Solutions

Exercice 3

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 $P = (X-\lambda)^2Q_1$ 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 $TP' = (X-\lambda)^2Q_1Q_2 = PQ_2$ 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 5

a)
```python
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 $M_nU_n(x_0)$ (avec i allant de 0 à n) est

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

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 $M_n^2U_n(x_0) = (1+x_0)^nM_nU_n(y_0) = (1+x_0)^n(1+y_0)^nU_n(z_0) = (1+x_0)^n\left(\frac{2}{1+x_0}\right)^n U_n(x_0) = 2^nU_n(x_0)$. Ainsi M_n^2X et $2^nI_{n+1}X$ 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^nI_{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 $y_i - y_j = \dfrac{1+j-i-ij-1-i+j+ij}{(1+i)(1+j)} \dfrac{2(j-i)}{(1+i)(1+j)} = -\dfrac{x_i - x_j}{(1+i)(1+j)}$. On calcule alors le déterminant de Vandermonde (la taille est $n+1$, le nombre de terme est $\dfrac{(n+1)n}{2}$) :

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

avec

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

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

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

et ce terme est aussi $P^2\prod_{i=0}^{n}(1+i)(1+i) = (n+1)!^2P^2$. 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 $M_nV_n(x) = W_n$. 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

$$\det(M_nV_n(x)) = \det(M_n)\det(V_n(x)) = 1^n2^n\dots(n+1)^n\det(V_n(y)) = ((n+1)!)^n\det V_n(y),$$

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 6

a)

```python
import numpy as np
import numpy.linalg as alg

def tournoi(n):
    A = np.zeros((n,n))
    for i in range(n):
        for j in range(i):
            t = np.random.rand()
            if t < 0.5:
                A[i, j] = 1
                A[j, i] = -1
            else:
                A[i, j] = -1
                A[j, i] = 1
    return A
```

b)

```python
for n in range(3,8):
    print(" n = ",n)
    for i in range(10):
        print(alg.det(tournoi(n)),end=" ")
    print()
```

Les déterminants semblent nuls lorsque n est impair et un entier (impair) au carré lorsque n est pair.

c) Une matrice de tournoi est antisymétrique, donc $\det(A) = \det(-{}^tA) = (-1)^n\det(A)$. Si n est impair, on a donc $\det(A) = 0$.

d)

i) Avec les opérations $C_1 \leftarrow C_1 - C_2 - \dots - C_n$ puis $C_j \leftarrow C_j - C_1$, on trouve que $\det(J_n - I_n) = (-1)^{n-1}(n-1)$.

ii) On travaille dans $\mathbb{Z}/2\mathbb{Z}$. Puisque le déterminant est une somme de produit des coefficients et que la congruence modulo 2 est compatible avec ces deux opérations, on a $\det M \equiv \det N[2]$.

iii) Les coefficients de $A-(J_n-I_n)$ sont tous pairs, donc $\det A = \det(J_n - I_n)$ modulo 2 donc vaut 1. Le déterminant est un entier impair... visiblement on n'aura pas mieux ainsi. Peut-être est-ce le résultat attentu (ou pas?)

Exercice 7

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 8

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 $R_n'(x) = -\displaystyle\sum_{i=0}^{n} \frac{1}{(x-i)^2} < 0$ sur $]k,k+1[$, R_n est strictement décroissante et comme $\lim\limits_{x\to k^+} R_n(x) = +\infty$, et $\lim\limits_{x\to(k+1)^-} R_n(x) = -\infty$, 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 9

a) On propose deux méthodes

```
import numpy as np

def F(n):
    a,b = 0,1
    for i in range(n):
        a,b = b,a+b
    return a

def A(n):
    f = np.vectorize(lambda i,j : F(i+j))
    return np.fromfunction(f,(n,n),dtype="int")
```

ainsi que

```
def A(n):
    A = np.zeros((n,n),dtype='int')
    A[0,1] = 1
    A[1,0] = 1
    A[1,1] = 2
    for i in range(2,n):
        A[0,i] = A[0,i-1]+A[0,i-2]
        A[1,i] = A[1,i-1]+A[1,i-2]
        # ou A[0:2,i]= A[0:2,i-1]+A[0:2,i-2]
    for j in range(2,n):
        A[j] = A[j-1]+A[j-2]
    return A
```

b) plus rapide :

```
def valeurs(n):
    return np.linalg.eigvals(A(n))
```

ou, pour un peu plus de lisibilité (les valeurs propres sont réelles puisque A_n est symétrique réelle), on peut prendre les parties réelles ou les modules

```
def valeurs(n):
    return np.real(np.linalg.eigvals(A(n)))
```

c) La matrice A_n est de rang 2 : à partir de la colonne 3, on a $C_i = C_{i-1} + C_{i-2}$ et les colonnes 1 et 2 ne sont pas proportionnelles. On en déduit que 0 est de multiplicité $n-2$ (c'est à la fois l'ordre de multiplicité et la dimension du sous-espace propre)

d) La matrice A_n a donc 3 valeurs propres : 0 de multiplicité $n-2$, α_n et β_n (éventuellement égales mais non nulles). La trace de A_n est strictement positive donc les deux valeurs propres restantes ne peuvent pas être strictement négatives. On va montrer qu'elles ne sont pas toutes les deux dans $\mathbb{R}_+^*$ et le résultat admis au début nous invite fortement à calculer ${}^tXA_nX$. On va exploiter la structure « croissante » des matrices. On retrouve A_2 dans le coin de chaque matrice A_n. On a $A_2 = \begin{pmatrix} 0 & 1 \\ 1 & 2 \end{pmatrix}$. Son polynôme caractéristique est $X^2 - 2X - 1$ et A_2 possède une racine strictement positive β_2 et une strictement négative α_1. Il existe X_2 unitaire de taille 2 tel que $A_2X_2 = \alpha_2 X_2$ et ${}^tX_2A_2X_2 = \alpha_2 < 0$. On considère le vecteur X_n de taille n obtenu en ajoutant $n-2$ zéros à la suite des deux coordonnées de X_2. On vérifie alors que ${}^tX_nA_nX_n = {}^tX_2A_2X_2 < 0$ (écrire tout cela avec des blocs). La matrice A_n n'est donc pas positive et admet une valeur propre strictement négative.

e) Rien d'évident a priori. On a un ordinateur, on peut au moins essayer de conjecturer les résultats...

```python
def b(n):
    return np.max(valeurs(n))
def a(n):
    L = np.sort(valeurs(n))
    return L[0]
```

```
>>> [a(n) for n in range(2,11)]
[-0.6180339887498949,
 -0.64575131106458983,
 -0.70820393249936864,
 -0.7119144780585055,
 -0.72135954999579366,
 -0.72189169131898268,
 -0.72327892872129851,
 -0.72335636314029272,
 -0.7235589623033718]
```

et

```
>>> [b(n) for n in range(2,11)]
[1.6180339887498949,
 4.6457513110645881,
 12.708203932499371,
 33.711914478058 51,
 88.721359549995782,
 232.72189169131909,
 609.72327892872136,
 1596.7233563631401,
 4180.7235589623033]
```

On se rend compte que b_n à l'air géométrique, de raison ρ^2 où ρ est le nombre d'or $\rho = \frac{1+\sqrt{5}}{2}$ (on sait que F_n s'exprime avec ρ^n et ρ^{-n}).

- d'après le cours (un peu hors programme), on a $\beta_n = \max\{{}^tXA_nX, \|X\| = 1\}$ et $\alpha_n = \min\{{}^tXA_nX, \|X\| = 1\}$.
- lorsque X décrit l'ensemble des vecteurs unitaires de taille n, alors le vecteur de taille $n+1$, $X_{n+1} = \begin{pmatrix} X_n \\ 0 \end{pmatrix}$ est un vecteur unitaire de taille $n+1$. Un calcul simple donne ${}^tX_{n+1}A_{n+1}X_{n+1} = {}^tXA_nX$ et en prenant X qui maximise la quantité de droite $\beta_{n+1} \geqslant {}^tX_{n+1}A_{n+1}X_{n+1} = {}^tXA_nX = \beta_n$. La suite β_n est croissante. De même α_n est décroissante.
- On a $\alpha_n + \beta_n = \operatorname{tr}(A_n) = F_0 + F_2 + \ldots + F_{2n-2}$. Or

$$F_2 + F_4 + \ldots + F_{2n-2} = (F_0 + F_1) + (F_2 + F_3) + \ldots + F_{2n-4} + F_{2n-3}$$

  Une récurrence simple montre que cette somme vaut $F_{2n-1} - 1$ et finalement $\alpha_n + \beta_n = F_{2n-1} - 1$. Ainsi $\lim\limits_{n\to+\infty} \alpha_n + \beta_n = +\infty$.
- Si on montre que α_n est minorée et donc convergente, on a aura obtenu en même temps que $\beta_n \sim F_{2n-1}$.
- Puisque $\alpha_n < 0$, on a $\beta_n + \alpha_n \leqslant \beta_n$ et on en déduit facilement que β_n tend vers $+\infty$.

Exercice 10

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

```python
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

```python
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

```python
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 $F(p^k)=\frac{p^{k+1}-1}{p-1}-\frac{p^k-1}{p-1}=p^k$. 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 14

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).

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 15

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

- on a $|u_{n+2}| \leqslant \frac{1}{6}|u_n|$. Ainsi $|u_{2n}| \leqslant \frac{1}{6^n}|u_0|$ et $|u_{2n=1}| \leqslant \frac{1}{6^n}|u_1|$. 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 $0 \leqslant w_{n+1} \leqslant \frac{1}{2}w_n$ et par récurrence $0 \leqslant w_n \leqslant \frac{1}{2^n}w_0$. La suite w tend vers 0 et on a $|u| \leqslant w$ et $|v| \leqslant w$.

e) Supposons que $(x_n)_{n\geqslant n_0}$ soit croissante. Puisque $y_{n+1} = \sqrt{7+x_n}$, on aurait $(y_n)_{n\geqslant n_0+1}$ croissante. Or $x_{n+1} = \sqrt{7-y_n}$ et $(x_n)_{n\geqslant n_0+2}$ serait décroissante (même situation pour les autres cas de figures). On peut utiliser les majorations précédentes. On a $|u_0| = 2$ et $|u_{2n}| \leqslant \frac{2}{12^n}$ donc pour $n = 4$, on a $|u_{2n}| \leqslant 10^{-3}$. On a $|u_1| = \sqrt{7} - 2 \leqslant 1$ et $|u_{2n+1}| \leqslant \frac{1}{12^n}$. Pour $n \geqslant 3$, on a $|u_{2n+1}| \leqslant 10^{-3}$. On est donc certain que $|x_n - 2| \leqslant 10^{-3}$ si $n \geqslant 6$. 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 16

a)

```
import numpy as np
import matplotlib.pyplot as plt
t = np.linspace(-np.pi,np.pi,100)
x = 3*np.cos(t)
y = 2*np.sin(t)
plt.axis('equal')
plt.plot(x,y)
```

b) puisque ce n'est pas précisé, on utilise les fonctions toutes prêtes...

```
import numpy as np
import scipy.integrate as integr
def L(a,b):
    # on suppose que a>b est vérifié
    c = np.sqrt(a*a-b*b)
    mu = c/a
    def f(t):
        return np.sqrt(1+mu*mu*(np.sin(t))**2)
    # ou
    # f = lambda t : np.sqrt(1+mu*mu*(np.sin(t))**2)
    return 4*a*integr.quad(f,0,np.pi/2)[0]
```

c) on ne demande pas forcément d'aller très loin, donc on peut créer une fonction factorielle puis une fonction de calcul de α_k. On peut exprimer une relation de récurrence sur les coefficients :

$$\alpha_{k+1} = \frac{1}{(2k+1)4^{k+1}} \frac{(2k+2)!}{(k!)^2} = \frac{2k-1}{4.(2k+1)} \frac{(2k+2)(2k+1)}{(k+1)^2} \alpha_k = \frac{2k-1}{2(k+1)} \alpha_k.$$

ce qui donne $\alpha_0 = -1$ et $\alpha_k = \frac{2k-3}{2k}\alpha_{k-1}$ si $k \geqslant 1$. Au choix :

```
def a(k):
    if k==0:
        return -1
    else:
        return (2*k-3)*a(k-1)/(2*k)

def a2(k):
    a = -1
    for i in range(1,k+1):
        a *= (2*i-3)/(2*i)
    return a
```

d) i) Par récurrence sur k... on a $g^{(k+1)}(x) = -k!\alpha_k(k-\frac{1}{2})(1-x)^{-k-1/2} = -(k+1)!\frac{\alpha_k(2k-1)}{2(k+1)}(1-x)^{-(k+1)+1/2} = -(k+1)!\alpha_{k+1}(1-x)^{-(k+1)+1/2}$.

ii) C'est une formule de Taylor avec reste intégrale : on a $\frac{g^{(k)}(0)}{k!} = -\alpha_k$ et

$$g(x) - \sum_{k=0} \frac{g^{(k)}(0)}{k!} x^n = \frac{1}{n!}\int_0^x (x-t)^n g^{(n+1)}(t)\,dt = -(n+1)\alpha_{n+1}\int_0^x (x-t)^n(1-t)^{-n-1/2}\,dt,$$

c'est-à-dire exactement le résultat souhaité.

iii) On majore. On a, si $x \in [0,1[$ et $t \in [0,x]$, $x<1$ donc $0 \leqslant x-t < 1-t$ et $0 \leqslant \frac{x-t}{1-t} < 1$. Ainsi

$$\left|g(x) + \sum_{k=0}^{n} \alpha_k x^k\right| \leqslant (n+1)\alpha_{n+1}\int_0^x \frac{dt}{\sqrt{1-t}} = \frac{2n-1}{2}\alpha_n\left[-2\sqrt{1-t}\right]_0^x = \frac{1}{2^{2n+1}}\binom{2n}{n}(2-2\sqrt{1-x}) \leqslant \frac{2}{2^{2n+1}}\binom{2n}{n}.$$

Exercice 17

a)

```
def euler(a,T,N):
    h = T/N
    X = [0] ; x = 0
    Y = [a] ; y = a
    for i in range(N):
        y = y + h*(np.cos(x)+np.cos(y))
        x = x + h
        Y.append(y)
        X.append(x)
    plt.plot(X,Y)
    plt.show()

for a in [-3,-1,0,0.5,1,2,3,3.5,4,4.2,4.3,5]:
    X,Y = euler(a,5,100)
    plt.plot(X,Y)
plt.show()
```

b) Pour x, y deux fonctions de E, on a, pour tout $t \in [0,T]$, $\phi(x)(t) - \phi(y)(t) = \int_0^t \big(\cos(x(s)) - \cos(y(s))\big)\,ds$ et

$$\left|\phi(x)(t) - \phi(y)(t)\right| \leqslant \int_0^t \left|\cos(x(s)) - \cos(y(s))\right| ds \leqslant \int_0^t \left|x(s) - y(s)\right| ds \leqslant t\left\|x-y\right\|_\infty,$$

car $|\cos(a) - \cos(b)| \leqslant |a,b|$ pour tous $a, b \in \mathbb{R}$. On montre alors le résultat par récurrence sur n :

$$\left|\phi^{n+1}(x)(t) - \phi^{n+1}(y)(t)\right| \leqslant \int_0^t \left|\cos(\phi^n(x)(s)) - \cos(\phi^n(y)(s))\right| ds \leqslant \int_0^t \left|\phi^n(x)(s) - \phi^n(y)(s)\right| ds \leqslant \left\|x-y\right\|_\infty \int_0^t \frac{s^n}{n!}\,ds = \frac{t^{n+1}}{(n+1)!}\left\|x-y\right\|_\infty.$$

c) On applique la relation avec x et $y=\phi(x)$. Cela donne, pour tout $t\in[0,T]$ et tout $n\in\mathbb{N}$,

$$\left|\phi^{n+1}(x)(t)-\phi^n(x)(t)\right|\leqslant\frac{t^n}{n!}\left\|x-\phi(x)\right\|_\infty\leqslant\frac{T^n}{n!}\left\|x-\phi(x)\right\|_\infty.$$

On note $f_n=\phi^n(x)$ et $g_n=f_{n+1}-f_n$. On a alors $\left\|g_n\right\|_{\infty,[0,T]}\leqslant\frac{T^n}{n!}\left\|x-\phi(x)\right\|_\infty$. Ainsi la série de fonctions $\sum g_n$ converge normalement sur $[0,T]$ et donc uniformément. On a $\sum_{k=0}^{n-1}g_k=f_n-x$. La suite de fonctions (f_n) converge donc uniformément sur $[0,T]$ vers une fonction qu'on note z qui est bien un élément de E puisque chacune des fonctions f_n est continue sur $[0,T]$ (et convergence uniforme).
La relation $\left|\phi(x)(t)-\phi(y)(t)\right|\leqslant t\left\|x-y\right\|_\infty$ donne $\left\|\phi(x)-\phi(y)\right\|_\infty\leqslant T\left\|x-y\right\|_\infty$ et l'application ϕ est donc continue de $(E,\|.\|_\infty)$ dans lui même. Puisque (f_n) tend vers z pour $\|.\|_\infty$, on a $(\phi(f_n))$ qui tend vers $\phi(z)$... mais également vers z en tant que suite extraite de (f_n) (on a $f_{n+1}=\phi(f_n)$). Finalement $z=\phi(z)$. Cela signifie que, pour tout $t\in[0,T]$, on a $a+\sin t+\int_0^t\cos(x(s))\,ds=z(t)$. La valeur en 0 donne $z(0)=a$. Par théorème fondamental, z est alors dérivable et sa dérivée vérifie, pour tout $t\in[0,T]$, $z'(t)=\cos t+\cos(z(t))$. La fonction z a les mêmes propriétés que x_a.

Exercice 18

a) i) On a $a_{n+1}=\frac{(2n+2)(2n+1)}{4(n+1)^2}a_n=\frac{2n+1}{2n+2}a_n$.

ii) On part de $a_0=1$ puis on utilise la relation de récurrence :

```
def seqa(n):
    a = 1
    L = [1]
    for k in range(n):
        a = a * (2*k+1)/(2*k+2)
        L.append(a)
    return L
```

iii)

```
def somp(n,x):
    L = seqa(n) # on ne va pas recalculer à chaque fois
    S = 0
    for i in range(n+1):
        S += L[i]/(i+x)
    return S
```

iv)

```
x = np.linspace(-4.001,4.001,1000)
y = [somp(100,a) for a in x]
plt.ylim(-5,5)
plt.plot(x,y)
plt.show()
```

b) Soit $u_n:x\mapsto\frac{a_n}{n+x}$ pour $n\in\mathbb{N}$. Chacune de ces fonctions est $\mathscr{C}^\infty$ sur $A=\mathbb{R}\backslash\mathbb{Z}$.

- existence (convergence simple) : c'est le plus délicat. On cherche un équivalent de a_n lorsque n tend vers $+\infty$ à l'aide de la formule de Stirling. On a

$$a_n=\frac{(2n)!}{4^n(n!)^n}\underset{n\to+\infty}{\sim}\frac{\sqrt{4\pi n}(2n/e)^{2n}}{4^n(2\pi n)(n/e)^{2n}}=\frac{\sqrt{4\pi n}}{2\pi n}=\frac{1}{\sqrt{n\pi}}.$$

On en déduit que $u_n(x)\underset{n\to+\infty}{\sim}\frac{1}{\sqrt{\pi}n^{3/2}}$ et la convergence simple de $\sum u_n$ sur A.

- On montre par une récurrence simple que $u_n^{(p)}(x)=\frac{(-1)^p\,p!\,a_n}{(n+x)^{p+1}}$. On se place sur un intervalle $I=[N+\varepsilon,N+1-\varepsilon]$ (ou $\varepsilon\in]0,1/2[$). On a, pour tout $x\in I$,

$$|u_k^{(p)}(x)|\leqslant p!a_k\max\left(\frac{1}{|k+N+\varepsilon|^{p+1}},\frac{1}{|k+N+1-\varepsilon|^{p+1}}\right),$$

ce qui donne $\left\||u_k^{(p)}|\right\|_{\infty,I}\underset{k\to+\infty}{\sim}\frac{p!}{\sqrt{\pi}k^{p+3/2}}$. La série de fonctions $\sum u_k$ converge normalement sur tout segment de $\mathbb{R}\backslash\mathbb{Z}$ donc S est continue sur $\mathbb{R}\backslash\mathbb{Z}$. On peut remarquer qu'on peut faire ce raisonnement directement sur les intervalles $[\varepsilon,+\infty[$ ce qui donne la classe $\mathscr{C}^\infty$ sur $\mathbb{R}_+^*$.

c) On a $\frac{1}{\sqrt{1-t}}=(1-t)^{-1/2}=1+\sum_{k=1}^{+\infty}b_k(-t)^k$ où

$$b_k=\frac{(-1/2)(-3/2)(-(k-1/2)}{k!}=(-1)^k\frac{1.3\cdots(2k-1)}{2^k\,k!}=(-1)^k\frac{(2k)!}{(2^k\,k!)^2}=(-1)^ka_k.$$

Puisque $a_0=1$, on en déduit que $\frac{1}{\sqrt{1-t}}=\sum_{n=0}^{+\infty}a_nt^n$ pour tout $t\in]-1,1[$.

d) Évidemment on se doute qu'il y a une intégration terme à terme... On a, pour tout $t\in]0,1[$, $\dfrac{t^{x-1}}{\sqrt{1-t}}=\sum_{n=0}^{+\infty}a_n t^{n+x-1}$. On note $f_n(t)=a_n t^{n+x-1}$. Chaque fonction est continue et intégrable sur $]0,1]$ (puisque $n+x-1\geqslant x-1>-1$), la série de fonctions $\sum f_n$ converge simplement sur $]0,1[$ et a pour somme $F(t)=\sum_{n=0}^{+\infty}f_n(t)=\dfrac{t^{x-1}}{\sqrt{1-t}}$, continue sur .$]0,1[$. De plus $\int_0^1|f_n(t)|\,dt=\dfrac{a_n}{n+x}$ et $\sum\dfrac{a_n}{n+x}$ converge. On en déduit que F est intégrable sur $]0,1[$ et que, pour tout $x>0$, $S(x)=\displaystyle\int_0^1\frac{t^{x-1}}{\sqrt{1-t}}\,dt$.

Exercice 19

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

$$\binom{n}{n}=\frac{n(n-1)\dots(n-k+1)}{k(k-1)\dots1}.$$

```
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 $\left|\dfrac{B_n}{n!}\right|\leqslant1$ et que le rayon de convergence de la série entière donnée est au moins 1.

d) On note $S(x)=\displaystyle\sum_{n=0}^{+\infty}\frac{B_n}{n!}x^n$. 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 20

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))
```

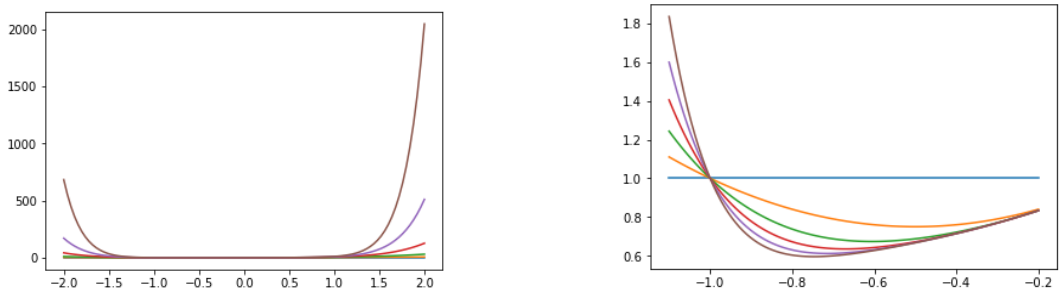

FIGURE 1 – Tracé des P_n, pour n de 0 à 5

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

$$2na_n^{2n+1} - (2n+1)a_n^{2n} + 1 = 0$$

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

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

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

Exercice 21

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))
  1.599999999999999
```

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 $|f_p(t)|\leqslant|\lambda|^{p+1}$. 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 $\lambda\neq1$. 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 $\lim_{m\to+\infty}S_m(t)\,dt=\frac{\lambda}{1-\lambda t}=S(t)$ 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 $|S_m(t)|\leqslant\frac{2}{|1-\lambda t|}$. 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

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

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 $\lambda\neq1$.

c) On le fait en deux étapes :

- si on suppose que $\rho(A)>1$: on montre que $\operatorname{tr}(A^k)$ 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 $\exp\left(\sum_{p=1}^{+\infty}\frac{\operatorname{tr}(A^p)}{p}\right)=\exp\left(-\int_0^1\frac{P_A'(t)}{P_A(t)}\,dt\right)$. 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 $\mathscr{C}^1$ sur $[0,1]$ et

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

  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 22

a) on utilise le module d'intégration numériqueme

```
import numpy as np
import scipy.integrate as integre

def f(x):
    return np.cos(x)

def T(f,x):
    return 1-integre.quad(lambda t : (x-t)*f(t),0,x)[0]
```

b) soit $H = T - 1$. Soient $f, g \in E$ et $\lambda \in \mathbb{R}$. Pour tout $x \in \mathbb{R}$, on a

$$H(f+\lambda g)(x) = -\int_0^x (x-t)(f+\lambda g)(t)\,dt = -\int_0^x (x-t)f(t)\,dt - \lambda\int_0^x (x-t)g(t)\,dt = (H(f)+\lambda H(g))(x)$$

On a bien $H(f+\lambda g) = H(f) + \lambda H(g)$ et $H = T - 1$ est linéaire. De plus, si $g = T(f)$ est dans Im T, alors pour tout $x \in \mathbb{R}$,

$$g(x) = -x\int_0^x f(t)\,dt + \int_0^x tf(t)\,dt$$

Puisque les primitives $x \mapsto \displaystyle\int_0^x f(t)\,dt$ et $x \mapsto \displaystyle\int_0^x tf(t)\,dt$ sont de classe $\mathscr{C}^1$ sur $\mathbb{R}$, la fonction g l'est aussi. Ainsi Im T est contenue dans $\mathscr{C}^1(\mathbb{R},\mathbb{R})$.

c)

```
x = np.linspace(-5,5,100)
y0 = np.cos(x)

def f1(x):
    return T(np.cos,x)
y1 = [f1(t) for t in x]

plt.plot(x,y0,x,y1)
plt.show()
```

on ne voit qu'une courbe : les deux sont confondues. Aurait-on $T(\cos) = \cos$?

d) Soit f une telle fonction. Pour tout $x \in \mathbb{R}$, on a $1 - x\displaystyle\int_0^x f(t)\,dt + \int_0^x tf(t)\,dt = f(x)$. Le premier membre étant de classe $\mathscr{C}^1$ par rapport à x, f l'est et en dérivant on obtient $1 - xf(x) - \displaystyle\int_0^x f(t)\,dt + xf(x) = f'(x)$. On obtient l'équivalence en ajoutant une condition d'égalité pour la valeur en 0 de chaque membre de l'égalité de départ (on a $f = g$ si et seulement si $f' = g'$ et $f(0) = g(0)$ pour des fonctions de classe $\mathscr{C}^1$). Cela donne $f(0) = 1$. La fonction f vérifie $T(f) = f$ si et seulement si $f(0) = 1$ et pour tout $x \in \mathbb{R}$, $-\displaystyle\int_0^x f(t)\,dt = f'(x)$. On recommence pour avoir $f(0) = 1$, $f'(0) = 0$ et $-f(x) = f''(x)$. Toutes ces conditions donnent la fonction cosinus (avec équivalence avec $T(f) = f$).

e) On a, pour $x \in \mathbb{R}$, $f_{n+1}(x) - f_n(x) = -\displaystyle\int_0^x (x-t)\big(f_n(t) - f_{n-1}(t)\big)\,dt$. Soit $A > 0$. On travaille sur le segment $[-A, A]$. On note M un majorant de $|f_1 - f_0|$ sur ce segment. Pour tout $x \in [-A, A]$, on a (car $x - t$ garde un signe constant)

$$\big|f_2(x) - f_1(x)\big| \leqslant \left|\int_0^x M|x-t|\,dt\right| = \left|\int_0^x M(x-t)\,dt\right| = M\frac{x^2}{2}.$$

puis

$$\big|f_3(x) - f_2(x)\big| \leqslant \left|\int_0^x |x-t||f_2(t) - f_1(t)\,dt\right| = \left|\int_0^x M(x-t)\frac{t^2}{2}\,dt\right| = M\left(\frac{x^4}{2.3} - \frac{x^4}{2.4}\right) = M\frac{x^4}{4!}.$$

On montre alors par récurrence que, pour tout $x \in [-A, A]$, $|f_{n+1}(x) - f_n(x)| \leqslant M\frac{x^{2n}}{(2n)!}$. C'est le même calcul :

$$\big|f_{n+2}(x) - f_{n+1}(x)\big| \leqslant \left|\int_0^x |x-t||f_{n+1}(t) - f_n(t)\,dt\right| = \left|\int_0^x M(x-t)\frac{t^{2k}}{(2k)!}\,dt\right| = M\left(\frac{x^{2k+2}}{(2k)!(2k+1)} - \frac{x^{2k+2}}{(2k)!.(2k+2)}\right) = M\frac{x^{2k+2}}{(2k+2)!}.$$

On a enfin $\|f_{n+1} - f_n\|_\infty \leqslant M\frac{A^{2n}}{(2n)!}$ et $\sum(f_{n+1} - f_n)$ converge normalement et uniformément sur $[-A, A]$. Puisque $f_n - f_0 = \displaystyle\sum_{k=0}^{n-1}(f_{k+1} - f_k)$, la suite de fonctions (f_n) converge uniformément sur $[-A, A]$. On note f sa limite qui est donc continue sur $[-A, A]$. On fixe $x \in [-A, A]$. On a

$$f_{n+1}(x) = 1 - \int_0^x (x-t)f_n(t)\,dt \qquad (*)$$

On a $|(x-t)f_n(t) - (x-t)f(t)| = |x-t||f_n(t) - f(t)| \leqslant A\|f_n - f\|_{\infty,[0,x]}$. La suite de fonctions $t \mapsto (x-t)f_n(t)$ converge donc uniformément vers la fonction $t \mapsto (x-t)f(t)$ sur $[0, x]$. En passant à la limite lorsque n tend vers $+\infty$, on obtient

$$f(x) = 1 - \int_0^x (x-t)f(t)\,dt = T(f)(x)$$

Ainsi (f_n) converge uniformément vers la fonction cosinus sur tout segment de $\mathbb{R}$.

Exercice 26

a)
```python
import numpy.random as rand

def suivant():
    return -1+2*rand.randint(0,2)

def simul():
    a = 0  # position du premier
    b = 1  # position du second
    n = 0  # numéro du lancer
    while a!=b:
        n += 1
        a = (a+suivant())%5
        b = (b+suivant())%5
    return n

def moyenne(N):
    S = 0
    for i in range(N):
        S += simul()
    return S/N
```

Quelques tests donnent une moyenne autour de 12.

b) On note A_n l'événement « les deux discoplanes sont dans les mains de deux joueurs voisins » et B_n l'événement « les discoplanes sont dans les mains de deux joueurs non voisins ». On note enfin C_n l'événement « les discoplanes sont entre les mains d'un même joueur ».

Exercice 27

a) i)
```python
import numpy as np

def h(x):
    if x==0:
        return 0
    else:
        return -x*np.log(x)

def entropie(p):
    n= len(p)
    e = 0
    for x in p:
        e += h(x)
    return e,e-np.log(n)
```

ii)
```python
def comb(n,p):
    N,D = 1,1 # numérateur, dénominateur
    for k in range(p):
        N = N * (n-k)
        D = D * (k+1)
    return N//D

def loibin(n,p):
    # strictement aucune optimisation, on n'a pas le temps
    L = [0]*(n+1)
    for k in range(n+1):
        L[k] = comb(n,k)*(p**k)*(1-p)**(n-k)
    return L

def loiuni(N):
    L = [0]*(N+1)
    for k in range(N+1):
        L[k]=1/(N+1)
    return L
```

iii)
```
>>> entropie(loibin(15,2/5))
 (2.0578509198462398, -0.71473780239354134)
>>> entropie(loiuni(15))
 (2.7725887222397811, 0.0)
```

iv) On doit calculer $\sum_{i=1}^{n}(-p_i \ln p_i)$. On pense rapidement à la concavité du logarithme : pour des poids p_i positifs de somme 1,

$$\text{si } x_1,\dots,x_n>0 \text{ alors } \ln\left(\sum_{i=1}^{n} p_i x_i\right) \geqslant \sum_{i=1}^{n} p_i \ln x_i,$$

en appliquant cela à $x_i = \frac{1}{p_i}$, ce qui donne $\sum_{i=1}^{n} p_i \ln(1/p_i) \leqslant \ln n$ (si l'un des p_i est nul alors $h(p_i)$ est nulle aussi). On a bien $h(X) \leqslant n$. Puisque la fonction est strictement convexe, on a égalité si et seulement si tous les x_i sont égaux, donc si et seulement si la loi est uniforme. De plus $h(X) \geqslant 0$ car tous les termes de la somme donnant $h(X)$ sont positifs.

v) répondu au dessus

b) On note N_k l'événement « boule noire au tirage k » et R_k pour une rouge. On a $\mathbb{P}(X=k) = \mathbb{P}\left(N_1 \cap N_2 \cap \ldots \cap N_{k-1} \cap R_k\right)$. La formule des probabilités totales donne

$$\mathbb{P}(X=k) = \mathbb{P}(N_1)\mathbb{P}_{N_1}(N_2)\mathbb{P}_{N_1\cap N_2}(N_2)\cdots\mathbb{P}_{N_1\cap\cdots\cap N_{k-2}}\left(N_{k-1}\right)\mathbb{P}_{N_1\cap\cdots\cap N_{k-1}}\left(R_k\right)$$

le conditionnement nous donne la structure de l'urne à un certain tirage et on obtient

$$\mathbb{P}(X=k) = \frac{1}{2}\frac{2}{3}\cdots\frac{k-1}{k}\frac{1}{k+1} = \frac{1}{k(k+1)} = p_k.$$

Puisque $\sum_{n\in\mathbb{N}^*} \frac{1}{k(k+1)} = 1$, on en déduit que $\mathbb{P}(X=0)=0$. On a $k\mathbb{P}(X=k) = \frac{1}{k+1}$ donc la variable aléatoire n'a pas d'espérance. En revanche, on a $-p_k\ln(p_k) = \frac{1}{k(k+1)}\ln(k(k+1)) \underset{k\to+\infty}{\sim} 2\frac{\ln k}{k^2}$ est le terme général d'une série convergente. La variable aléatoire admet une entropie.

c) Chaque $p_n = \mathbb{P}(X=n)$ est positif, et par téléscopage, $\sum_{n\geqslant 2} p_n = \ln 2.\left(\frac{1}{\ln 2} - 0\right) = 1$. On définit bien ainsi une loi. On a

$$np_n = n\ln 2\frac{\ln(n+1)-\ln n}{\ln n\ln(n+1)} = \ln 2(n\ln(1+\frac{1}{n}))\frac{1}{\ln n\ln(n+1)} \underset{n\to+\infty}{\sim} \frac{\ln 2}{\ln^2 n},$$

la variable aléatoire n'admet pas d'espérance. Le calcul précédent donne $p_n \underset{n\to+\infty}{\sim} \frac{\ln 2}{n\ln^2 n}$ et, puisque ce terme tend vers 0,

$$\ln p_n \underset{n\to+\infty}{\sim} \ln(\ln 2) - \ln n - 2\ln\ln n \underset{n\to+\infty}{\sim} -\ln n$$

et finalement $-p_n \ln p_n \underset{n\to+\infty}{\sim} \frac{\ln 2}{n\ln n}$, juste ce qu'il faut pour que la variable aléatoire n'admette pas d'entropie.

Exercice 29

a) i)

```python
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)

```python

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

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

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) $f(t)u(t)=\sum_{k=1}^{+\infty}c_kt^k$ 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 $f(t)u(t)=\sum_{k=1}^{+\infty}u_kt^k=f(t)-u_0=f(t)-1$.

v) On a $f_1=\mathbb{P}(Y_1=1)=1-p$ et $\mathbb{P}(Y_1=2)=p$. Ainsi $u(t)=f_1t+f_2t^2=(1-p)t+pt^2$. 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 $k_1,\dots,k_n$ les entiers pour lesquels $\mathbb{P}(Y_1=k)$ est non nul. On a $u(t)=\sum_{i=1}^{n}p_it^{k_i}$ où $p_i=f_{k_i}$ pour simplifier... à terminer