Solutions

Exercice 4

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

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

On essaie de déterminer des relations de récurrence pour le calcul. Comme $B_{k,n}$ admet 0 et 1 comme racine lorsque $1 \leqslant k \leqslant n-1$, on obtient que $\int_0^1 B'_{k,n}(t)\mathrm{d}t = 0$ 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 $I_{k,n} = \int_0^1 B_{k,n}(t)\mathrm{d}t$, on obtient

$$\forall k \in [\![1; n-1]\!], n(I_{k-1,n-1} - I_{k,n-1}) = 0$$

et ainsi $I_{k,n-1} = I_{k-1,n-1}$. La suite est constante et, pour tout $k \in [\![0; n]\!]$, $\int_0^1 B_{k,n}(t)\mathrm{d}t = \frac{1}{n+1}$.

d) On vérifie que, pour $n \in \mathbb{N}$, la famille $(B_{k,n})_{k\in[\![0;n]\!]}$ 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 $\mathscr{C}^2$ sur $[0,1]$ telle que, pour tout $k \in [\![0;n]\!]$, $f\left(\frac{k}{n}\right) = \alpha_k$ (par exemple en prenant un polynôme d'interpolation de Lagrange).

Exercice 5

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

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

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 6

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 $X^TBX = (AX)^T(AX) = \|AX\|_2^2 \geqslant 0$ et $X^TBX = \lambda\|X\|_2^2$ donc $\lambda \geqslant 0$.

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

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

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

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

ii) Chaque coefficient diagonal de B vaut n (la norme au carré d'une colonne de A). On a $(\det A^TA) = (\det A)^2 = \det B \leqslant n^n$ et $|\det A| \leqslant n^{n/2}$.

d) Si $\frac{1}{\sqrt{n}}A$ est orthogonale alors $|\det A|.\frac{1}{n^{n/2}} = 1$ 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 lieue 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 $C^TC = \frac{1}{n}B = I_n$.

Exercice 7

a) i) le plus simple puisque la question est posée ainsi est de réaliser un crible dÉratosthène (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
```

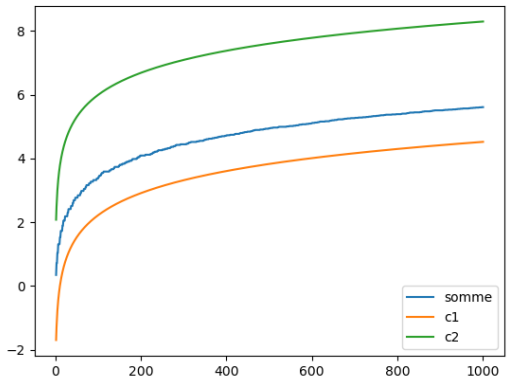
```
>>> premier(20)
[0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0]
```

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 $x-1 < \lfloor x\rfloor \leqslant x$, on obtient

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

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

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

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

Cela donne l'encadrement

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

c) i) On a pour tout $x\in]-1,1[$, $\frac{1}{1-x} = \sum_{k=0}^{+\infty} x^k$ et $\frac{1}{(1-x)^2} = \sum_{k=1}^{+\infty} k x^{k-1}$. 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 $\left(\frac{\ln n}{n(n+1)}\right)_{n\in\mathscr{P}}$ est sommable.

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

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

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

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

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

$$0 \leqslant 1 - k\ln(1+\frac{1}{k}) \leqslant \frac{1}{2k}$$

On utilise enfin que $v_n - v_1 = v_n = \sum_{k=1}^{n-1} (v_{k+1} - v_k)$ ce qui donne $0 \leqslant v_n \leqslant \frac{1}{2}\sum_{k=1}^{n-1} \frac{1}{k}$. 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 $w_m = \prod_{\substack{p\in\mathscr{P}\\ m+1<p\leqslant 2m+1}} p \leqslant \binom{2m+1}{m} = \binom{2m+1}{m+1}$. 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

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

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 $2^{2m+1} = (1+1)^{2m+1} = \sum_{k=0}^{2m+1}\binom{2m+1}{k} = \binom{2m+1}{m} + \binom{2m+1}{m+1} + \cdots$ (termes positifs). On a donc $2\binom{2m+1}{m} \leqslant 2^{2m+1}$, ce qui donne $\binom{2m+1}{m} \leqslant 4^m$.
- 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 $\prod_{p\in\mathscr{P}_{n+1}} p = \prod_{p\in\mathscr{P}_n} p \leqslant 4^n \leqslant 4^{n+1}$. Si $n+1 = 2m+1$ est premier, alors

$$\prod_{p\in\mathscr{P}_{2m+1}} p = \prod_{p\in\mathscr{P}_{m+1}} p.w_m \leqslant 4^{m+1}4^m = 4^{2m+1}$$

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

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

soit

$$n s_n \leqslant \sum_{p\in\mathscr{P}_n} \ln p + n\ln n - n + 1 + \ln n$$

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

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

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

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

Exercice 8

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 $[\![1\,;p-1]\!]$ 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 $pB_{p-1}\sum_{k=1}^{p-1} a_k = 2(p-1)!A_{p-1}$ 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 $k \in [\![1\,;p-1]\!]$. 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 $\sum_{k=1}^{p-1} a_k = p.q$. Alors la question précédente donne $p^2.q = 2(p-1)!H_{p-1}$. Le raisonnement précédent donne $p^2 | A_{p-1}$.

Exercice 9

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 $P_k = \lambda_k(X^k - \sum_{i=0}^{k-1} \left\langle X^k, P_i, P \right\rangle_i)$ 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 $R_{k-1}[X]$ dans $R_k[X]$ - 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 $x_1 < \ldots < x_{n-1}$ les racines de P_{n-1} et $y_0 < y1 < \ldots < y_{n-1}$ celles de P_n. L'hypothèse de récurrence donne
$$y_0 < x_1 < y_1 < x_2 < \ldots < x_{n-1} < y_{n-1}$$
Puisque les racines de P_{n-1} sont simples et que $\lim_{x\to+\infty} P_{n-1}(x) = +\infty$, 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 $\lim_{x\to+\infty} P_{n+1}(x) = +\infty$, 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 10

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 $z_n = \frac{1}{2}\begin{pmatrix} (2+\sqrt{3})^n + (2-\sqrt{3})^n \\ \frac{1}{\sqrt{3}}\left((2+\sqrt{3})^n - (2-\sqrt{3})^n\right)\end{pmatrix}$

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 $x \geqslant p-1$.

Exercice 11

a) Soient T_1 et T_2 deux polynômes et $\lambda \in \mathbb{C}$. On écrit les divisions euclidiennes $T_1P' = Q_1P + R_1$ et $T_2P' = Q_2P + R_2$ avec R_1 et R_2 de degré au plus $n-1$. On a alors $(T_1 + \lambda T_2) = (Q_1 + \lambda T_2)P + (R_1 + \lambda R_2)$ avec $\deg(R_1 + \lambda R_2) \leqslant n-1$. 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)
```python
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 $i \in [\![1; n]\!]$, on a $L_iP' = Q_iP + R_i$ 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 $TP' = (X-\lambda)^2 Q_1 Q_2 = PQ_2$ 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 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 déja sa diagonale constante, on peut prendre $\theta = 0$, c'est-à-dire $\Omega = I_2$.
- lorsque $a \neq d$, on obtient cotan $(2\theta) = \dfrac{b+c}{d-a}$ (si on préfère tan, on obtient en général, $\theta = \frac{1}{2}\arctan(\frac{d-a}{b+c})$ 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 $\sup(x,y) = \dfrac{x+y+|x-y|}{2}$ 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 $M \mapsto (|M_{ii} - M_{jj}|)_{i,j\in[\![1;n]\!]}$ 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

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

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 égaler) mais on a changé plein d'autres termes $|M_{kk} - M_{ii}|$. On va légérement changer le départ : on ordonne les éléments diagonaux de M : $\alpha_1 \leqslant \alpha_2 \leqslant \dots \leqslant \alpha_n$ 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 $\lim_{x\to k^+} R_n(x) = +\infty$, et $\lim_{x\to(k+1)^-} R_n(x) = -\infty$, 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 $a_{n,k} \leqslant a_{n-1,k}$. 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 $a_{n,k} \to k$, $A_n \to \sum_{i=0}^{k-1} \frac{1}{l-i} = C$ et puisque $k \leqslant a_{n,k} \leqslant k+1$,

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

on peut encadrer B_n à l'aide des nombres harmoniques : $H_{n-k} \leqslant B_n \leqslant H_{n-k-1} + \frac{1}{k+1-a_{n,k}}$, 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
$$\theta : \left\{ \begin{array}{ccc} \mathbb{Z}/(mn)\mathbb{Z} & \to & \mathbb{Z}/n\mathbb{Z} \times \mathbb{Z}/m\mathbb{Z} \\ \overline{x}^{mn} & \mapsto & (\overline{x}^n, \overline{x}^m) \end{array} \right.$$
où $\overline{x}^p$ 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 $x \in [\![1; mn]\!]$, alors x est premier avec mn si et seulement si $\overline{x}$ 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 $\beta_i \in [\![1; \alpha_i]\!]$ 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 $x \in [\![1; mn]\!]$ 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

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

Finalement $\mathbf{1}*\mu(n)=\delta_{1n}=e(n)$ 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 permier. En effet, si on connait 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 $j\in[\![0;k]\!]$. On a alors $(\mathbf{1}*\mu)(p^k)=\sum_{j=0}^{k}\mu(p^j)=\mu(1)+\mu(p)+0=1-1=0$. Ainsi $(\mathbf{1}*\mu)$ est nulle sur tous les p^k avec $k\neq 0$. On en déduit que pour tout $n\in\mathbb{N}^*$, $(\mathbf{1}*\mu)(n)=0$.

e) On a $h(n)=(\mathbf{1}*H)(n)$ d'où $h=\mathbf{1}*H$ et ainsi $\mu*h=\mu*\mathbf{1}*H=e*H=H$. Avec $H=\varphi$, on obtient $h(n)=\sum_{d|n}\varphi(d)=\varphi*\mathbf{1}$. Comme précédemment, on calcule H en p^k. On a (voir cours), $\varphi(p^j)=p^j-p^{j-1}$ si $j\geqslant 1$ 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 $\varphi * \mathbf{1} = \mathrm{Id}$ et ainsi $\varphi = \mathrm{Id} * \mu = \mu * \mathrm{Id}$.

f) On calcule brutalement :

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

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

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

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

- lorsque $f = \mathrm{Id}$: 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 $F(p^k) = \dfrac{p^{k+1}-1}{p-1} - \dfrac{p^k-1}{p-1} = p^k$. Cette fois on a $\mu * \sigma = \mathrm{Id}$. 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) On intègre le polynôme précédent et on détermine la constante d'intégration avec l'intégrale qui doit être nulle. On trouve

$$A_1 = X - \frac{1}{2} \qquad A_2 = \frac{1}{2}X^2 - \frac{1}{2}X + \frac{1}{12} \qquad A_3 = \frac{1}{6}X^3 - \frac{1}{4}X^2 + \frac{1}{12}X.$$

b) i) on le fait récursivement (ce n'est pas une obligation) :

```
from numpy.polynomial import Polynomial

def A(n):
    if n==0:
        return Polynomial([1])
    P = A(n-1)
    P = P.integ()  # le polynôme à une constante près
    Q = P.integ() # primitive nulle en 0
    P = P-Q(1)
    return P
```

on peut alors tester (lire coefficients sont dans l'ordre des degrés croissants) - la fonction `print` permet d'avoir un affichage plus lisible

```
>>> print(A(3))
-6.9388939e-18 + 0.08333333 x - 0.25 x**2 + 0.16666667 x**3
```

On regarde les premières valeurs de `A(n)(0)`. On constate que les valeurs pour n impair différent de 1 sont nulles et que celles pour les indices pairs tendent vers 0 (et ainsi $\lim_{n\to+\infty} a_n = 0$).

ii) on regarde :

```
>>> [A(n)(1)-A(n)(0) for n in range(7)]
```

ne donne que des 0.

```
>>> X = Polynomial([0,1])
>>> print(A(2),A(2)(1-X))
0.08333333 - 0.5 x + 0.5 x**2
0.08333333 - 0.5 x + 0.5 x**2
>>> print(A(3),A(3)(1-X))
-6.9388939e-18 + 0.08333333 x - 0.25 x**2 + 0.16666667 x**3
-6.9388939e-18 - 0.08333333 x + 0.25 x**2 - 0.16666667 x**3
>>> print(A(4),A(4)(1-X))
-0.00138889 - (6.9388939e-18) x + 0.04166667 x**2 - 0.08333333 x**3 +
   0.04166667 x**4
-0.00138889 - (6.9388939e-18) x + 0.04166667 x**2 - 0.08333333 x**3 +
   0.04166667 x**4
>>> print(A(5),A(5)(1-X))
-1.08420217e-19 - 0.00138889 x - (3.46944695e-18) x**2 + 0.01388889 x**3 -
   0.02083333 x**4 + 0.00833333 x**5
-1.08420217e-19 + 0.00138889 x + (5.20417043e-18) x**2 - 0.01388889 x**3 +
   0.02083333 x**4 - 0.00833333 x**5
```

Visiblement $A_n(1-X)$ vaut $(-1)^n A_n(X)$.

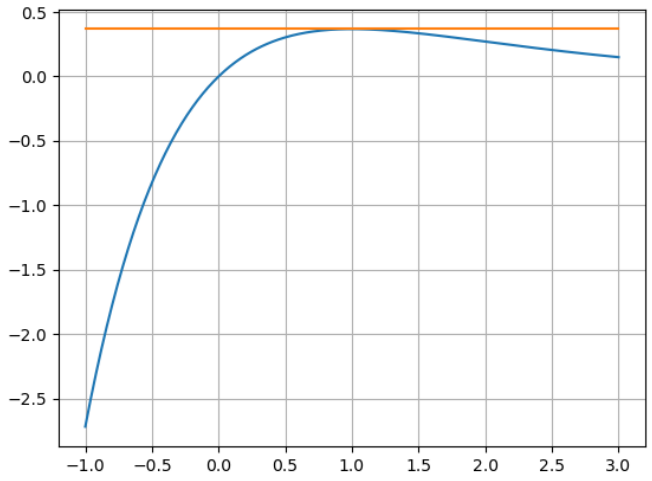
Exercice 17

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

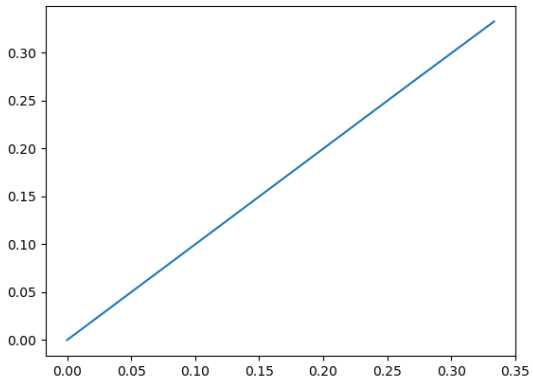
$$\frac{u_{n+1}}{u_n} = \frac{(n+1)^n}{n^{n-1}} \frac{1}{n+1} = \left(\frac{n+1}{n}\right)^{n-1} = \left(1+\frac{1}{n}\right)^{n-1} \underset{n\to+\infty}{\longrightarrow} e$$

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 $m \geqslant 2$. Or $g(t) \underset{t\to 0}{\sim} \frac{t^m}{m!}$; g est $\mathscr{C}^\infty$ 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 $m \geqslant 2$. 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 $m \geqslant 2$:

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

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 $m \geqslant 2$ 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) 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

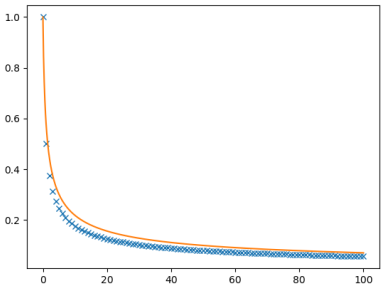
```
def liste_a(n):
    s = 1
    L = [1]
    for i in range(n):
        s = s*(2*i+1)/(2*i+2)
        L.append(s)
    return L
```

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 $\lim\limits_{n\to+\infty} \dfrac{1}{na_n^2} = \pi$.

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 $a_n \leqslant \dfrac{1}{\sqrt{2n+1}}$ pour tout $n\mathbb{N}$. Cela revient à montrer que $(2n+1)a_n^2 \leqslant 1$. 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 $\lim\limits_{n\to+\infty} \dfrac{na_{n+1}}{(n+1)a_n} = 1$ si bien que le rayon de convergence de la série entière vaut 1. On étudie aux deux extrémités. On a montré que $0 \leqslant \dfrac{a_n}{n} \leqslant \dfrac{1}{n\sqrt{2n+1}} \underset{n\to+\infty}{\sim} \dfrac{1}{\sqrt{2}n^{3/2}}$ 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 $(1-u)^\alpha$), 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 $u\in]-1;1[$,

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

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 $h(u)=\dfrac{1}{\sqrt{1-u}}$ et finalement $\sum_{n=0}^{+\infty}a_nx^{2n}=\dfrac{1}{\sqrt{1-x^2}}$.

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

$$u(x)=\frac{1-\sqrt{1-x^2}}{x\sqrt{1-x^2}}\underset{x\to0}{\sim}\frac{x^2}{2x}=\frac{x}{2}$$

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, $U(x)=\ln2-\ln\left(1+\sqrt{1-x^2}\right)$

f) On a, pour $x\in]-1,1[$, $f'(x)=2\sum_{n=1}^{+\infty}a_nx^{2n-1}=2\dfrac{1}{x}\left(\sum_{n=0}^{+\infty}a_nx^{2n}-a_0\right)=2u(x)$. Puisque $f(0)=0$, on a besoin de la primitive de u qui s'annule en 0. On a donc $f(x)=2\ln2-2\ln\left(1+\sqrt{1-x^2}\right)$. En justifiant qu'on peut calculer $\sum_{n=1}^{+\infty}\dfrac{a_n}{n}u^n$ en remplaçant x^2 par u de chaque côté et en évaluant en -1 (possible par le théorème d'Abel radial), on trouve

$$\sum_{n=1}^{+\infty}\frac{(-1)^na_n}{n}=2\ln2-2\ln(1+\sqrt{2})=\ln(12-8\sqrt{2})$$

Exercice 22

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 ($Q_(1) = 1-n<0$) si $n \geqslant 2$. On se place dans le cas où $n \geqslant 2$. 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 $\mathbb{R}^+\backslash\{1\}$ si et seulement si $P_n = X^{n+1}-2X^n+1$ s'annule $\mathbb{R}^+\backslash\{1\}$. 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 raisonablement 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 utilisder 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).

On peut également utiliser la fonction `fsolve` de `scipy.optimize` :

```
def P(n):
    # autre version pour définir le polynôme
    L = [-1]*n+[1]
    return Polynomial(L)

def x(n):
    return resol.fsolve(P(n),2)[0]
    # on part de 2, proche de la racine
```

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

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

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 $h_n = \underset{n\to+\infty}{O}\left(\frac{1}{2^n}\right)$. 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 $h_n = \underset{n\to+\infty}{O}\left(\frac{1}{2^n}\right)$). En reprenant $(*)$, on en déduit que $h_n \underset{n\to+\infty}{\sim} \frac{1}{2^n}$. On a donc $x_n = 2 - h_n$ soit $2 - x_n = h_n \underset{n\to+\infty}{\sim} \frac{1}{2^n}$ ou encore $2^n(2-x_n)$ de limite 1.

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

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

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 23

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

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

```
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 $B_n \leqslant n!$. 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

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

On en déduit que $\left|\frac{B_n}{n!}\right| \leqslant 1$ 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 $R \geqslant 1$), on a

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

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 25

a) i)
```
import numpy.random as rnd

def simulS(l,a,b):
    N = rnd.poisson(l)
    s = 0
    for k in range(N):
        X = rnd.randint(a,b+1)
        s += X
    return s
```

ii) on effectue les tests sur plusieurs valeurs au hasard, après avoir programmé une fonction qui approche l'espérance :

```
def esperanceS(l,a,b):
    n = 10000
    s = 0
    for _ in range(n):
        s += simulS(l,a,b)
    return s/n

def test():
    for _ in range(20):
        l = rnd.randint(1,11)
        a = rnd.randint(0,5)
        b = rnd.randint(6,12)
        e = l*(a+b)/2
        print("lambda :",l,"a :",a,"b : ",b," : ",e," -> ",esperanceS(l,a,b)
            )
```

Cela donne par exemple :

```
lambda : 3 a : 0 b :  6  :  9.0  ->  8.9833
lambda : 9 a : 4 b :  7  :  49.5  ->  49.6687
lambda : 7 a : 0 b :  8  :  28.0  ->  28.1473
lambda : 6 a : 4 b :  11  :  45.0  ->  44.8733
lambda : 7 a : 4 b :  7  :  38.5  ->  38.492
lambda : 2 a : 4 b :  9  :  13.0  ->  13.069
lambda : 7 a : 4 b :  7  :  38.5  ->  38.8695
lambda : 10 a : 4 b :  8  :  60.0  ->  59.8971
lambda : 8 a : 1 b :  10  :  44.0  ->  44.1632
lambda : 10 a : 4 b :  10  :  70.0  ->  69.8951
lambda : 4 a : 1 b :  10  :  22.0  ->  21.9546
...
```

iii)

```
def T(alpha,beta):
    s = 0
    for k in range(1,1000):
        s += rnd.randint(-1,2)
        if s<=alpha or s>=beta:
            return k

    return 0

def esperanceT(alpha,beta):
    s, n = 0,10000
    for _ in range(n):
        s += T(alpha,beta)
    return s/n
```

```
def test2():
    for _ in range(20):
        alpha = -rnd.randint(5,20)
        beta = rnd.randint(5,20)
        print(esperanceT(alpha,beta), -3*alpha*beta/2)
```

On teste tout cela :

```
227.8946 243.0
277.5667 306.0
204.7537 216.0
150.7025 153.0
189.6812 192.0
79.7874 81.0
132.3493 132.0
310.216 370.5
235.0867 247.5
136.2409 135.0
185.1341 189.0
191.0773 199.5
333.727 432.0
337.5656 459.0
90.1634 90.0
37.0103 37.5
267.4397 292.5
148.4005 148.5
295.5236 336.0
259.1824 285.0
```

b) ...

c) Pour tout $t \in]-1,1[$, on a

$$G'_{S_N}(t) = G'_{X_1}(t).G'_N(G_{S_N}(t))$$

Lorsque t tend vers 1, on a $\lim_{t\to 1} G_{S_N}(t) = G_{S_N(1)} = 1$ et puisque $Xi1$ et S_N sont d'espérance finie, $\lim_{t\to 1} G'_{X_1}(t) = \mathbb{E}(X_1)$ et de même pour G'_N. On en déduit que $\lim_{t\to 1} G'_{S_N}(t) = \mathbb{E}(X_1).\mathbb{E}(S_N)$ et ainsi l'existence de l'espérance et sa valeur.

d) Soit M tel que $|X_1| \leqslant M$. On a alors $|S_T| \leqslant \sum_{k=1}^{T} |X_k| \leqslant M.T$. Puisque T est d'espérance finie, on en déduit que S_T l'est également. On choisit alors $\beta = -\alpha = M$.

On note Y la variable aléatoire $Y = \sum_{j=1}^{+\infty} X_j\left(1 - \mathbf{1}_{T<j}\right)$. Soit $\omega \in \Omega$. On a

$$Y(\omega) = \sum_{j=1}^{+\infty} X_j(\omega)\left(1 - \mathbf{1}_{T<j}(\omega)\right) = \sum_{j=1}^{+\infty} X_j(\omega)\mathbf{1}_{T\geqslant j}(\omega)$$

Or $\mathbf{1}_{T\geqslant j}(\omega)$ vaut 1 lorsque $T(\omega) \geqslant j$ et 0 si $j > T(\omega)$. Ainsi $Y(\omega) = \sum_{j=1}^{T(\omega)} X_j(\omega) = S_T(\omega)$. Au passage, on a prouvé que la somme était finie et que Y a bien un sens.

L'événement $(T < j)$ se décompose en $(T = 0) \uplus (1 \leqslant T \leqslant j-1)$. **Je suppose** que l'événement $T = 0$ est négligeable (ce qui doit être le cas sauf si X_k est presque sûrement nulle). Ce dernier événement fait apparaître une réunion des événements $(S_k \leqslant \alpha) \bigcup (S_k \geqslant \beta)$ pour $k \in [\![1; j-1]\!]$ et donc dépend uniquement des variables aléatoires $X_1, \ldots, X_{j-1}$. Il est donc indépendant de X_j. On peut alors écrire

$$\begin{aligned}\mathbb{E}\left(\sum_{j=1}^{+\infty} X_j\left(1 - \mathbf{1}_{1\leqslant T<j}\right)\right) &= \sum_{j=1}^{+\infty} \mathbb{E}\left(X_j\right)\left(1 - \mathbb{E}\left(\mathbf{1}_{1\leqslant T<j}\right)\right) = \sum_{j=1}^{+\infty} \mathbb{E}\left(X_j\right)\left(1 - \mathbb{P}\left(1 \leqslant T < j\right)\right) \\ &= \sum_{j=1}^{+\infty} \mathbb{E}\left(X_j\right)\left(1 - \mathbb{P}\left(T < j\right)\right) = \sum_{j=1}^{+\infty} \mathbb{E}\left(X_1\right)\mathbb{P}\left(T \geqslant j\right) = \mathbb{E}(X_1)\mathbb{E}(T)\end{aligned}$$

Puisque $\mathbf{1}_{1\leqslant T<j} + \mathbf{1}_{T=0} = \mathbf{1}_{T<j}$, la différence entre l'espérance calculée et celle que l'on cherche est $\sum_{j=1}^{+\infty} \mathbb{E}\left(X_j \mathbf{1}_{T=0}\right)$. Puisque $\left|X_j \mathbf{1}_{T=0}\right| \leqslant M.\mathbf{1}_{T=0}$, on a $\left|\mathbb{E}\left(X_j \mathbf{1}_{T=0}\right)\right| \leqslant M\mathbb{P}(T=0) = 0$. Finalement

$$\mathbb{E}(S_T) = \mathbb{E}\left(\sum_{j=1}^{+\infty} X_j \left(1 - \mathbf{1}_{T<j}\right)\right) = \mathbb{E}\left(\sum_{j=1}^{+\infty} X_j \left(1 - \mathbf{1}_{1\leqslant T<j}\right)\right) = \mathbb{E}(S_T)\mathbb{E}(X_1).$$

Remarque : ça aurait certainement été plus simple de noter l'événement $T = +\infty$...

Exercice 27

a)

i)

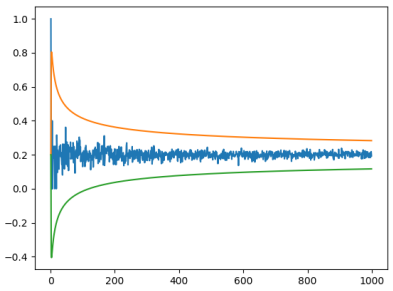
```
import numpy.random as alea
import matplotlib.pyplot as plt
import numpy as np

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 coïncées 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 $\lambda = \dfrac{1-x}{2} \in [0,1]$ avec $1-\lambda = \dfrac{1+x}{2}$:

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

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

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

en passant aux espérances, avec $\mathbb{E}(X)=0$, on obtient $\mathbb{E}\left(\exp(tX)\right) \leqslant \dfrac{e+e^{-t}}{2} = \operatorname{ch} t$.

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

c)

i) Par indépendance,

$$\mathbb{E}\left(\exp(tS_n)\right) = \prod_{k=1}^{n} \mathbb{E}\left(\exp(tX_i)\right) = \prod_{k=1}^{n} \mathbb{E}\left(\exp((a_i t)(X_i/a_i))\right)$$

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

$$\mathbb{E}\left(\exp((a_i t)(X_i/a_i))\right) \leqslant \exp((a_i t)^2/2)$$

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 :

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

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

iv)

Exercice 28

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 $1/\mathbb{E}(Y1)$.

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

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

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

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

et puisque les Y_j suivent toutes la même loi,

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

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,

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

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 $u(t) = f_1 t + f_2 t^2 = (1-p)t + pt^2$. On en déduit que $f(t) = \dfrac{1}{1 - (1-p)t - pt^2} = \dfrac{1}{(1-t)(1+pt)}$. On décompose en éléments simples, puis on somme :

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

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

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

vi) On note $k_1, \ldots, k_n$ 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 29

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 $\frac{6}{\pi^2} \equiv 0.6079271$:

```
>>> 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 $p\ell \leqslant n$, soit $p \leqslant \frac{n}{\ell}$. 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 :

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

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

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

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

f) une petite question toute simple... évidemment, on s'intéresse à $\lim_{n\to+\infty} \frac{1}{n^2}\operatorname{card}(A_n)$. On a $\frac{1}{n^2}\operatorname{card}(A_n) = \sum_{d=1}^{n} \frac{\mu(d)}{n^2}\left\lfloor \frac{n}{d} \right\rfloor^2$. Lorsque n tend vers $+\infty$, on a (par encadrement simple), $\frac{1}{n^2}\left\lfloor \frac{n}{d} \right\rfloor^2 \underset{n\to+\infty}{\sim} \frac{1}{d^2}$. 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 le 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 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
$$\lim_{n\to+\infty}\frac{1}{n^2}\sum_{d=1}^{n}\mu(d)\left\lfloor\frac{n}{d}\right\rfloor^2=\sum_{d=1}^{+\infty}\frac{\mu(d)}{d^2}.$$
On étudie la différence
$$S_n=\sum_{d=1}^{n}\mu(d)\left(\frac{1}{n^2}\left\lfloor\frac{n}{d}\right\rfloor^2-\frac{1}{d^2}\right)$$
On a ensuite $0\leqslant\frac{n}{d}-1<\left\lfloor\frac{n}{d}\right\rfloor\leqslant\frac{n}{d}$ et ainsi $\left(\frac{n}{d}-1\right)^2<\left\lfloor\frac{n}{d}\right\rfloor^2\leqslant\frac{n^2}{d^2}$, puis
$$\frac{1}{n^2}\left(\frac{n^2}{d^2}-2\frac{n}{d}+1\right)-\frac{1}{d^2}\leqslant\left(\frac{1}{n^2}\left\lfloor\frac{n}{d}\right\rfloor^2-\frac{1}{d^2}\right)\leqslant 0$$
On a ainsi $\left|\frac{1}{n^2}\left\lfloor\frac{n}{d}\right\rfloor^2-\frac{1}{d^2}\right|\leqslant\frac{2}{nd}$ et ainsi
$$|S_n|\leqslant\sum_{d=1}^{n}\left|\frac{1}{d^2}-\frac{1}{n^2}\left\lfloor\frac{n}{d}\right\rfloor^2\right|\leqslant\frac{2}{n}\sum_{d=1}^{n}\frac{1}{d}$$
On sait que $\sum_{d=1}^{n}\frac{1}{d}\underset{n\to+\infty}{\sim}\ln n$, ce qui donne $\lim_{n\to+\infty}S_n=0$.