Planche 1 :

1) **a)**

```
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import random as rd
# Q1.a  :

# Ancienne version, encore indiquée dans le memento Centrale  :
#ax = Axes3D(plt.figure(),auto_add_to_figure=False)

# Nouvelle version  :
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

def f (x,y)  :
    return -2*x**2+y**2+3*x*y

f=np.vectorize(f)
X = np.arange(-2,2,0.02)
Y = np.arange(-2,2,0.02)
X, Y = np.meshgrid(X, Y)
Z = f(X, Y)
ax.plot_surface(X, Y, Z)
plt.show()
```

b) Gradient : $(-2x+3y, 3x+2y)$, un seul point critique : $(0,0)$. La Hessienne est $\begin{pmatrix} -2 & 3 \\ 3 & 2 \end{pmatrix}$, deux valeurs propres réelles de signe opposé, non nulles, donc pas d'extremum.

c) $f(1,y) = -2+y^2+3y$, qui prend avec le TVI toutes les valeurs de -2 à $+\infty$. Et $f(x,1) = -2x^2+1+3x$, qui prend toutes les valeurs de $-\infty$ à 1. Donc f est surjective de $\mathbb{R}^2$ dans $\mathbb{R}$.

2) **a)** Plus que classique, par analyse-synthèse. $M = \dfrac{M+M^\top}{2} + \dfrac{M-M^\top}{2}$.

b)

```
M = np.array([[-2,4],[-1,2]])

def sym(M)  :
    return (1/2)*(M+np.transpose(M))
```

3) **a)**

```
M = np.array([[291,332,413],[332,291,508],[291,413,332]])
S = sym(M)


def fi_M (X)  :
    Y = M.dot(X)
    return (np.transpose(X)).dot(Y)

def fi_S (X)  :
    Y = S.dot(X)
    return (np.transpose(X)).dot(Y)

def test ()  :
    A=np.array([[rd.randrange(50)] for j in range(3) ])
```

```
        return (fi_M(A), fi_S(A))
```

b) On conjecture que $\varphi_M = \varphi_S$.

4) Si $M = A + S$ avec A antisymétrique, alors $X^\top AX$ est un réel, donc $X^\top AX = \left(X^\top AX\right)^\top = X^\top A^\top X = -X^\top AX$ donc $X^\top AX = 0$. Alors $\varphi_m(X) = \varphi_A(X) + \varphi_S(X) = \varphi_S(X)$.

5) **a)** $\operatorname{tr} M = \operatorname{tr} A + \operatorname{tr} S = \operatorname{tr} S$ car une matrice antisymétrique a une diagonale nulle.

b) On sait que le spectre est réduit à 0.

c) S n'est pas nulle car M n'est pas antisymétrique.
Elle est diagonalisable dans $\mathbb{R}$ car symétrique réelle, et comme elle est non nulle, elle a une valeur propre λ non nulle.
Soit X est un vecteur propre pour λ, alors $\varphi_S(X) = \lambda \left\|X\right\|^2 \neq 0$.
Donc l'image est $\mathbb{R}$.

Planche 2 :

```
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import random as rd

# Q1  : les X_n ne prennent p.s. que deux valeurs  : 1 et 2, donc S_n div grossiérement.

def X (p)  :
    x = rd.random()
    if x < p  : return 1
    else  : return 2

def Y (k,p)  :
    S = 0
    n = 0
    while S < k  :
        S += X(p)
        n += 1
    return n

# Q2  :

def esp (k,p)  :
    N = 1000
    S = 0
    for n in range(N)  :
        S += Y(k,p)
    return S/N

P = [0.1,0.3,0.5,0.7,0.9]

def Q2 ()  :
    plt.clf()
    X = [i for i in range(1,100)]
    for p in P  :
```

```
        Y = [esp(k,p) for k in X]
        plt.plot(X,Y,label="p="+str(p))
    plt.legend()
    plt.savefig("Q2.png")

# Q3  :

# P(Yk=n)=P(Yk=n|Y(k-1)=n-1).P(Y(k-1)=n-1) + P(Yk=n|Y(k-2)=n-1).P(Y(k-1)=n-2)
#           +somme pour i < k-2 P(Yk=n|Y(k-i)<n-1).P(Y(k-i)=n-2)
# = P(Xk=1).P(Y(k-1)=n-1) + P(Xk=2).P(Y(k-2)=n-1) + 0

# Q4  :

# E(Yk) = somme P(Yk=n).n = somme pP(Y(k-1)=n-1).n + (1-p)P(Y(k-2)=n-1).n
# = somme pP(Y(k-1)=n-1).(n-1) + (1-p)P(Y(k-2)=n-1).(n-1) + pP(Y(k-1)=n-1)
# + (1-p)P(Y(k-2)=n-1) = pE(Y(k-1)) + (1-p)E(Y(k-2)) + p + (1-p)

# Q5  :

# On remarque que (k/(2-p)) est une suite solution de cette relation de récurrence.
# Équation homogène  : poca X**2-pX-(1-p), discriminant p**2+4(1-p)=(2-p)**2
# racines 1 et (p-1), solutions de la forme a + b(p-1)**n + n/(2-p)
# ce qui est équivalent à n/(2-p) car (p-1) \in ]-1,0[.

P2=[0.1,0.5,0.9]

def Q5 ()  :
    plt.clf()
    X = [i for i in range(1000,2000,100)]
    for p in P2  :
        cp = 1/(2-p)
        Y = [esp(k,p)-k*(cp) for k in X]
        plt.plot(X,Y,label="p="+str(p))
    plt.legend()
    plt.savefig("Q5.png")
```

Planche 3 :

RMS2023 1059. PYTHON .

```
import math

import matplotlib.pyplot as plt

import numpy
import numpy.polynomial as pol

def A(n) :
    U=pol.Polynomial([1])
    if n==0 :
        return(U)
```

```
    else :
        B=A(n-1).integ()
        C=B.integ(1,0)
        D=B-C(1)
    return(D)

def u(x) :
    return x /(math.exp(x)-1)

def w(x) :
    S=A(0)(0)
    for k in range (1,11)  :
        S+=A(k)(0)*x**k
    return S

X=numpy.linspace(-2,2,10)

Y=[u(x) for x in X]

Z=[w(x) for x in X]

plt.plot(X, Y)

plt.plot(X, Z)

plt.show()
```

1) On a directement $A_1 = X - \frac{1}{2}$ puis, par intégration et annulation de l'intégrale, $A_2 = \frac{X^2}{2} - \frac{X}{2} + \frac{1}{12}$. On trouve aussi $A_3 = \frac{X^3}{6} - \frac{X^2}{4} + \frac{X}{12}$.

2) Voici un programme qui renvoie A_n en tant qu'objet Python de type Polynomial.

```
import numpy.polynomial as pol
def A(n):
    U=pol.Polynomial([1])
     if n==0:
         return(U)
     else
    B=A(n-1).integ()
    C=B.integ(1,0)
    D=B-C(1)
     return(D)
```

3) On compare $A_n(0)$ et $A_n(1)$ pour les 9 premières valeurs via le programme suivant :

```
for n in range (1,10) :
  print(n, A(n)(0), A(n)(1))
```

Le résultat obtenu suggère la conjecture suivante :

Conjecture 1. On a $A_n(0) = A_n(1)$ pour tout $n \geqslant 2$.

On peut également composer les polynômes (le code de programmation de composition des polynômes est intuitif) :

```
for n in range (1,10) :
```

```
S=pol.Polynomial([1,-1])
print(n)
print(A(n))
print(A(n)(S))
```

Au signe près, on trouve les mêmes valeurs numériques (non facilement identifiables) pour les coefficients de $A_n(1-X)$ et A_n. La conjecture qui se profile est :

Conjecture 2. On a $A_n(1-X)=(-1)^n A_n$ pour tout $n \in \mathbf{N}$.

4) Passons à la représentation graphique demandée.

Conjecture 3. On a

$$\forall x \in \mathbf{R}\backslash\{0\}, \quad \frac{x}{e^x-1}=\sum_{k=0}^{+\infty} A_k(0) x^k$$

5) Preuve de la conjecture 1 . Soit $n \in \mathbf{N}$ tel que $n \geqslant 2$. On a tout simplement :

$$A_n(1)-A_n(0)=\int_0^1 A_n^{\prime}(t) \mathrm{d} t=\int_0^1 A_{n-1}(t) \mathrm{d} t=0$$

Preuve de la conjecture 2 par récurrence. La formule est triviale pour $n=0$. On suppose que l'on a $A_n(1-X)=(-1)^n A_n$ et l'on souhaite montrer la formule $A_{n+1}(1-X)=(-1)^{n+1} A_{n+1}$. Or le polynôme différence $A_{n+1}(1-X)-(-1)^{n+1} A_{n+1}$ est clairement d'intégrale nulle de 0 à 1 (quitte à faire un changement de variable linéaire immédiat sur le terme en $1-X$) et son polynôme dérivé est

$$-A_{n+1}^{\prime}(1-X)-(-1)^{n+1} A_{n+1}^{\prime}=-A_n(1-X)-(-1)^{n+1} A_n=0$$

Ainsi, $A_{n+1}(1-X)-(-1)^{n+1} A_{n+1}$ est forcément le polynôme nul.

Preuve de la conjecture 3. ll y a deux difficultés dans cette conjecture. D'abord, il faut justifier que la série du second membre converge et préciser pour quelles valeurs de x. En l'occurrence, on a peut-être été trop généreux en s'autorisant à choisir x dans $\mathbf{R}$ (privé de $\{0\}$). Il serait déjà satisfaisant d'avoir une information pour x voisin de l'origine.

Justifions que la suite $(A_k(0))$ est bornée. Cela prouvera, d'après la théorie des séries entières, que la série $\sum A_k(0) x^k$ converge pour tout $x \in]-1,1[$ (le rayon de la série entière est au moins égal à 1). D'après la définition des polynômes A_k, l'inégalité $\sup_{-1 \leqslant x \leqslant 1}|A_k(x)| \leqslant 1$ s'obtient immédiatement par récurrence grâce au lemme suivant :

Lemme 1. Soit $f \in \mathscr{C}^1([0,1], R)$ vérifiant $\int_0^1 f(x) \mathrm{d} x=0$ et $\|f^{\prime}\|_{\infty} \leqslant 1$. Alors $\|f\|_{\infty} \leqslant 1$.

Preuve. Pour tout $x \in[-1,1]$, on écrit

$$f(x)=(f(x)-f(1/2))+f(1/2)$$

L'inégalité des accroissements finis donne

$$|f(x)-f(1/2)| \leqslant|x-1/2| \times\|f^{\prime}\|_{\infty} \leqslant \frac{1}{2}$$

En intégrant () sur $[0,1]$ et en exploitant (), on obtient

$$|f(1/2)|=\left|\int_0^1 f(x)-f(1/2) \mathrm{d} x\right| \leqslant \int_0^1|f(x)-f(1/2)| \mathrm{d} x \leqslant \frac{1}{2}$$

Grâce à () et à ( ) , on conclut que $|f(x)| \leqslant 1$. $\square$.

Ensuite, il faut trouver un moyen d'identifier les coefficients de deux côtés ! Il est plus simple de tenter de démontrer la formule suivante :

$$\forall x \in]-1,1[, \quad x = (e^x - 1)\sum_{k=0}^{+\infty} A_k(0)x^k$$

Remarque. L'étude de l'intervalle maximal sur lequel la série est convergente (c'est-à-dire le calcul exact du rayon de convergence) dépasse largement le cadre du programme des classes préparatoires (il faudrait invoquer un cours d'analyse complexe et l'appliquer à la fonction holomorphe $z \mapsto \frac{z}{e^z-1}$ sur le disque complexe ouvert de centre 0 et de rayon 2π). D'ailleurs, si on reprend les représentations graphiques précédentes sur un intervalle strictement plus grand que $[-2\pi, 2\pi]$, on constate effectivement une explosion à l'extérieur de cet intervalle :

Revenons à la preuve. Par convergence absolue des séries envisagées, le membre droit se reformule via un produit de Cauchy :

$$\forall x \in]-1,1[, \left(\sum_{k=1}^{+\infty} \frac{x^k}{k!}\right) \times \left(\sum_{k=0}^{+\infty} A_k(0)x^k\right) = \sum_{n=1}^{+\infty}\left(\sum_{k=1}^{n} \frac{A_{n-k}(0)}{k!}\right)x^n.$$

Le coefficient d'indice $n=1$ vaut $A_0(0)=1$ (d'après la question a)). La formule () sera donc conséquence des identités suivantes :

$$\forall n \geqslant 2, \quad \sum_{k=1}^{n} \frac{A_{n-k}(0)}{k!} = 0$$

Cette formule est a priori non évidente mais devient très abordable si l'on invoque le lemme suivant :

Lemme 2. Pour tout polynôme $P \in \mathbb{R}_n[X]$ on a la formule $P(1) = \displaystyle\sum_{k=0}^{n} \frac{P^{(k)}(0)}{k!}$.

Preuve. Il suffit d'appliquer la formule de Taylor de 0 à 1 .

On applique le lemme précédent au polynôme A_n (qui est de degré n et vérifie $A_n^{(k)} = A_{n-k}$ par récurrence immédiate) :

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

La conjecture 1 (prouvée ci-dessus) affirme que $A_n(1) = A_n(0)$.

Planche 4 :

1)
```
import numpy.linalg as alg

def M(n)  :
A = n p.zeros((n, n))
for j in range(n)  :
    for i in range(n)  :
        if i ! = j  :
            A[i, j] = j + 1
return A
```

2)
```
for n in range(2, 11)  :
A = M(n)
print('n = ', n)
print(A)
print(alg.eigvals(A))
```

Conjecture : A possède n valeurs propres distinctes, une strictement positive et les autres strictement négatives.

3) On prouve plus précisément que $\lambda \in \mathrm{Sp}\,(A) \iff \sum_{k=1}^{n} \frac{k}{\lambda+k} = 1$.
Un réel λ est valeur propre de A si et seulement si le système linéaire $(\mathscr{S}_\lambda)$: $AX = \lambda X$ possède une solution non nulle. Or

$$(\mathscr{S}_\lambda) \iff \forall k \in \{1, \cdots, n\}, \quad \sum_{i=1, i\neq k}^{n} ix_i = \lambda x_k \iff \forall k \in \{1, \cdots, n\}, \quad \sum_{i=1}^{n} ix_i = (\lambda + k)x_k.$$

Montrons tout d'abord que si $\lambda = -j$ pour un certain j de $\{1, \ldots, n\}$ alors λ n'est pas valeur propre. En effet si X est solution du système $(\mathscr{S}_\lambda)$, alors l'équation (L_j) donne $\sum_{i=1}^{n} ix_i = 0$ et, pour $k \neq j$, l'équation (L_k) donne ensuite $0 = (-j + k)x_k$, soit $x_k = 0$, pour tout $k \neq j$ mais $\sum_{i=1}^{n} ix_i = 0$ donne aussi $x_j = 0$ puis $X = 0$.

Reprenons alors l'étude de $(\mathscr{S}_\lambda)$ pour un réel $\lambda \notin \{-1, -2, \ldots, -n\}$.

— Supposons que λ est valeur propre et que X est une solution non nulle de $(\mathscr{S}_\lambda)$. Posons $s = \sum_{i=1}^{n} ix_i$. Il vient
$$\forall k \in \{1, \ldots, n\}, \quad x_k = \frac{s}{\lambda + k},$$
donc $s \neq 0$ (sinon $X = 0$) puis $s = \sum_{k=1}^{n} kx_k = \sum_{k=1}^{n} \frac{ks}{\lambda+k}$ et en simplifiant par s, on obtient $1 = \sum_{k=1}^{n} \frac{k}{\lambda+k}$.

— Réciproquement soit λ un réel tel que $\sum_{k=1}^{n} \frac{k}{\lambda+k} = 1$. Le vecteur $X = (\frac{1}{\lambda+1}, \frac{2}{\lambda+2}, \ldots, \frac{n}{\lambda+n})^\top$ vérifie
$$\sum_{i=1}^{n} ix_i = \sum_{i=1}^{n} \frac{i}{\lambda + i} = 1 = (\lambda + k)x_k$$
pour tout $k \in \{1, \ldots, n\}$, ce qui équivaut au système $(\mathscr{S}_\lambda)$ et donc $AX = \lambda X$. Comme X est non nul, un tel λ est valeur propre de A.

4) La fonction $f\colon x \mapsto \sum_{k=1}^{n} \frac{k}{x+k} - 1$ est définie sur $D = \mathbb{R} \setminus \{-n, -n+1, \ldots, -2, -1\}$ et pour $k \in \{2, \ldots, n\}$ f décroît strictement sur l'intervalle $]-k, -k+1[$ de $+\infty$ à $-\infty$. Par continuité, elle s'y annule une fois et une seule en λ_k. De même sur $]-1, +\infty[$ f décroît strictement de $+\infty$ à -1. On obtient ainsi n valeurs propres distinctes pour A_n, notées $\lambda_1, \ldots, \lambda_n$, vérifiant :

$$-n < \lambda_n < -n + 1 < \lambda_{n-1} < -n + 2 < \cdots < -3 < \lambda_3 < -2 < \lambda_2 < -1 < \lambda_1.$$

La matrice A_n est donc diagonalisable.

Remarque. — Comme $f(0) > 0$ on a en fait $\lambda_1 > 0$. Il semble aussi que λ_1 devienne de plus en plus grand avec n. Confirmons-le : on a $f(n) = \sum_{k=1}^{n} \frac{k}{n+k} - 1 \geqslant \frac{1}{2n}(\sum_{k=1}^{n} k) - 1 = \frac{n+1}{4} - 1 \geqslant 0$ si $n \geqslant 3$. On en déduit que $\lambda_1 \geqslant n$.

Planche 5 :

1)
```
def Q1(n) :
    A = np.zeros((n+1,n+1))
    for j in range(1,n+1) :
        A[j-1,j]=j # Attention, on commence à zéro !!
    for j in range(n) :
        A[j+1,j]=n-j # Attention, on commence à zéro !!
    return A
```

2)
```
def Q2(n) :
    A=Q1(n)
    return alg.eigvals(A)
```
Après quelques tests pour différentes valeurs de n, on conjecture que les valeurs propres sont $-n, -n+2, -n+4, \dots, n-2, n$.

3) On le voit sur la base canonique, et $Q = 1 - X^2$.

4) Il faut résoudre $P = P' + nXP$, donc $P(1-nX) = (1-X^2)P'$.
En raisonnant sur le terme de plus haut degré : si P vecteur propre, $\deg P = n$.

5) La matrice A_{n+1} est-elle diagonalisable ?

Planche 6 :

```
import matplotlib.pyplot as plt
import numpy as np
from math import log

# Q1 :

def P(n,x) :
    puiss = 1
    somme = 1
    for k in range(2*n) :
        puiss *= x
        somme += puiss
    return somme

def Q1() :
    plt.clf()
    N=100
    X=[-2+4*k/N for k in range(N+1)]
    for n in range(1,10) :
        Y = [P(n,x) for x in X]
        plt.plot(X,Y,label="n="+str(n))
    plt.legend()
    plt.axis([-2, 2, 0, 5])
    plt.savefig("Q1.png")

def Q1bis(a,b) :
    plt.clf()
    N=100
    X=[a+(b-a)*k/N for k in range(N+1)]
    for n in range(1,10) :
        Y = [P(n,x) for x in X]
        plt.plot(X,Y,label="n="+str(n))
    plt.legend()
    plt.savefig("Q1bis.png")

# prendre a = -0.9 et b= -0.35 pour bien voir les minima. Un seul par fonction
# sur [-2,2].
```

```
# Q2  : si x  != 1, P_n(x)=(1-x**(2n+1))/(1-x), donc
# P_n'(x)=(2nx**(2n+1)-(2n+1)x**(2n)+1)/(1-x)**2.


# Q3  : u_n'(x) = 2n(2n+1)x**(2n-1) * (x-1) et u_n(0)=1, u_n(1)=0
# donc u_n est négative donc P_n décroissante sur ]-inf,a_n] et
# positive donc P_n croissante sur [a_n,+inf[.
# pour un certain a_n < 0.

# Q4  : u_n(-1)=-4n <0 donc a_n\in]-1,0[. On fait une dichotomie sur cet
# intervalle.

def u(n,x)  :
    puiss = 1
    y = x**2
    for i in range(n)  :
        puiss *= y
    return 2*n*x*puiss-(2*n+1)*puiss+1

def dicho (n)  :
    a = -1
    b = 0
    e = 10**(-4)
    while abs(b-a)>e  :
        m = (a+b)/2
        if u(n,m) > 0  : b = m
        else  : a = m
    return (a+b)/2

# Q5  :

def Q5()  :
    X = [n for n in range(1,501)]
    plt.clf()
    Y = [dicho(n) for n in X]
    plt.plot(X,Y)
    plt.savefig("Q5.png")

# On conjecture que cette suite tend vers -1.

# Q6  :

# équivalent simple  : ln(n).
# on écrit u_n(a_n)=0, on passe au log, on a 2n ln|a_n| = -ln(2n+1-2na_n)
# donc ln|a_n| equivalent à -ln(n) / 2n, qui tend donc vers 0, donc |a_n| -> 1
# or a_n<0 donc a_n -> -1.


# Q7  :

# -h_n ~ ln|a_n| ~ -(ln n)/2n donc h_n ~ (ln n)/(2n).
```

```
# Q8  :

def w(n)  :
    return dicho(n)+1-log(n)/(2*n)-log(2)/n

def Q8()  :
    X = [n for n in range(1,501)]
    plt.clf()
    Y = []
    somme = 0
    for i in range(1,501)  :
        somme += w(i)
        Y.append(somme)
    plt.plot(X,Y)
    plt.savefig("Q8.png")
    return Y

# Ca semble converger.

# Q9  : on reprend ln|a_n| = -ln(2n+1-2na_n), on développe, on trouve
# h_n=(ln n)/2n + (ln 2)/n + o(h_n**2) or h_n**2 ~ (ln n)**2 / 4n**2, dont la
# série conv (série de Bertrand classique).
```

Planche 7 :

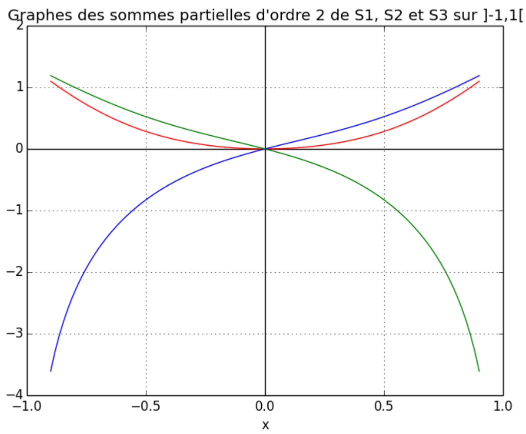
1) Si $|x| < 1$ est fixé, les trois termes généraux des séries étudiées sont équivalents à des termes généraux de séries géométriques de raison x ou x^2, donc les trois séries convergent absolument, donc convergent. Si $|x| \geqslant 1$, la première série est grossièrement divergente, et les deux autres sont soit grossièrement divergentes, soit ne sont même pas définies. En conclusion, les domaines de définition de S_1, S_2, S_3 sont les mêmes et valent $]-1,1[$.

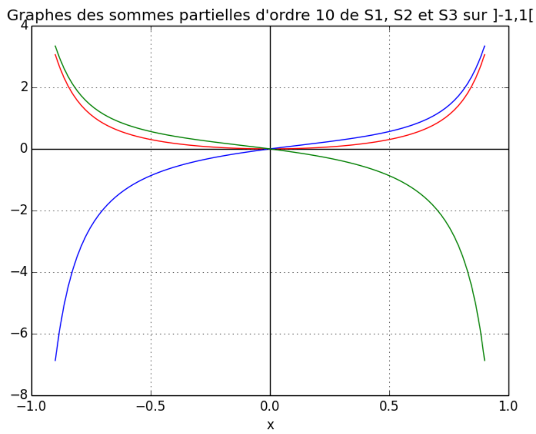
2) Voici les fonctions utilisées pour cette question et la suivante.

```
def S1(n, x) :
    resultat = 0
    for k in range(1, n + 1) :
        resultat = resultat +log(1 + x**(2*k))
    return(resultat)

def S2(n, x) :
    resultat = 0
    for k in range(1, n + 1) :
        resultat = resultat +log(1 + x**(2*k - 1))
    return(resultat)

def S3(n, x) :
    resultat = 0
    for k in range(1, n + 1) :
        resultat = resultat +log(1 - x**(2*k + 1))
    return(resultat)
```

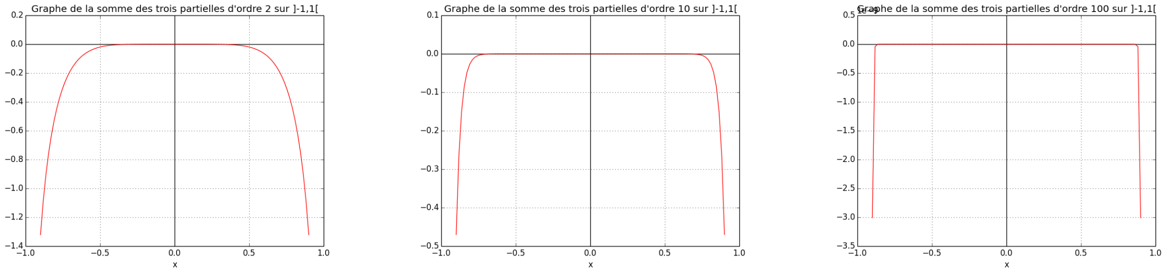
```
def S(n, x) :
    return(S1(n, x)+S2(n, x)+S3(n, x))

def trace_sommes_partielles(n) :
    x = linspace(-0.9, 0.9, 100)
    y1 = S1(n, x)
    y2 = S2(n, x)
    y3 = S3(n, x)
    grid()
    title("Graphes des sommes partielles d'ordre "+ str(n)+" de S1, S2 et S3 sur ]-1, 1[")
    xlabel("x")
    axhline(color = "black")
    axvline(color = "black")
    plot(x, y1, color = "red")
    plot(x, y2, color = "blue")
    plot(x, y3, color = "green")
    show()

def trace_somme_des_trois_sommes_partielles(n) :
    x = linspace(-0.9, 0.9, 100)
    y = S(n, x)
    grid()
    title("Graphe de la somme des trois partielles d'ordre "+ str(n)+" sur ]-1, 1[")
    xlabel("x")
    axhline(color = "black")
    axvline(color = "black")
    plot(x, y, color = "red")
    show()
```

Voici les graphes des sommes partielles d'ordre 2 et 10, tracés en fait sur l'intervalle $[-\frac{9}{10}, \frac{9}{10}]$:





3) Et voici les graphes de la somme des trois sommes partielles d'ordre 2, 10 et 100 :



On conjecture que $S_1+S_2+S_3$ est identiquement nulle sur $]-1,1[$.

4) On pose $u_n\colon x\in]-1,1[\mapsto\ln(1+x^{2n})$, $v_n\colon x\in]-1,1[\mapsto\ln(1+x^{2n-1})$, $w_n\colon x\in]-1,1[\mapsto\ln(1-x^{2n-1})$. Ces fonctions sont de classe $\mathscr{C}^1$ avec

$$\forall x\in]-1,1[,\quad u_n'(x)=\frac{2nx^{2n-1}}{1+x^{2n}},\quad v_n'(x)=\frac{(2n-1)x^{2n-2}}{1+x^{2n-1}},\quad w_n'(x)=-\frac{(2n-1)x^{2n-2}}{1-x^{2n-1}}.$$

Soit $a\in[0,1[$. La norme uniforme commune aux fonctions $x\mapsto\frac{1}{1+x}$ et $x\mapsto\frac{1}{1-x}$ sur $[-a,a]$ est $K:=\frac{1}{1-a}$. Comme $x\in[-a,a]$ implique $\{x^{2n},x^{2n-1},-x^{2n-1}\}\subset[-a,a]$, on en déduit que

$$\|u_n'\|_\infty^{[-a,a]}\leqslant 2na^{2n-1},\quad \|v_n'\|_\infty^{[-a,a]}\leqslant(2n-1)Ka^{2n-2},\quad \|w_n'\|_\infty^{[-a,a]}\leqslant(2n-1)Ka^{2n-2}.$$

Les théorèmes de croissance comparées montrent alors que les trois séries dérivées $\sum u_n'$, $\sum v_n'$ et $\sum w_n'$ convergent normalement sur $[-a,a]$. Comme la convergence simple des trois séries elles-mêmes sur $]-1,1[$ a été prouvée à la question précédente, on déduit du théorème de dérivation que S_1, S_2 et S_3 sont de classe $\mathscr{C}^1$ sur $]-1,1[$, et que

$$\forall x\in]-1,1[,\quad S_1'(x)=\sum_{n=1}^{+\infty}\frac{2nx^{2n-1}}{1+x^{2n}},\quad S_2'(x)=\sum_{n=1}^{+\infty}\frac{(2n-1)x^{2n-2}}{1+x^{2n-1}},\quad S_3'(x)=-\sum_{n=1}^{+\infty}\frac{(2n-1)x^{2n-2}}{1-x^{2n-1}}.$$

5) On pose $S=S_1+S_2+S_3$. Soit $x\in]-1,1[$. Dans la somme $S(x)$, on regroupe les termes provenant de $S_2(x)$ et $S_3(x)$, en profitant de ce que $(1-x^{2n-1})(1-x^{2n+1})=1-x^{4n-2}$:

$$S(x)=\sum_{n=1}^{+\infty}\ln(1+x^{2n})+\sum_{n=1}^{+\infty}\ln(1-x^{4n-2}).$$

On sépare ensuite, dans la première somme, les termes d'indice pair de ceux d'indice impair : $\sum_{n=1}^{+\infty}\ln(1+x^{2n})=\sum_{n=1}^{+\infty}\ln(1+x^{4n})+\sum_{n=1}^{+\infty}\ln(1+x^{4n-2})$. Cette séparation est légitime car les deux séries qu'on vient d'écrire convergent, et elle permet le regroupement des termes $\ln(1+x^{4n-2})$ et $\ln(1-x^{4n-2})$:

$$S(x)=\sum_{n=1}^{+\infty}\ln(1+x^{4n})+\sum_{n=1}^{+\infty}\ln(1-x^{8n-4}).$$

On a alors l'idée de démontrer par récurrence sur $p\in\mathbb{N}^*$ que

$$\forall x\in]-1,1[,\quad S(x)=\underbrace{\sum_{n=1}^{+\infty}\ln\left(1+x^{2^pn}\right)}_{u_p(x)}+\underbrace{\sum_{n=1}^{+\infty}\ln\left(1-x^{2^p(2n-1)}\right)}_{v_p(x)}.\qquad(\mathscr{H}_p)$$

Les propriétés $(\mathscr{H}_1)$ et $(\mathscr{H}_2)$ ont été établies plus haut. Si $(\mathscr{H}_p)$ est vraie, alors, en séparant les termes pour n pair de ceux pour n impair dans la première somme, puis en regroupant ces derniers avec ceux de la seconde somme, on obtient :

$$\begin{aligned}\forall x\in]-1,1[,\quad S(x)&=\sum_{n=1}^{+\infty}\ln\left(1+x^{2^{p+1}n}\right)+\sum_{n=1}^{+\infty}\ln\left(1+x^{2^p(2n-1)}\right)+\sum_{n=1}^{+\infty}\ln\left(1-x^{2^p(2n-1)}\right)\\&=\sum_{n=1}^{+\infty}\ln\left(1+x^{2^{p+1}n}\right)+\sum_{n=1}^{+\infty}\ln\left(1-x^{2^{p+1}(2n-1)}\right)=u_{p+1}(x)+v_{p+1}(x).\end{aligned}$$

C'est bien la propriété $(\mathscr{H}_{p+1})$. Il convient de noter que le membre de gauche, $S(x)$, est indépendant de p. On va alors montrer que $S(x) = 0$ en fixant $x \in]-1, 1[$, et en faisant varier p.
Comme l'exposant $2^p n$ est pair, $x^{2^p n}$ est positif, donc on peut lui appliquer l'encadrement $\forall y \in \mathbb{R}_+,\ 0 \leqslant \ln(1+y) \leqslant y$. On obtient alors

$$0 \leqslant \sum_{n=1}^{+\infty} \ln\left(1 + x^{2^p n}\right) \leqslant \sum_{n=1}^{+\infty} x^{2^p n} = \frac{x^{2^p}}{1 - x^{2^p}}.$$

Comme le majorant tend vers zéro quand p tend vers $+\infty$, on a déjà prouvé que la suite $(u_p(x))_{p\in\mathbb{N}^*}$ converge vers zéro. On ne dispose pas d'inégalités aussi efficaces pour établir la convergence de la suite $(v_p(x))_{p\in\mathbb{N}^*}$, mais on va se contenter d'appliquer l'équivalent $\ln(1+y) \sim y$ en zéro. On en déduit l'existence de $\varepsilon_0 > 0$ tel que $\forall y \in [-\varepsilon_0, \varepsilon_0],\ |\ln(1+y)| \leqslant \frac{3}{2}|y|$. Comme on dispose de la majoration *uniforme en n* suivante

$$\forall n \in \mathbb{N}^*, \quad \left|-x^{2^p(2n-1)}\right| \leqslant x^{2^p},$$

et comme x^{2^p} tend vers zéro quand p tend vers $+\infty$, il existe $p_0 \in \mathbb{N}^*$ tel que $\forall n \in \mathbb{N}^*,\ |\ln(1 - x^{2^p(2n-1)})| \leqslant \frac{3}{2} x^{2^p(2n-1)}$. Par suite,

$$\forall p \geqslant p_0, \quad |v_p(x)| \leqslant \frac{3}{2} \sum_{n=1}^{+\infty} x^{2^p(2n-1)} = \frac{3}{2} \frac{x^{2^p}}{1 - x^{2^{p+1}}}.$$

Cela prouve que la suite $(v_p(x))_{p\in\mathbb{N}^*}$ converge vers zéro et, finalement, que $S(x) = 0$, pour tout $x \in]-1, 1[$. La conjecture est prouvée. On remarque pour terminer que, comme l'indiquent les graphes de la question **3)**, la convergence de $(S_n(x))_{n\in\mathbb{N}^*}$ vers la fonction nulle ne semble pas uniforme sur $]-1, 1[$.

Planche 8 :

```
import matplotlib.pyplot as plt
import numpy as np
import random as rd
import numpy.linalg as lin



# Q1  ;

def position(n)  :
    pos=[1]
    for i in range(n-1)  :
        if pos[-1] == 1 : pos.append(rd.randint(1,4))
        else  : pos.append(pos[-1]-1)
    return pos

def trace_pos()  :
    plt.clf()
    N=[10,50,100]
    for n in N  :
        X=[k for k in range(n)]
        Y=position(n)
        plt.plot(X,Y,label="n="+str(n))
    plt.legend()
```

```
    plt.savefig("position.png")


# Conjecture  : plus la valeur est basse, plus souvent elle est atteinte

# Q3.a  : A =   [[1/4,  1,  0,  0],
#               [1/4,  0,  1,  0],
#               [1/4,  0,  0,  1],
#               [1/4,  0,  0,  0]]
A=np.array([[1/4,1,0,0],[1/4,0,1,0],[1/4,0,0,1],[1/4,0,0,0]])

# Q3.b  : c'est un matrice compagnon de polynôme caractéristique
# X**4-1/4(X**3+X**2+X+1)

poca = np.poly(A)
lin.eigvals(A) # on trouve 4 racines distinctes, dont deux ne sont pas réelles
# donc diagonalisable dans C mais pas dans R.


# Q3.c  : ces racines sont  : 1, et les autres de module <1
# donc A**n converge vers P((1,0,0,0),(0,0,0,0),(0,0,0,0),(0,0,0,0))P-1.
# Donc U_n converge vers cte * le vecteur propre associé à la vp 1, et la somme
# des coordonnées vaut 1. On résout. On trouve 1/10(4,3,2,1).

# Q4.a  :

def compte(n)  :
    a = position(n+1)
    Y = [0, 0, 0, 0]
    for p in a  :
        Y[p-1] += 1
    return Y

# Q5.b  :

def test()  :
    C = []
    for i in range(100)  :
        C.append(compte(100))
    return C

# Q5.c  :

def esperance()  :
    C=test()
    return [(1/101)*sum([C[j][i] for j in range(len(C))]) for i in range(4)]

# ça colle.
```

Planche 9 :

1)
```
>>> def Suite(n, x) :
        y = x
        for i in range(1, n + 1) :
            y = (y**2)/i
        return y
>>> import matplotlib.pyplot as plt
>>> plt.plot([Suite(n, 1.6616) for n in range(32)])
>>> plt.show()
```

2) Python renvoie 0.0 dans le premier cas, 49.02934290985918 dans le second, ce qui montre la sensibilité à de petites perturbations…

3) Supposons (i) , soit donc n tel que $u_{n+1} \leqslant u_n$. Nécessairement $u_{n+1} \geqslant 0$ donc $u_{n+1}^2 \leqslant u_n^2$, donc $u_{n+2} = \frac{u_{n+1}^2}{n+2} \leqslant \frac{u_n^2}{n+2} \leqslant \frac{u_n^2}{n+1} = u_{n+1}$. Par récurrence, on montre que la suite $(u_k)_{k\geqslant n}$ est alors décroissante.

En particulier, elle est majorée par son premier terme u_n, et en considérant un entier $k \geqslant n$ tel que $\frac{u_n^2}{k+1} < 1$, qui existe, on a alors $u_{k+1} = \frac{u_k^2}{k+1} \leqslant \frac{u_n^2}{k+1} < 1$, d'où (ii).

Supposons (ii) et soit $n \in \mathbb{N}^*$ tel que $u_n < 1$. Alors une récurrence élémentaire montre que $\forall k \geqslant n$, $0 \leqslant u_k < 1$ donc $0 \leqslant u_{k+1} < \frac{1}{k+1}$. Par le théorème d'encadrement, on a donc (iii).

Supposons enfin (iii). La suite est à valeurs positives (à partir du rang 1) et de limite nulle, elle ne peut donc être strictement croissante, d'où (i).

4) Supposoons trouvé $N \in \mathbb{N}$ tel que $u_N \geqslant N + 2$. Soit, pour $n \geqslant N$, $\mathscr{P}(n)$ la propriété $u_n \geqslant n + 2$.

- $\mathscr{P}(N)$ est l'hypothèse faite ici.
- Soit $n \geqslant N$, supposons $\mathscr{P}(n)$. Alors $u_n^2 \geqslant (n + 2)^2$ donc $u_{n+1} \geqslant \frac{(n+2)^2}{n+1} \geqslant n + 3$, en effet $(n + 2)^2 = n^2 + 4n + 4 \geqslant n^2 + 4n + 3 = (n + 1)(n + 3)$, soit $\mathscr{P}(n + 1)$.

On conclut par récurrence que $\forall n \geqslant N$, $u_n \geqslant n + 2$.

5) Pour $x = 1$, on a $u_0 = 1 = u_1$ et $u_2 = \frac{1}{2} \leqslant u_1$ donc u est de limite nulle, soit $1 \in E_0$.
Pour $x = 2$, on a $u_0 = 2, u_1 = 4 \geqslant 3$ donc $\forall n \geqslant 1, u_n \geqslant n + 2$ donc u est de limite $+\infty$, soit $2 \in E_\infty$.
La question c montre que pour tout x, on a soit $(u_n(x))$ strictement croissante (et positive), soit il existe $n \in \mathbb{N}$ tel que $u_{n+1}(x) \leqslant u_n(x)$ et alors $(u_n(x))$ est de limite nulle. Dans tous les cas, $(u_n(x))$ admet une limite dans $\mathbb{R}_+ \cup \{+\infty\}$.
Or si $u_n(x) \xrightarrow[n\to+\infty]{} \ell \in \mathbb{R}_+$, alors $\frac{u_n(x)^2}{n+1} \xrightarrow[n\to+\infty]{} 0$ soit $u_{n+1}(x) \xrightarrow[n\to+\infty]{} 0$ donc $\ell = 0$. Donc pour tout $x > 0$, $(u_n(x))$ admet une limite égale à 0 ou à $+\infty$, c'est-à-dire que $\mathbb{R}_+^* = E_0 \cup E_\infty$.
Enfin pour $0 < x \leqslant y$, on a par une récurrence immédiate $\forall n \in \mathbb{N}, u_n(x) \leqslant u_n(y)$ donc $\lim\limits_{n\to+\infty} u_n(x) \leqslant \lim\limits_{n\to+\infty} u_n(y)$. Donc si $y \in E_0$ alors $x \in E_0$. Finalement $\forall y \in E_0$, $]\,0\,;y\,] \subset E_0$. E_0 est donc convexe donc c'est un intervalle.
De même si $x \in E_\infty$ alors $y \in E_\infty$. Finalement $\forall x \in E_\infty$, $[\,x\,;+\infty\,[\subset E_\infty$. E_∞ est donc convexe donc c'est un intervalle.

Planche 10 :

Planche 11 :

1) On donne une première fonction de test, qui utilise la question **2)**, et une seconde pour générer une des 2^{n^2} matrices à tester à partir de l'entier correspondant (ordre lexicographique inverse), et la fonction principale…

```
import numpy as n p

def test(M, n) :
M = n p.transpose(M).dot(M) - n*n p.eye(n)
for i in range(n) :
    for j in range(n) :
        if M[i, j] ! = 0 :
            return False
return True

def matrice(k, n) :
l = n p.array([1 for i in range(n**2)])
for i in range(n**2) :
    if k%2 = = 1 :
        l[i] = -1
    k = k//2
return l.reshape(n, n)

def denombre(n) :
compteur = 0
for k in range(2**(n**2)) :
    M = matrice(k, n)
    if test(M, n) :
        compteur + = 1
return compteur
```

2) Soit $A \in \mathscr{M}_n(\mathbb{R})$ dont les coefficients appartiennent tous à $\{-1, 1\}$. Si A vérifie $\mathscr{H}_n$, ses colonnes $C_1, \dots, C_n$ sont de norme $\sqrt{n}$, orthogonales deux à deux, donc la famille $(\frac{1}{\sqrt{n}}C_1, \dots, \frac{1}{\sqrt{n}}C_n)$ est orthonormée, si bien que $\frac{1}{\sqrt{n}}A$ est orthogonale.

Réciproquement si $\frac{1}{\sqrt{n}}A$ est orthogonale, alors la famille $(\frac{1}{\sqrt{n}}C_1, \dots, \frac{1}{\sqrt{n}}C_n)$ est orthonormée donc $(C_1, \dots, C_n)$ est orthogonale, c'est-à-dire que A vérifie $\mathscr{H}_n$.

3) Les 8 matrices vérifiant $\mathscr{H}_2$ sont exactement les 4 matrices dont trois coefficients valent 1 et le dernier -1, et les 4 matrices dont trois coefficients valent -1 et le dernier 1. En divisant par $\sqrt{2}$, on obtient 8 matrices orthogonales dont 4 sont symétriques et correspondent à des matrices de symétries orthogonales, et les 4 autre correspondent à des matrices de rotation d'angles respectifs $\pm\frac{\pi}{4}$ et $\pm\frac{3\pi}{4}$. Les endomorphismes associés à nos 8 matrices sont donc les composées commutatives de ces symétries ou rotations avec l'homothétie de rapport $\sqrt{2}$, soit des similitudes (directes ou indirectes) du plan.

4) Il suffit de noter que les coefficients de $A^\top$ restent dans $\{-1, 1\}$ et que $\frac{1}{\sqrt{n}}A$ est orthogonale si et seulement si $\frac{1}{\sqrt{n}}A^\top$ l'est.

5) Changer les signes sur la colonne j revient à changer C_j en $-C_j$, et préserve bien sûr le caractère orthogonal de la famille des colonnes.

Et changer les signes sur la ligne i correspond à changer les signes sur la colonne i de $A^\top$, ce qui préserve la propriété $\mathscr{H}_n$ pour $A^\top$ donc pour A.

6) Il suffit de tester en plus la première ligne et la première colonne... On obtient 6 matrices, pas si compliquées à trouver à la main...

```
def testbis(M, n) :
N = n p.transpose(M).dot(M) - n*n p.eye(n)
```

```
for k in range(n) :
    if M[k, 0] ! = 1 or M[0, k] ! = 1 :
        return False
for i in range(n) :
    for j in range(n) :
        if N[i, j] ! = 0 :
            return False
return True

def denombrebis() :
n = 4
compteur = 0
for k in range(2**(n**2)) :
    M = matrice(k, n)
    if testbis(M, n) :
        compteur + = 1
return compteur
```