premières équations. On a donc, en simplifiant, $\ell_2 - \ell_1 = 1$. En remplaçant dans la première équation : $\ell_1^2 = 7 - (1 + \ell_1)$ soit $\ell_1^2 + \ell_1 - 6 = 0$. On a $\ell_1 = 2$ ou $\ell_1 = -3$ et le signe donne $\ell_1 = 2$. Enfin $\ell_2 = 3$.

- d) Il est toujours plus simple de prouver que des limites sont nulles. On note $u_n = x_n - 2$ et $v_n = y_n - 3$. On réécrit les relations de récurrence :

$$u_{n+1} = \sqrt{7 - y_n} - 2 = \sqrt{7 - (3 + v_n)} - 2 = \sqrt{4 - v_n} - 2 = \frac{-v_n}{2 + \sqrt{4 - v_n}},$$

et de la même manière

$$v_{n+1} = \sqrt{7 + x_n} - 3 = \sqrt{9 + u_n} - 3 = \frac{u_n}{3 + \sqrt{9 + u_n}}.$$

Par conséquent, $|u_{n+1}| \leq \frac{1}{2}|v_n|$ et $|v_{n+1}| \leq \frac{1}{3}|u_n|$. On peut conclure de plusieurs manières :

- on a $|u_{n+2}| \leq \frac{1}{6}|u_n|$. Ainsi $|u_{2n}| \leq \frac{1}{6^n}|u_0|$ et $|u_{2n+1}| \leq \frac{1}{6^n}|u_1|$. On en déduit que les deux suites extraites des termes pairs et impairs sont de limite nulle et ainsi la suite u converge vers 0. De même, v converge vers 0.
- on peut noter $w_n = \max(|u_n|, |v_n|)$. On a $0 \leq w_{n+1} \leq \frac{1}{2}w_n$ et par récurrence $0 \leq w_n \leq \frac{1}{2^n}w_0$. La suite w tend vers 0 et on a $|u| \leq w$ et $|v| \leq w$.

- e) Supposons que $(x_n)_{n \geq n_0}$ soit croissante. Puisque $y_{n+1} = \sqrt{7 + x_n}$, on aurait $(y_n)_{n \geq n_0+1}$ croissante. Or $x_{n+1} = \sqrt{7 - y_n}$ et $(x_n)_{n \geq n_0+2}$ serait décroissante (même situation pour les autres cas de figures). On peut utiliser les majorations précédentes. On a $|u_0| = 2$ et $|u_{2n}| \leq \frac{2}{12^n}$ donc pour $n = 4$, on a $|u_{2n}| \leq 10^{-3}$. On a $|u_1| = \sqrt{7} - 2 \leq 1$ et $|u_{2n+1}| \leq \frac{1}{12^n}$. Pour $n \geq 3$, on a $|u_{2n+1}| \leq 10^{-3}$. On est donc certain que $|x_n - 2| \leq 10^{-3}$ si $n \geq 6$. On a la même chose pour $|y_n - 3|$. Le rang $n_0 = 6$ convient (et en examinant les premiers termes, c'est le plus petit rang qui convient).

Exercice 26

- a) pour la programmation des coefficients binomiaux : au choix en programmant une fonction factorielle, soit avec la relation

$$\binom{n}{k} = \frac{n(n-1)\dots(n-k+1)}{k(k-1)\dots 1}.$$

```
def fact(n):
    p = 1
    for i in range(2, n+1):
        p = p*i
    return p

def binom(n, k):
    return fact(n)//fact(k)//fact(n-k)

def binom(n, k):
    Nume = 1
    Deno = 1
    for i in range(k):
        Nume = Nume * (n-i)
        Deno = Deno * (i+1)
    return Nume//Deno
```

- b) Pour la fonction de Bell, on a le choix entre une version récursive (mais avec une très mauvaise complexité puisqu'on passe le temps à recalculer les mêmes termes), on une version qui stocke les anciennes valeurs (c'est plutôt cela qui est attendu puisqu'on demande à la fin la liste des premiers termes de la suite) :

```
def Bell(n):
    if n==0:
        return 1
    S = 0
    for k in range(n):
        S += binom(n-1, k)*Bell(k)
    return S

def Bell(n):
    B = [1]
    for m in range(1, n+1):
        S = 0
        # ajout de Bell m
        for k in range(m):
            S += binom(m-1, k)*B[k]
        B.append(S)
    return B
```

ce qui donne

```
>>> Bell(6)
[1, 1, 2, 5, 15, 52, 203]
```

- c) Quelques tests semblent confirmer que $B_n \leq n!$. On le montre par récurrence. C'est vrai pour $B_0 = 1, B_1 = 1, B_2 = 2$ (si on veut insister). Si la propriété est vraie jusqu'à un certain rang n , alors

$$B_{n+1} \leq \sum_{k=0}^n \frac{n!}{(n-k)!} \leq \sum_{k=0}^n n! = (n+1)n! = (n+1)!$$

On en déduit que $\left| \frac{B_n}{n!} \right| \leq 1$ et que le rayon de convergence de la série entière donnée est au moins 1.

d) On note $S(x) = \sum_{n=0}^{+\infty} \frac{B_n}{n!} x^n$. On a la relation de récurrence

$$\frac{B_{n+1}}{n!} = \sum_{k=0}^n \frac{1}{(n-k)!} \frac{B_k}{k!}$$

ou encore

$$(n+1)B_{n+1} = \sum_{k=0}^n \frac{1}{(n-k)!} \frac{B_k}{k!}$$

Le second terme fait penser à un produit de Cauchy entre les développements en série entière de $S(x)$ et $\exp(x)$. Le premier terme à une dérivée. Pour tout $x \in]-R, R[$ (avec $R \geq 1$), on a

$$S'(x) = \sum_{n=1}^{+\infty} n B_n x^{n-1} = \sum_{n=0}^{+\infty} (n+1) B_{n+1} x^n = S(x) e^x$$

Puisque $S(0) = B_0 = 1$, on en déduit que $S(x) = \exp(\exp(x) - 1)$. On effectue alors un développement de cette expression pour avoir une expression de B_n : pour tout $x \in \mathbb{R}$,

$$\exp(\exp x - 1) = e^{-1} \sum_{n=0}^{+\infty} \frac{e^{nx}}{n!} = e^{-1} \sum_{n=0}^{+\infty} \left(\frac{1}{n!} \sum_{k=0}^{+\infty} \frac{(nx)^k}{k!} \right) = e^{-1} \sum_{n=0}^{+\infty} \sum_{k=0}^{+\infty} \frac{n^k x^k}{k! n!}.$$

La famille des $\frac{n^k x^k}{k! n!}$ est donc sommable (immédiat si $x > 0$ puisque la somme double qu'on vient de faire apparaître existe, idem si $x < 0$ avec la même somme avec $|x|$). On peut permuter l'ordre de sommation pour obtenir

$$S(x) = \sum_{k=0}^{+\infty} \frac{1}{k!} \left(e^{-1} \sum_{n=0}^{+\infty} \frac{n^k}{n!} \right) x^k$$

Par unicité du développement, on en déduit que $B_k = e^{-1} \sum_{n=0}^{+\infty} \frac{n^k}{n!}$ (ou si l'on préfère, $B_n = e^{-1} \sum_{k=0}^{+\infty} \frac{k^n}{k!}$, ce qui est, comme tout le monde l'a remarqué, le moment d'ordre n d'une loi de Poisson de paramètre 1).

Exercice 27

a) en version complète :

```
import numpy as np
import matplotlib.pyplot as plt

def P(n):
    def P(n,x):
        S = 0
        for i in range(2*n+1):
            S += x**i
        return S
    return lambda x:P(n,x)
```

on peut bien évidemment faire plus simple et créer simplement une fonction de 2 variables $P(n, x)$. Pour le tracé

```
x = np.linspace(-2,2,101)
for k in range(6):
    plt.plot(x,P(k)(x))
```

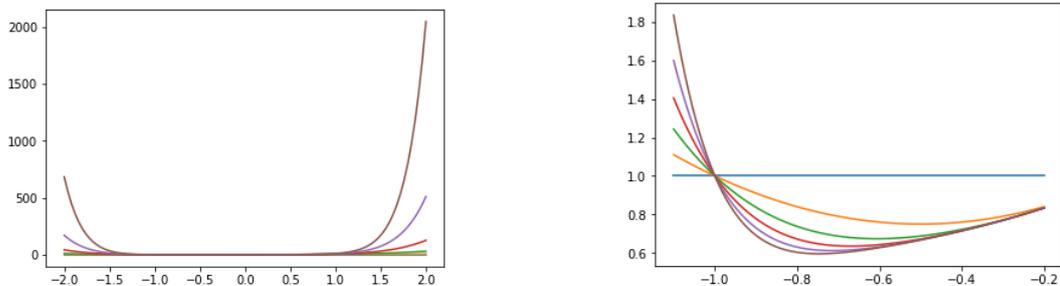


FIGURE 1 – Tracé des P_n , pour n de 0 à 5

b) On simplifie : $P_n(x) = \frac{1-x^{2n+1}}{1-x} = \frac{x^{2n+1}-1}{x-1}$ si $x \neq 1$ (et $P_n(1) = 2n+1$). Si $x > 1$ alors $x^{2n+1} - 1 > 0$ et $x - 1 > 0$ donc $P_n(x) > 0$. De même, si $x < 1$, $x^{2n+1} < 1$ et $x - 1 < 0$ donc $P_n(x) > 0$. On peut remarquer que P_n est strictement croissante sur \mathbb{R}^+ avec une valeur minimale 1 en 0. On a

$$P'_n(x) = \frac{(2n+1)x^{2n}(x-1) - (x^{2n+1}-1)}{(x-1)^2} = \frac{2nx^{2n+1} - (2n+1)x^{2n} + 1}{(x-1)^2}.$$

On note $Q_n(x) = 2nx^{2n+1} - (2n+1)x^{2n} + 1$. On a $Q'_n(x) = 2n(2n+1)x^{2n-1}(x-1)$.

x	$-\infty$	a_n	0	1	$+\infty$
$Q'_n(x)$		+	0	- 0	+
$Q_n(x)$	$-\infty$	↗ 0	↘ 1	↗ 0	↘ $+\infty$
$P'_n(x)$		-	0		+
$P_n(x)$	$+\infty$	↘ b_n		↗	$+\infty$

c) Le tableau de variation nous donne bien l'existence d'un unique a_n en lequel P_n est minimal. On a également $Q_n(-1) = -4n < 0$ donc $a_n \in]-1, 0[$.

d) On peut conjecturer graphiquement le comportement. On peut aussi essayer informatiquement;

```
def Pprime(n):
    def Pprime(n,x):
```

```

    S = 0
    for i in range(1, 2*n+1):
        S += i*x**(i-1)
    return S
return lambda x: Pprime(n, x)

import scipy.optimize as resol

def a(n):
    return resol.fsolve(Pprime(n), -1)[0]

```

et

```

>>> [a(n) for n in range(2, 10)]
[-0.60582958618826799,
 -0.67033204760309684,
 -0.71453772716734076,
 -0.74705407486515596,
 -0.77214163552346593,
 -0.79217785460570356,
 -0.80860489787230283,
 -0.82235341023852881]

```

et $a(1000)$ donne -0.99586246866079331 . La suite semble être décroissante vers -1 . On reprend les calculs : on a $P'_n(a_n) = Q_n(a_n) = 0$ ce qui donne

$$2n\alpha_n^{2n+1} - (2n+1)\alpha_n^{2n} + 1 = 0$$

On note $\alpha_n = -a_n$ pour avoir un nombre entre 0 et 1. On a

$$-2n\alpha_n^{2n+1} - (2n+1)\alpha_n^{2n} + 1 = 0 \text{ ou encore } 2n\alpha_n^{2n+1} + (2n+1)\alpha_n^{2n} = 1.$$

On a $\alpha_n^{2n+1} \leq \alpha_n^{2n}$ ce qui donne $1 \leq (4n+1)\alpha_n^{2n}$. On a également $(2n+1)\alpha_n^{2n} \leq 2n\alpha_n^{2n+1} + (2n+1)\alpha_n^{2n} = 1$. Cela donne

$$\frac{1}{4n+1} \leq \alpha_n^{2n} \leq \frac{1}{2n+1} \text{ et } -\frac{1}{2n} \ln(4n+1) \leq \ln \alpha_n \leq -\frac{1}{2n} \ln(2n+1).$$

Par croissances comparées, $\ln \alpha_n$ tend vers 0 et α_n vers 1. On a bien $\lim_{n \rightarrow +\infty} a_n = -1$.

Exercice 31

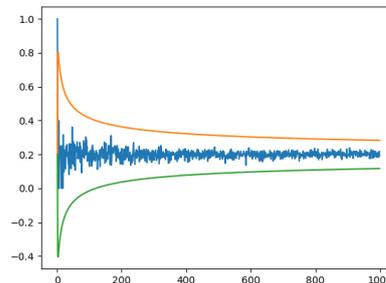
a)

```
i) import numpy.random as alea
import matplotlib.pyplot as plt
import numpy as np

def S(n,p):
    return alea.binomial(n,p)/n

def test(n,p):
    X = []
    Y = []
    Y1 = []
    Y2 = []
    for k in range(1,n):
        X.append(k)
        Y.append(S(k,p))
        Y1.append(p+np.sqrt(np.log(k)/k))
        Y2.append(p-np.sqrt(np.log(k)/k))
    plt.plot(X,Y)
    plt.plot(X,Y1)
    plt.plot(X,Y2)
    plt.show()
```

Si on essaie `test(1000,0.2)`, on obtient



ii)

Visiblement les valeurs prises par S_n se rapprochent de l'espérance et restent coincées entre les deux courbes proposées (du moins il semble qu'il y ait peu de chances pour ne pas être entre les deux courbes).

b)

- i) Plusieurs possibilités. On peut étudier le terme de droite en tant que fonction de t et montrer que le minimum est (supérieur ou) égal à e^{xt} . On peut voir cela comme une inégalité de convexité pour la fonction $u \mapsto e^{ut}$ avec $a = -1$, $b = 1$ et $\lambda = \frac{1-x}{2} \in [0, 1]$ avec $1 - \lambda = \frac{1+x}{2}$:

$$\exp\left(\left(\frac{1-x}{2}(-1) + \frac{1+x}{2}(1)\right)t\right) = \exp(xt) \leq \frac{1-x}{2}e^{-t} + \frac{1+x}{2}e^t$$

- ii) On a $0 \leq e^{-t} \leq \exp(tX) \leq e^t$ donc $\exp(tX)$ est bornée et elle est d'espérance finie. On a alors

$$\exp(tX) \leq \frac{1-X}{2}e^{-t} + \frac{1+X}{2}e^t$$

en passant aux espérances, avec $\mathbb{E}(X) = 0$, on obtient $\mathbb{E}(\exp(tX)) \leq \frac{e+e^{-t}}{2} = \text{ch } t$.

On a $\text{ch } t = \sum_{n=0}^{+\infty} \frac{x^{2n}}{(2n)!}$ et $\exp(t^2/2) = \sum_{n=0}^{+\infty} \frac{x^{2n}}{2^n n!}$. On vérifie facilement que $(2n)! \geq (2n)(2n-2)\dots 2 = 2^n n!$ puis la majoration demandée.

c)

i) Par indépendance,

$$\mathbb{E}(\exp(tS_n)) = \prod_{k=1}^n \mathbb{E}(\exp(tX_k)) = \prod_{k=1}^n \mathbb{E}(\exp((a_k t)(X_k/a_k)))$$

Or $\frac{1}{a_i} X_i$ vérifie les bonnes hypothèses, si bien que

$$\mathbb{E}(\exp((a_i t)(X_i/a_i))) \leq \exp((a_i t)^2/2)$$

En effectuant le produit, on obtient la majoration.

ii) S_n est bornée, donc $\exp(tS_n)$ également et à valeurs positives. De plus la fonction $x \mapsto \exp(tx)$ est croissante. On peut appliquer l'inégalité de Markov :

$$\mathbb{P}(S_n > \varepsilon) = \mathbb{P}(\exp(tS_n) > \exp(t\varepsilon)) \leq \frac{\mathbb{E}(\exp(tS_n))}{\exp(t\varepsilon)} \leq \exp\left(-t\varepsilon + \frac{t^2}{2} \sum_{i=1}^n a_i^2\right)$$

iii) En étudiant la fonction $t \mapsto \exp\left(-t\varepsilon + \frac{t^2}{2} \sum_{i=1}^n a_i^2\right)$ sur \mathbb{R}_+^* , on obtient un minimum qui vaut $\exp\left(\frac{-\varepsilon}{2 \sum_{i=1}^n a_i^2}\right)$. La majoration de la question précédente en cette valeur de t donne la majoration $\mathbb{P}(S_n > \varepsilon) \leq \exp\left(-\varepsilon + \frac{\varepsilon^2}{2 \sum_{i=1}^n a_i^2}\right)$.

iv)

Exercice 32

a) i)

```

import numpy as np
import numpy.random as rd

def loi(p):
    x = rd.random()
    if x < p:
        return 2
    else:
        return 1
    # ou
    # return 1+(x<p)

def simulation(k,p):
    position = 0
    while position < k:
        # tant qu'on n'a pas atteint ou dépassé k
        position += loi(p)
    return int(position==k)
    # ou
# if position==k:
# return 1
# else:
# return 0
#

```

ii) 30 essais, c'est un peu court pour faire des moyennes... on fait un peu plus (mais faire comme ils disent ou bien le signaler)

```

def moyenne(k,p):
    succes = 0
    for i in range(3000):
        succes += simulation(k,p)
    return succes/3000

def test(k):
    for i in range(10):
        p = i/10
        print(1/(1+p), moyenne(k,p))

```

En testant pour la case 100 (si on est trop proche de 0 c'est plus aléatoire :

```

>>> test(100)
1.0 1.0
0.9090909090909091 0.9183333333333333
0.8333333333333334 0.8336666666666667
0.7692307692307692 0.759
0.7142857142857143 0.7226666666666667
0.6666666666666666 0.679
0.625 0.633
0.5882352941176471 0.5913333333333334
0.5555555555555556 0.538
0.5263157894736842 0.5303333333333333

```

On peut raisonnablement conjecturer que la proportion se rapproche de $1/E(Y_1)$.b) i) Puisque la suite S_n est strictement croissante, pour une expérience donnée, il ne peut y avoir qu'au plus un seul n tel que $S_n(\omega) = k$. On a alors $E_k = \bigcup_{n \in \mathbb{N}^*} (S_n = k)$.

ii) On a

$$\mathbb{P}(E_k \cap (Y_1 = j)) = \mathbb{P}\left(\bigcup_{n \in \mathbb{N}^*} ((S_n = k) \cap (Y_1 = j))\right) = \mathbb{P}\left(\bigcup_{n \in \mathbb{N}^*} \left(Y_1 + \sum_{i=2}^n Y_i = k \cap (Y_1 = j)\right)\right) = \mathbb{P}\left(\left((Y_1 = j) \cap (Y_1 = k)\right) \cup \bigcup_{n \geq 2} \left(\sum_{i=2}^n Y_i = k - j\right)\right)$$

et puisque Y_1 est indépendante de toutes les autres variables Y_i , on obtient finalement, lorsque $j < k$:

$$\mathbb{P}(E_k \cap (Y_1 = j)) = \mathbb{P}\left(\bigcup_{n \in \mathbb{N}^*} ((S_n = k) \cap (Y_1 = j))\right) = \mathbb{P}\left(\bigcup_{n \geq 2} \left(\sum_{i=1}^{n-1} Y_{i+1} = k - j\right)\right) \mathbb{P}(Y_1 = j)$$

et puisque les Y_j suivent toutes la même loi,

$$\mathbb{P}\left(\bigcup_{n \geq 2} \left(\sum_{i=1}^{n-1} Y_{i+1} = k - j\right)\right) = \mathbb{P}\left(\bigcup_{n \geq 1} \left(\sum_{i=1}^n Y_{i+1} = k - j\right)\right) = \mathbb{P}(E_{k-j}).$$

Pour $k = j$, il ne reste plus que $\mathbb{P}(Y_1 = k)$.

iii) Puisque les événements $Y_1 = j$ pour $j \in \mathbb{N}^*$ forme un système complet d'événements,

$$u_k = \sum_{j=1}^{+\infty} \mathbb{P}(E_k \cap (Y_1 = j)) = \sum_{j=1}^k \mathbb{P}(E_k \cap (Y_1 = j)) = f_k + \sum_{j=1}^{k-1} u_{k-j} f_j,$$

puisque si on arrive en case k , on doit avoir un premier déplacement inférieur à k . Puisque $u_0 = 1$, on obtient bien la relation demandée.

iv) Toutes les séries entières qui apparaissent ont un rayon de convergence au moins égal à 1. On doit montrer que $f(t) = 1 +$

$f(t)u(t)$. Or, par produit de Cauchy, on a ($u(t)$ n'a pas de terme constant) $f(t)u(t) = \sum_{k=1}^{+\infty} c_k t^k$ où $c_k = \sum_{j=0}^k u_{k-j} f_j = \sum_{j=0}^k u_{k-j} f_j =$

$\sum_{j=1}^k u_{k-j} f_j$ car $f_0 = 0$. On a bien $c_k = u_k$ pour tout $k \in \mathbb{N}^*$. On a ainsi $f(t)u(t) = \sum_{k=1}^{+\infty} u_k t^k = f(t) - u_0 = f(t) - 1$.

v) On a $f_1 = \mathbb{P}(Y_1 = 1) = 1 - p$ et $\mathbb{P}(Y_1 = 2) = p$. Ainsi $u(t) = f_1 t + f_2 t^2 = (1 - p)t + p t^2$. On en déduit que $f(t) = \frac{1}{1 - (1 - p)t - p t^2} = \frac{1}{(1 - t)(1 + p t)}$. On décompose en éléments simples, puis on somme :

$$\frac{1}{(1 - t)(1 + p t)} = \frac{1}{1 + p} \frac{1}{1 - t} + \frac{p}{1 + p} \frac{1}{1 + p t} = \frac{1}{1 + p} \sum_{k=0}^{+\infty} (1 + p(-p)^k) t^k.$$

On a alors $u_k = \frac{1}{1 + p} (1 + p(-p)^k)$ et la limite de u_k est $\frac{1}{1 + p} = 1/\mathbb{E}(Y_1)$ (c'est donc seulement la limite qui vaut cette valeur...).

On peut vérifier que $u_0 = 1$ et $u_1 = \frac{1 - p^2}{1 + p} = 1 - p$ (ce qui est normal).

vi) On note k_1, \dots, k_n les entiers pour lesquels $\mathbb{P}(Y_1 = k)$ est non nul. On a $u(t) = \sum_{i=1}^n p_i t^{k_i}$ où $p_i = f_{k_i}$ pour simplifier... à terminer

Exercice 33

a)

```
def card_A(n):
    T = 0
    for i in range(1, n+1):
        for j in range(1, n+1):
            if entre_eux(i, j):
                T += 1
    return T

def f(n):
    return card_A(n) / (n**2)
```

On teste avec $\frac{6}{\pi^2} \approx 0.6079271$:

```
>>> f(100)
0.6087
>>> f(1000)
0.608383
>>> f(5000)
0.6080366
```

- b) On a $(a, b) \notin A_n$ si et seulement si (a, b) est dans l'un des U_i . On a $\overline{A_n} = \bigcup_{i=1}^k U_i$
- c) ce sont les nombres $p\ell$ tant que $p\ell \leq n$, soit $p \leq \frac{n}{\ell}$. Il y en a donc $\lfloor \frac{n}{\ell} \rfloor$.
- d) L'ensemble $\bigcap_{i \in I} U_i$ est l'ensemble des couples (a, b) avec a et b tous les deux divisibles par $d = \prod_{i \in I} p_i$. Cela revient à dire que a et b sont chacun des multiples de d . Il y en a $\lfloor \frac{n}{d} \rfloor$ pour chaque donc $\lfloor \frac{n}{d} \rfloor^2$ couples possibles.
- e) On applique la formule du crible :

$$\begin{aligned} \text{card } A_n &= n^2 - \text{card} \left(\bigcup_{i=1}^k U_i \right) = n^2 - \sum_{j=1}^k (-1)^{j-1} \sum_{1 \leq i_1 < \dots < i_j \leq k} \text{card} (U_{i_1} \cap \dots \cap U_{i_j}) \\ &= n^2 + \sum_{j=1}^k (-1)^j \sum_{1 \leq i_1 < \dots < i_j \leq k} \text{card} (U_{i_1} \cap \dots \cap U_{i_j}) \end{aligned}$$

Si d est un entier entre 1 et n , on calcule $\mu(d) \lfloor \frac{n}{d} \rfloor^2$:

- si $d = 1$ alors il vaut n^2 ,
- si d contient un p_i^2 , le terme est nul,
- si $d = p_{i_1} \dots p_{i_j}$, le terme vaut $(-1)^j \text{card} (U_{i_1} \cap \dots \cap U_{i_j})$

On obtient bien $\text{card} (A_n) = \sum_{d=1}^n \mu(d) \lfloor \frac{n}{d} \rfloor^2$.

- f) une petite question toute simple... évidemment, on s'intéresse à $\lim_{n \rightarrow +\infty} \frac{1}{n^2} \text{card} (A_n)$. On a $\frac{1}{n^2} \text{card} (A_n) = \sum_{d=1}^n \frac{\mu(d)}{n^2} \lfloor \frac{n}{d} \rfloor^2$. Lorsque n tend vers $+\infty$, on a (par encadrement simple), $\frac{1}{n^2} \lfloor \frac{n}{d} \rfloor^2 \underset{n \rightarrow +\infty}{\sim} \frac{1}{d^2}$. On s'attend donc à avoir une limite $\sum_{d=1}^{+\infty} \frac{\mu(d)}{d^2}$. On justifiera cette limite dans un second temps. On s'intéresse à son calcul.
- La série $\sum \frac{\mu(d)}{d^2}$ converge absolument, comme $\sum \frac{1}{n^2}$. Par résultat sur les sommes doubles :

$$S = \left(\sum_{d=1}^{+\infty} \frac{\mu(d)}{d^2} \right) \left(\sum_{n=1}^{+\infty} \frac{1}{n^2} \right) = \sum_{(d,n) \in \mathbb{N}^{*2}} \frac{\mu(d)}{(dn)^2}$$

On effectue une sommation par paquet sur les paquets $I_k = \{(d, n) \in \mathbb{N}^{*2}, dn = k\}$. Cela donne

$$S = \sum_{k=1}^{+\infty} \frac{1}{k^2} \left(\sum_{d|k} \mu(d) \right)$$

- on calcule $\sum_{d|k} \mu(d)$. On note q_1, \dots, q_m les diviseurs premiers de k . Lorsque d est divisible par l'un des q_j^2 , $\mu(d) = 0$. Lorsqu'on a p facteurs premiers deux à deux distincts dans d , on a $\mu(d) = (-1)^p$. Il y a exactement $\binom{m}{p}$ p -uplets de la sorte. En les regroupant, on a

$$\sum_{d|k} \mu(d) = \sum_{p=0}^m (-1)^p \binom{m}{p} = (1-1)^m = 0,$$

sauf lorsque $k = 1$ où on a alors un seul terme valant 1. Finalement

$$S = 1 + \sum_{k=2}^{+\infty} \frac{0}{k^2} = 1 = \frac{\pi^2}{6} \sum_{d=1}^{+\infty} \frac{\mu(d)}{d^2} \text{ et } \sum_{d=1}^{+\infty} \frac{\mu(d)}{d^2} = \frac{6}{\pi^2}.$$

- il reste à montrer ce que l'on a admis, à savoir

$$\lim_{n \rightarrow +\infty} \frac{1}{n^2} \sum_{d=1}^n \mu(d) \left[\frac{n}{d} \right]^2 = \sum_{d=1}^{+\infty} \frac{\mu(d)}{d^2}.$$

On étudie la différence

$$S_n = \sum_{d=1}^n \mu(d) \left(\frac{1}{n^2} \left[\frac{n}{d} \right]^2 - \frac{1}{d^2} \right)$$

On a ensuite $0 \leq \frac{n}{d} - 1 < \left[\frac{n}{d} \right] \leq \frac{n}{d}$ et ainsi $\left(\frac{n}{d} - 1 \right)^2 < \left[\frac{n}{d} \right]^2 \leq \frac{n^2}{d^2}$, puis

$$\frac{1}{n^2} \left(\frac{n^2}{d^2} - 2 \frac{n}{d} + 1 \right) - \frac{1}{d^2} \leq \left(\frac{1}{n^2} \left[\frac{n}{d} \right]^2 - \frac{1}{d^2} \right) \leq 0$$

On a ainsi $\left| \frac{1}{n^2} \left[\frac{n}{d} \right]^2 - \frac{1}{d^2} \right| \leq \frac{2}{nd}$ et ainsi

$$|S_n| \leq \sum_{d=1}^n \left| \frac{1}{d^2} - \frac{1}{n^2} \left[\frac{n}{d} \right]^2 \right| \leq \frac{2}{n} \sum_{d=1}^n \frac{1}{d}$$

On sait que $\sum_{d=1}^n \frac{1}{d} \underset{n \rightarrow +\infty}{\sim} \ln n$, ce qui donne $\lim_{n \rightarrow +\infty} S_n = 0$.