Centrale Maths 2 : Quelques planches corrigées

Exercice 1. *Arithmétique : pas de module particulier à importer.*

1. Écrire une fonction qui renvoie le PGCD de deux entiers relatifs.

2. Soit φ une fonction de $\mathbb{N}$ vers $\mathbb{N}$ telle que $\varphi(0) = 0$, $\varphi(1) = 1$ et

$$\forall\, n \in \mathbb{N} \qquad \varphi(2n) = \varphi(n) \qquad \text{et} \qquad \varphi(2n+1) = \varphi(n+1) + \varphi(n)$$

Écrire une fonction qui renvoie $\varphi(n)$.

3. Que vaut $\text{PGCD}(\varphi(n), \varphi(n+1))$? Conjecture, démonstration.

Corrigé.

1. Par exemple (le PGCD n'est pas défini lorsque les deux entiers sont nuls, on renvoie 0 ici) :

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

2. On écrit une fonction qui renvoie toutes les valeurs de φ entre 0 et $n-1$ directement.

```
def valeurs_phi(n):
    """ retourne toutes les valeurs de phi entre 0 et n-1"""
    L=[0]*n
    L[1]=1
    for i in range(2,n):
        if i%2==0:
            L[i]=L[i//2]
        else:
            L[i]=L[i//2]+L[i//2+1]
    return L
```

Remarque : une fonction récursive qui ferait deux appels récursifs dans le cas n impair serait très inefficace, à cause des appels qui se chevauchent.

3. La fonction suivante calcule les PGCD de deux termes successifs de la suite $(\varphi(i))_{0\leq i\leq n-1}$

```
def test_PGCD(n):
    L=valeurs_phi(n)
    return [PGCD(L[i], L[i+1]) for i in range(n-1)]
```

On vérifie que pour $n = 10^5$ par exemple, le résultat ne contient que des 1, par exemple avec l'instruction suivante :

```
print(False in [x==1 for x in test_PGCD(100000)]) #False, s'affiche !
```

Montrons par récurrence forte la propriété $\mathcal{P}(n)$: $\text{PGCD}(\varphi(n), \varphi(n+1)) = 1$:

- la propriété est vérifiée pour $n = 0$ et $n = 1$ car $\varphi(2) = \varphi(1) = 1$;
- soit $n \geq 2$ un entier tel que $\mathcal{P}$ soit vérifiée pour tous les entiers naturels strictement inférieurs. Alors :
  - si n est pair et s'écrit $2p$, on a

$$\text{PGCD}(\varphi(2p), \varphi(2p+1)) = \text{PGCD}(\varphi(p), \varphi(p+1) + \varphi(p)) = \text{PGCD}(\varphi(p), \varphi(p+1))$$

    Ce dernier PGCD vaut 1 par hypothèse de récurrence.
  - sinon, n s'écrit $2p-1$. Alors

$$\text{PGCD}(\varphi(2p-1), \varphi(2p)) = \text{PGCD}(\varphi(p) + \varphi(p-1), \varphi(p)) = \text{PGCD}(\varphi(p), \varphi(p-1))$$

    Et le résultat suit.

— Par principe de récurrence, la propriété est vérifiée pour tout entier $n \geq 0$.

Exercice 2. *Arithmétique : pas de module particulier à importer.* On écrit $H_n = \sum_{k=1}^{n} \frac{1}{k}$ sous la forme d'une fraction irréductible $\frac{A_n}{B_n}$. On pose aussi $C_n = n!H_n$. On veut montrer la propriété :

$$\mathcal{P} : \quad \forall p \geq 5 \text{ et } p \text{ premier}, \quad p^2 | A_{p-1}$$

1. (a) Coder une fonction qui prend en paramètre n et renvoie C_n.
   (b) Montrer que $\mathcal{Q} \Rightarrow \mathcal{P}$, où $\mathcal{Q} : \forall p \geq 5$ et p premier, $p^2 | C_{p-1}$.
   (c) Montrer que $\mathcal{P}$ est vérifiée pour tout $p \leq 1000$.
2. Soit $p \geq 5$ premier, montrer que $(p-1)! \equiv -1[p]$.
3. On pose $a_k = \frac{(p-1)!}{k(p-k)}$ pour tout $k > 0$. Trouver une relation entre $\sum_{k=1}^{p-1} a_k$ et H_{p-1}.
4. Montrer que $k^2 a_k \equiv 1[p]$, puis que p divise $\sum_{k=1}^{p-1} a_k$. Conclure.

Corrigé.

1. Le code suivant produit C_n :

```
from math import *
def C(n):
    f=factorial(n)
    return sum([f//k for k in range(1,n+1)])
```

Comme p ne divise pas $(p-1)!$, il est clair que si p^2 divise l'entier C_{p-1}, il divise A_{p-1}. Le code suivant vérifie $\mathcal{Q}$ pour les entiers premiers entre 5 et 1000. On écrit au préalable une fonction testant (très basiquement !) si un entier naturel est premier.

```
def est_premier(n):
    for i in range(2,n):
        if n%i==0:
            return False
    return n>1 #sinon n n'est pas premier non plus !

P=[i for i in range(5,1000) if est_premier(i)] #les entiers premiers >= à 5 et <1000
print(False in [C(x-1)%(x**2)==0 for x in P]) #vérification de la conjecture
```

On vérifie que `False` n'est pas dans la liste en question, donc $\mathcal{Q}$ est vérifiée jusqu'à 1000.

2. Soit p premier, au moins égal à 5. $\mathbb{Z}/p\mathbb{Z}$ est un corps, chaque élément non nul y possède un inverse. L'équation $x^2 = 1$ y a au plus deux solutions (comme sur tout corps), or 1 et $-1 = p-1$ sont deux solutions distinctes (car $p \neq 2$), donc ce sont les seules. Il s'ensuit que 1 et $p-1$ sont les seuls entiers modulo p à être leurs propres inverses modulo p, d'où $\prod_{i=1}^{p-1} i \equiv 1 \times (p-1) \equiv -1[p]$.

3. On utilise le fait que $\frac{1}{k(p-k)} = \frac{1/p}{k} + \frac{1/p}{p-k}$. D'où

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

4. D'après la définition de a_k et le résultat de la question 2, on a pour tout $k \in [\![1, p-1]\!]$: $k(p-k)a_k \equiv -1[p]$, d'où $k^2 a_k \equiv 1[p]$. Cette relation nous dit que les $(a_k)_{0 \leq k \leq n-1}$ décrivent les inverses des carrés des éléments non nuls de $\mathbb{Z}/p\mathbb{Z}$, donc ils décrivent précisément les carrés des éléments non nuls de $\mathbb{Z}/p\mathbb{Z}$. Autrement dit :

$$\sum_{k=1}^{p-1} a_k \equiv \sum_{k=1}^{p-1} k^2 \equiv 0[p] \qquad \text{car} \qquad \sum_{k=1}^{p-1} k^2 = \frac{p(p-1)(2p-1)}{6} \text{ et 6 est inversible modulo } p \geq 5.$$

On peut donc conclure : de la question 3 on tire que C_{p-1} est divisible par p, et de la question 4 on tire que $\frac{C_{p-1}}{p}$ l'est aussi, donc on a bien $p^2 | C_{p-1}$. $\mathcal{Q}$ est démontrée !

Exercice 3. *Calcul matriciel : déterminant.*

Soient $(a_n)_{n\geq 1} \in (\mathbb{R}_+)^{\mathbb{N}^*}$ et

$$A_n = \begin{pmatrix} 1 & -1 & 0 & \cdots & 0 \\ a_1 & 1 & -1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \cdots & \ddots & \ddots & \ddots & -1 \\ 0 & \cdots & 0 & a_n & 1 \end{pmatrix} \in \mathcal{M}_{n+1}(\mathbb{R})$$

pour tout $n \in \mathbb{N}^*$.

1. Écrire une fonction Python calculant et affichant le déterminant Δ_n de A_n pour de petites valeurs de n.
2. Exécuter le programme lorsque $a_n = 1$ et lorsque $a_n = 1/n^2$. Que peut-on dire ?
3. Montrer que lorsque $a_n = 1$, la suite $(\Delta_n)_n$ est divergente. Donner un équivalent de Δ_n quand n tend vers l'infini.
4. Montrer que $\Delta_n \leq \prod_{k=1}^{n}(1 + a_k)$
5. Montrer que la suite $(\Delta_n)_n$ converge si et seulement si $\sum a_n$ converge.

Corrigé

1. Il n'est pas précisé ce que la fonction doit prendre en entrée : on suppose que c'est simplement la liste des valeurs (a_i) :

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

def calcul_delta_n(A):
    """ A une liste contenant les valeurs de la suite a_i = A[i-1].
    Renvoie Delta_n """
    n=len(A)
    M=np.eye(n+1)
    for i in range(n):
        M[i,i+1]=-1
        M[i+1,i]=A[i]
    return alg.det(M)
```

2. Pour (a_n) la suite constante égale à 1, on reconnaît les valeurs de la suite de Fibonacci (aux erreurs de calcul avec des flottants près) :

```
>>> [calcul_delta_n([1]*n) for n in range(2,10)]
[2.0, 2.9999999999999996, 4.999999999999998, 7.999999999999998, 13.0, 21.0, 34.00000000000001,
55.000000000000014, 88.99999999999999]
```

Pour $(a_n) = (1/n^2)$, il semble que (Δ_n) possède une limite lorsque n tend vers l'infini :

```
>>> [calcul_delta_n([1/i**2 for i in range(1,n+1)]) for n in range(1,10)]
[2.0, 2.25, 2.4722222222222223, 2.6128472222222223, 2.7117361111111116, 2.7843152006172844,
2.8396567539052664, 2.8831616789149117, 2.9182191697038657]
```

3. Supposons $(a_n) = (1)_n$. Un développement suivant la première colonne de A_n donne rapidement : $\Delta_n = \Delta_{n-1} + \Delta_{n-2}$. On calcule ensuite $\Delta_1 = 2$ et $\Delta_2 = 3$ (on pose $\Delta_0 = 1$, ce qui est cohérent). On a donc

$$\Delta_n = \alpha\left(\frac{1+\sqrt{5}}{2}\right)^n + \beta\left(\frac{1-\sqrt{5}}{2}\right)^n$$

avec α et β vérifiant : $\alpha + \beta = 1$, $\frac{\alpha+\beta}{2} + \sqrt{5}\frac{\alpha-\beta}{2} = 2$, on trouve facilement $\alpha = \frac{3+2\sqrt{5}}{2\sqrt{5}}$ et $\beta = \frac{-3+2\sqrt{5}}{2\sqrt{5}}$.

4. On note $\Delta_n(a_1, \ldots, a_n)$ le déterminant de taille $(n+1)$ associé à $(a_i)_{1\leq i\leq n}$. On a $\Delta_1(a_1) = 1+a_1$ (donc l'inégalité demandée est vérifiée pour $n = 1$), et $\Delta_2(a_1, a_2) = 1 + a_1 + a_2 \leq (1 + a_1)(1 + a_2)$ (donc l'inégalité demandée est vérifiée pour $n = 2$). En reprenant le développement inité plus haut, on trouve la relation pour $n \geq 3$:

$$\Delta_n(a_1, \ldots, a_n) = \Delta_{n-1}(a_2, \ldots, a_n) + a_1\Delta_{n-2}(a_3, \ldots, a_n)$$

Et une récurrence immédiate permet d'obtenir l'inégalité voulue.

5. Si $\sum a_n$ converge, alors $a_n \to 0$. En prenant le logarithme, on a donc (les a_i étant positifs) :

$$\ln(\Delta_n) \leq \sum_{k=1}^{n} \ln(1+a_k) \leq \sum_{k=1}^{n} a_k$$

En effectuant un développement sur la dernière ligne plutôt que la première colonne, on obtient la relation similaire :

$$\Delta_n(a_1, \ldots, a_n) = \Delta_{n-1}(a_1, \ldots, a_{n-1}) + a_n \Delta_{n-2}(a_1, \ldots, a_{n-2})$$

qui montre que la suite $(\Delta_n)_n$ est croissante. Comme elle est bornée, elle converge. Réciproquement, cette relation permet de montrer par récurrence assez immédiate que $\Delta_n(a_1, \ldots, a_n) \geq 1 + \sum_{i=1}^{n} a_i$, donc la convergence de (Δ_n) implique que $\sum a_i$ est bornée, et converge car elle est croissante. Ainsi (Δ_n) converge si et seulement si $(\sum a_i)$ converge.

Exercice 4. *Polynômes, calcul matriciel, valeurs propres.*

Soit $E = \mathbb{R}_{n-1}[X]$. On pose pour $P \in E$, $\varphi(P) = (n-1)XP + (1-X^2)P'$.

1. Montrer que φ est un endomorphisme de E.

2. Écrire une fonction Python qui calcule $\varphi(P)$.

3. Calculer la matrice de φ dans la base canonique pour $n = 3$, puis pour n quelconque.

4. Déterminer dans le cas général les éléments propres de φ.

Corrigé.

1. φ est linéaire, il suffit de justifier que l'image de la base canonique est bien dans E. C'est clairement le cas de $\varphi(X^i)$ pour $0 \leq i < n-1$ car $\deg(\varphi(P)) \leq 1 + \deg(P)$, de plus $\varphi(X^{n-1}) = (n-1)X^{n-2} \in E$. φ est bien un endomorphisme de E.

2. On utilisera les modules suivants :

```
from numpy.polynomial import Polynomial
import numpy.linalg as alg
import numpy as np
```

Voici la fonction demandée :

```
def calcule_phi(P,n):
    X=Polynomial([0, 1])
    return (n-1)*P*X+(1-X**2)*P.deriv()
```

3. Une fonction qui calcule la matrice :

```
def calcule_mat(n):
    M=np.zeros((n,n))
    X=Polynomial([0, 1])
    for i in range(n):
        P=X**i
        L=calcule_phi(P,n).coef
        for j in range(min(n,len(L))):
            M[j,i]=L[j]
    return M
```

Remarque : le `min(n,len(L))` est là au cas où le coefficient de X^n dans $\varphi(X^{n-1})$ ne soit pas calculé comme étant zéro exactement, à cause des calculs flottants (ce qui produirait un dépassement d'indice). Voici le résultat pour $n = 3$:

```
>>> calcule_mat(3)
array([[ 0.,  1.,  0.],
       [ 2.,  0.,  2.],
       [ 0.,  1.,  0.]])
```

Et $n = 5$:

```
>>> calcule_mat(5)
array([[ 0.,  1.,  0.,  0.,  0.],
       [ 4.,  0.,  2.,  0.,  0.],
       [ 0.,  3.,  0.,  3.,  0.],
       [ 0.,  0.,  2.,  0.,  4.],
       [ 0.,  0.,  0.,  1.,  0.]])
```

4. Python peut nous permettre d'intuiter les valeurs propres :

```
>>> alg.eigvals(calcule_mat(3))
array([ -2.00000000e+00,  -1.73255507e-16,   2.00000000e+00])
>>> alg.eigvals(calcule_mat(4))
array([-3., -1.,  3.,  1.])
>>> alg.eigvals(calcule_mat(5))
array([  4.00000000e+00,  -4.00000000e+00,  -2.00000000e+00,
         5.37393438e-16,   2.00000000e+00])
```

Il semblerait que les valeurs propres de la matrice de φ dans la base canonique soient $-(n-1), -(n-3), -(n-5), \dots, n-3, n-1$. *Remarque :* la fonction `alg.eig(A)` renvoie un couple (v, P), où v est la liste des valeurs propres de A (donnée sous la forme d'un tableau Numpy de dimension 1) et P est une matrice des vecteurs propres associés, on a donc $A = PDP^{-1}$ avec D la matrice diagonale dont les éléments diagonaux sont ceux de v :

```
>>> alg.eig(calcule_mat(3))
(array([ -2.00000000e+00,  -1.73255507e-16,   2.00000000e+00]),
array([[  4.08248290e-01,   7.07106781e-01,   4.08248290e-01],
       [ -8.16496581e-01,   1.83954421e-17,   8.16496581e-01],
       [  4.08248290e-01,  -7.07106781e-01,   4.08248290e-01]]))
```

Plutôt que de deviner les vecteurs propres, cherchons à résoudre $\varphi(P) = \lambda P$. On a donc une équation différentielle à résoudre, et le théorème de Cauchy-Lipschitz assure que pour tout $\lambda \in \mathbb{R}$ l'espace des solutions sur $]-1, 1[$ est de dimension 1. Explicitons les solutions : elles sont de la forme $f_\lambda : x \mapsto \gamma \exp(A(x))$, où $\gamma \in \mathbb{R}$ et A est une primitive de $x \mapsto -\frac{(n-1)x-\lambda}{1-x^2} = -\frac{(n-1)x}{1-x^2} + \frac{\lambda/2}{1-x} + \frac{\lambda/2}{1+x}$. On peut prendre

$$A(x) = \frac{n-1}{2}\ln(1-x^2) - \frac{\lambda}{2}\ln(1-x) + \frac{\lambda}{2}\ln(1+x)$$

Avec $\gamma = 1$, on obtient la solution :

$$f_\lambda(x) = (1-x^2)^{(n-1)/2}(1-x)^{-\lambda/2}(1+x)^{\lambda/2} = (1-x)^{(n-1-\lambda)/2}(1+x)^{(n-1+\lambda)/2}$$

f_λ est une fonction polynomiale pour tout $\lambda \in [\![-(n-1), n-1]\!]$ de même parité que $n-1$. Le polynôme P_λ associé vérifie donc $\varphi(P_\lambda) = \lambda P_\lambda$ (car $\varphi(P_\lambda) - \lambda P_\lambda$ s'annule sur $]-1, 1[$ et est donc nul). On a donc trouvé n valeurs propres distinctes et n vecteurs propres associés : φ est diagonalisable et ses éléments propres sont complètement déterminés. *Remarque :* c'est cohérent avec ce que renvoie `alg.eig`. Le jour de l'oral, deviner une base de vecteurs propres et vérifier qu'elle convient est tout aussi acceptable !

Exercice 5. *Polynômes, tracé, et résolution d'équations numériques.*

Pour $n \in \mathbb{N}^*$ et $x \in \mathbb{R}$, on pose $P_n(x) = 1 + x + x^2 + \cdots + x^{2n} = \sum_{k=0}^{2n} x^k$.

1. À l'aide de l'ordinateur, tracer les courbes des fonctions P_n pour $-2 \leq x \leq 2$ et $1 \leq n \leq 10$. On utilisera la commande `plt.axis([-2, 2, 0, 5])` afin de cadrer la fenêtre graphique.

2. Pour $x \neq 1$ et $n \in \mathbb{N}^*$, montrer que $P_n'(x) = \frac{u_n(x)}{(x-1)^2}$, où u_n est une fonction polynomiale à déterminer.

3. Pour $n \in \mathbb{N}^*$, donner l'allure du tableau de variation de la fonction P_n. Montrer en particulier que P_n possède un unique minimum sur $\mathbb{R}$. Dans la suite, on notera a_n le réel où P_n atteint son minimum.

4. Créer une fonction informatique `A` prenant en argument un entier $n \in \mathbb{N}^*$, et renvoyant une valeur approchée de a_n. *Remarque : la méthode de Newton ne s'applique pas bien, utiliser la méthode dichotomique, voir scipy.optimize.bisect.*

5. Représenter graphiquement a_n en fonction de n pour $1 \leq n \leq 500$. Que peut-on conjecturer sur la limite de cette suite ?

6. Déterminer un équivalent simple de la quantité $\ln(2n + 1 - 2na_n)$, puis, en exploitant la relation $P_n'(a_n) = 0$, en déduire la limite de la suite $(a_n)_{n\in\mathbb{N}^*}$.

7. On pose maintenant $a_n = -1 + h_n$. Déterminer un équivalent de h_n lorsque n tend vers $+\infty$.

8. On pose $w_n = h_n - \frac{\ln n}{2n} - \frac{\ln 2}{n}$. À l'aide d'une représentation graphique, conjecturer la nature de la série $\sum w_n$.

9. Démontrer le résultat conjecturé à la question précédente.

Corrigé. On importe les modules utiles comme suit (comme dans les fiches !).

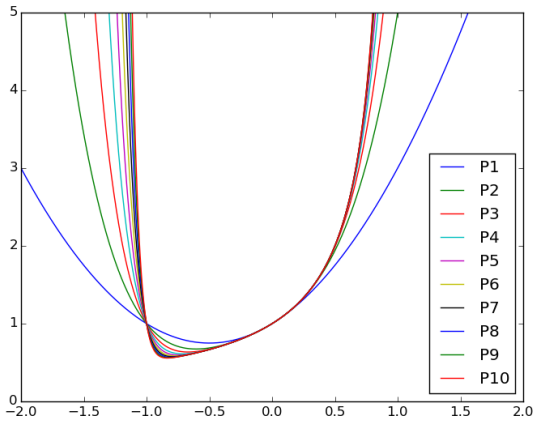
```
from numpy.polynomial import Polynomial
import scipy.optimize as resol
import matplotlib.pyplot as plt
import numpy as np
from math import *
```

1. On définit la fonction `P` qui renvoie P_n, puis une boucle pour tracer les courbes. Les polynômes sont des fonctions vectorielles, ici `f(T)` crée directement un tableau Numpy contenant les `f(x)` pour `x` dans `T`.

```
def P(n):
    return Polynomial([1]*(2*n+1))

T=np.linspace(-2,2,1000)
for n in range(1,11):
    f=P(n)
    plt.plot(T,f(T),label="P"+str(n))
plt.axis([-2, 2, 0, 5])
plt.legend(loc="lower right")
plt.show()
```

On obtient :



2. En écrivant que $P_n(x) = \frac{x^{2n+1}-1}{x-1}$, on obtient $P_n'(x) = \frac{2nx^{2n+1}-(2n+1)x^{2n}+1}{(x-1)^2}$.

3. Il suffit d'étudier u_n, on vérifie facilement que u_n' s'annule en 0 et 1, et l'étude des valeurs en 0 et 1 permet de montrer que P_n' s'annule en un certain $\alpha < 0$, et P_n est décroissante jusqu'en α puis croissante ensuite, d'où le résultat.

4. L'étude graphique indique que $a_n \in [-1, 0]$, on peut donc chercher via la méthode dichotomique (`resol.bisect`) un zéro de P_n' (obtenu avec `P(n).deriv()`).

```
def A(n):
    f=P(n).deriv()
    return resol.bisect(f, -1, 0)
```

La méthode de résolution standard (`resol.fsolve`, méthode de Newton, fournie dans les fiches) fonctionne aussi, en utilisant u_n et non pas P_n', et en partant du point -1 :

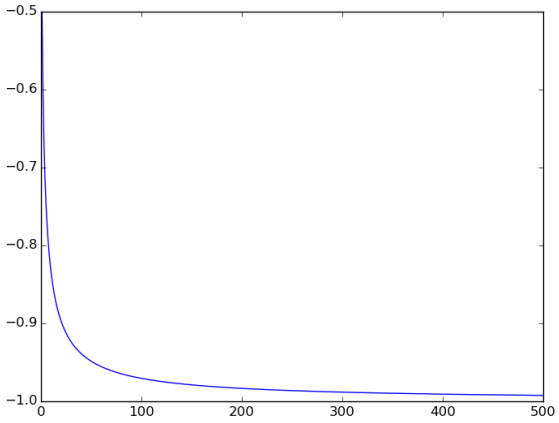
```
def A(n):
    L=[0]*(2*n+2)
    L[-1]=2*n
    L[-2]=-2*n-1
```

```
    L[0]=1
    f=Polynomial(L)
    return resol.fsolve(f,-1)
```

5. Par exemple :

```
L=list(range(1,501))
plt.plot(L,[A(x) for x in L])
plt.show()
```

On obtient :



Et on conjecture que la limite de a_n est -1.

6. $P_n(-1) = P_n(0) = 1$, donc $a_n \in]-1, 0[$. Ainsi, $2n+1-2na_n \in]2n+1, 4n+1,[$. Un équivalent simple de $\ln(2n+1-2na_n)$ est donc $\ln(n)$. La relation $P_n'(a_n) = 0$ donne $a_n^{2n}(2n+1-2na_n) = 1$. En passant au ln, on obtient $2n\ln|a_n| + \ln(2n+1-2na_n) = 0$, et donc :

$$|a_n| = \exp\left(-\frac{\ln(2n+1-2na_n)}{2n}\right) \underset{n\to+\infty}{\longrightarrow} 1$$

Donc a_n a bien pour limite -1.

7. $a_n \in]-1, 0[$, donc $h_n > 0$ et $|a_n| = 1 - h_n$. À partir de l'égalité vue un peu plus haut, on tire :

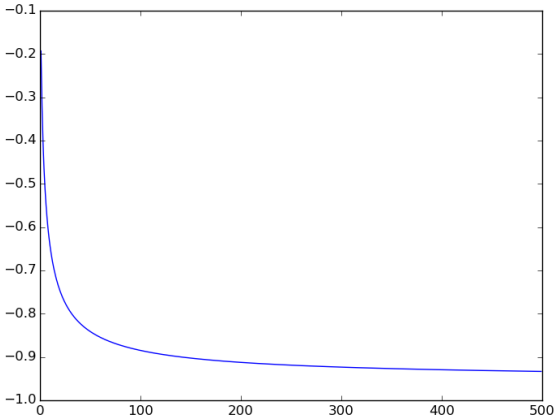
$$-h_n \underset{n\to+\infty}{\sim} \ln(1-h_n) = \frac{-\ln(2n+1-2na_n)}{2n} \underset{n\to+\infty}{\sim} -\frac{\ln(n)}{2n}$$

D'où l'équivalent $h_n \underset{n\to+\infty}{\sim} \frac{\ln(n)}{2n}$.

8. À la suite du code précédent, on exécute :

```
T=[0]
for n in L:
    T.append(T[-1]+A(n)+1-log(n)/(2*n)-log(2)/n)
plt.plot(L,T[1:])
plt.show()
```

Pour obtenir



La série des w_n semble donc être convergente.

9. On écrit que $a_n = -1 + \frac{\ln(n)}{2n} + v_n$, dont on sait que $v_n \underset{n\to+\infty}{=} o\left(\frac{\ln(n)}{n}\right)$, et on repart de $-2n\ln|a_n| = \ln(2n + 1 - 2na_n)$.

- D'une part,
$$-2n\ln|a_n| = -2n\ln\left(1 - \frac{\ln(n)}{2n} - v_n\right) = 2n\left(\frac{\ln(n)}{2n} + v_n + O\left(\frac{\ln(n)^2}{n^2}\right)\right)$$
- D'autre part,
$$\ln(2n + 1 - 2na_n) = \ln(4n + 1 - \ln(n) - 2nv_n) = \ln(n) + 2\ln(2) + O\left(\frac{\ln(n)}{n}\right)$$

On en déduit $v_n = \frac{\ln(2)}{n} + O\left(\frac{\ln(n)}{n^2}\right)$. Ainsi $w_n = O\left(\frac{\ln(n)}{n^2}\right)$, terme général d'une série absolument convergente.

Exercice 6. *Polynômes, intégration, calcul matriciel.*

On munit $\mathbb{R}[X]$ du produit scalaire défini par

$$\forall\ P, Q \in \mathbb{R}[X], \langle P, Q\rangle = \int_{-\infty}^{+\infty} P(t)Q(t)\,\mathrm{e}^{-t^2}\,\mathrm{d}t$$

1. Vérifier que $\langle .,. \rangle$ est bien un produit scalaire.

2. Pour $n \in \mathbb{N}$, on pose $I_n = \int_{-\infty}^{+\infty} t^n\,\mathrm{e}^{-t^2}\,\mathrm{d}t$. On admettra que $\int_{-\infty}^{+\infty} \mathrm{e}^{-t^2}\,\mathrm{d}t = \sqrt{\pi}$

(a) Trouver une relation encore I_n et I_{n-2}. Écrire une fonction récursive `Calcul(n)` qui renvoie la valeur de I_n.

(b) Calculer I_n pour tout $n \in \mathbb{N}$.

3. Écrire une fonction `ps(P,Q)` qui renvoie le produit scalaire ci-dessus, à l'aide de `Calcul`.

4. Soit $f : x \mapsto \mathrm{e}^{-x^2}$.

(a) Montrer qu'il existe une unique suite de polynômes $(H_n)_{n\in\mathbb{N}}$ telle que, pour tout $x \in \mathbb{R}$, $f^{(n)}(x) = \mathrm{e}^{-x^2} H_n(x)$.

(b) Trouver une relation entre H_{n+2}, H_{n+1} et H_n (solution : $H_{n+2} + 2xH_{n+1} + 2(n+1)H_n = 0$.)

(c) Écrire une fonction `Poly2(n)` qui renvoie la liste de taille $n+1$ des $(H_k)_{0\leq k\leq n}$.

(d) Écrire une fonction `Gram(n)` renvoyant la matrice des $\langle H_i, H_j\rangle$ pour $i, j \in [\![0, n]\!]$. Tester pour de petits n. Observation ?

(e) Prouver le résultat observé.

Corrigé. On ne développe pas trop ici la partie maths, cet exercice est long mais pas très difficile. Réponses aux questions théoriques :

— On montre facilement par IPP que $I_n = \frac{n-1}{2} I_{n-2}$ (et on voit que $I_k = 0$ pour k impair, par parité).

— Pour prouver la relation $H_{n+2} + 2xH_{n+1} + 2(n+1)H_n = 0$, il suffit de la vérifier pour $n = 0$, puis d'initier une récurrence. Pour cela, on utilise le fait que $H_n' = H_{n+1} + 2xH_n$, qu'il s'uffit de réinjecter dans l'équation $(H_{n+2} + 2xH_{n+1} + 2(n+1)H_n)' = 0$.

— Enfin, la preuve que les H_i sont orthogonaux (et même que $\|H_i\|^2 = 2^i i!\sqrt{\pi}$) se fait par intégration par parties répétées, en utilisant le fait que $\langle H_i, H_j \rangle = \int_{-\infty}^{+\infty} H_i(x) f^{(j)}(x)\,\mathrm{d}x$, et que le coefficient dominant de H_i est $\pm 2^i$.

Solution de la partie pratique :

```
from numpy.polynomial import Polynomial
import numpy as np
from math import *

def Calcul(n):
    if n%2==1:
        return 0
    elif n==0:
        return sqrt(pi)
    else:
        return (n-1)/2*Calcul(n-2)

def ps(P,Q):
    R=P*Q
    T=R.coef
    s=0
    for i in range(len(T)):
        s+=Calcul(i)*T[i]
    return s

def Poly2(n):
    L=[Polynomial([1]),Polynomial([0,-2])]
    X=Polynomial([0,1])
    for i in range(2,n+1):
        L.append(-2*X*L[-1]-2*(i-1)*L[-2])
    return L

def Gram(n):
    B=np.zeros((n+1,n+1))
    Hn=Poly2(n)
    for i in range(n+1):
        for j in range(n+1):
            B[i][j]=ps(Hn[i],Hn[j])
    return B
```

Un petit exemple :

```
>>> Gram(3)
array([[  1.77245385,   0.        ,   0.        ,   0.        ],
       [  0.        ,   3.5449077 ,   0.        ,   0.        ],
       [  0.        ,   0.        ,  14.17963081,   0.        ],
       [  0.        ,   0.        ,   0.        ,  85.07778484]])
```

Exercice 7. *Suites récurrentes : pas de module particulier à importer.*

Pour $n \in \mathbb{N}$, on pose $u_n = \arctan(n+1) - \arctan(n)$. Soit $\varepsilon = (\varepsilon_n)_{n\geq 0} \in \{0,1\}^{\mathbb{N}}$.

1. Montrer que la série de terme général $u_n\varepsilon_n$ converge et que $0 \leq \sum_{n=0}^{+\infty} \varepsilon_n u_n \leq \pi/2$.

2. **Un cas particulier.** Soit $x \in [0, \pi/2]$. On définit la suite des $(\varepsilon_n)_{n\in\mathbb{N}}$ par récurrence :

— si $x < \pi/4$, alors $\varepsilon_0 = 0$, sinon $\varepsilon_0 = 1$;

— pour $n \in \mathbb{N}$, si $x > \sum_{k=0}^{n} \varepsilon_k u_k + u_{n+1}$, alors $\varepsilon_{n+1} = 1$, sinon $\varepsilon_{n+1} = 0$.

(i) Programmer en Python une fonction `suite` qui prend en paramètre $(x, n) \in [0, \pi/2] \times \mathbb{N}$ et qui renvoie $\sum_{k=0}^{n} \varepsilon_k u_k$.

(ii) En testant pour des valeurs de x et $n = 100$, faire une conjecture.

(iii) Prouver la conjecture.

3. Le résultat de la question précédente est-il vrai dans le cas où $u_n = \frac{2}{3^{n+1}}$ et $x = 0.5$?

Corrigé. ***1.*** $0 \leq \varepsilon_n u_n \leq u_n$, et la série $\sum u_n$ converge vers $\pi/2$ par télescopage, d'où le résultat.

2. (i)

```
def suite(x,n):
    if x<pi/4:
        s=0
    else:
        s=atan(1) #pi/4
    for i in range(1,n+1):
        ti=atan(i+1)-atan(i)
        if s+ti<x:
            s+=ti
    return s
```

(ii) Par exemple :

```
>>> [(x, suite(x, 100)) for x in [1, pi/8, sqrt(2)]]
[(1, 0.9999382603820008), (0.39269908169872414, 0.39269725002790445),
(1.4142135623730951, 1.4141733536975594)]
```

On peut conjecturer que $\sum \varepsilon_n u_n = x$ dans ce cas.

(iii) Soit $n \geq 1$. On a $\sum_{k=n}^{+\infty} u_k = \pi/2 - \arctan(n)$, et $u_{n-1} = \arctan(n) - \arctan(n-1)$. Une étude de $x \mapsto \pi/2 - 2\arctan(x) + \arctan(x-1)$ sur $[1, +\infty[$ montre que $\sum_{k=n}^{+\infty} u_k \geq u_{n-1}$. Soit maintenant $x \in [0, \pi/2]$ et (ε_n) la suite construite par l'algorithme. Cette propriété permet de montrer que l'on a

$$0 \leq x - \sum_{k=0}^{n} \varepsilon_k u_k \leq \sum_{k=n+1}^{+\infty} u_k$$

pour tout n, et le résultat suit en faisant tendre n vers l'infini.

3. Non. On vérifie que pour tout $(\varepsilon_n) \in \{0,1\}^{\mathbb{N}}$, $\sum \varepsilon_n u_n \geq 2/3$ si $\varepsilon_0 = 1$ et $\sum \varepsilon_n u_n \leq 1/3$ si $\varepsilon_0 = 0$, ainsi on ne peut avoir $\sum \varepsilon_n u_n = 1/2$.

Exercice 8. *Tracé.*

On considère dans cet exercice la série $\sum_{n\geq 1} a_n$, avec $\forall n > 0$, $a_n = (-1)^n \ln(1 + 1/n)$.

1. Démontrer que la série $\sum_{n\geq 1} a_n$ converge.

2. a. Vérifier avec le logiciel la relation $\sum_{n\geq 1} a_n = \ln\left(\frac{2}{\pi}\right)$.

b. Démontrer avec rigueur le résultat.

3. Montrer que le rayon de convergence de la série entière $\sum_{n\geq 1} a_n x^n$ est 1. On note f sa somme sur $]-1, 1[$.

4. a. Faire un tracé de f avec le logiciel et vérifier que $f(x) \underset{x\to 1^-}{\to} \ln\left(\frac{2}{\pi}\right)$.

b. Démontrer avec rigueur le résultat.

5. a. Faire un tracé de f' avec le logiciel et vérifier que f' admet une limite en 1.

b. Démontrer avec rigueur le résultat.

Corrigé.

1. Le critère des séries alternées s'applique, car $\ln(1 + 1/n)$ est strictement décroissante, positive, et de limite nulle.

2. Vérifions, le code suivant permet d'obtenir a_n.

```
from math import *

def a(n):
    b=log(1+1/n)
    if n%2==0:
        return b
    else:
        return -b
```

Regardons pour un grand nombre de termes :

```
>>> print(sum([a(n) for n in range(1,10000)]))
-0.45163270528935584
>>> print(log(2/pi))
-0.4515827052894548
```

Montrons le rigoureusement. On peut examiner simplement les termes d'ordre pair de la somme partielle. Ainsi

$$\exp\left(\sum_{k=1}^{2n} a_k\right) = \prod_{k=1}^{2n} \exp(a_k) = \prod_{k=1}^{2n} (1+1/k)^{(-1)^k} = \prod_{i=1}^{n} \frac{(1+\frac{1}{2i})}{(1+\frac{1}{2i-1})} = \prod_{i=1}^{n} \frac{(2i+1)(2i-1)}{2i\times 2i} = \prod_{i=1}^{n} \frac{(2i+1)(2i-1)(2i)^2}{(2i)^4}$$

Par suite, en appliquant la formule de Stirling,

$$\exp\left(\sum_{k=1}^{2n} a_k\right) = \frac{(2n+1)!(2n)!}{16^n n!^4} \underset{n\to+\infty}{\sim} \frac{(2n+1)\times(2n)^{4n}e^{-4n}(4\pi n)}{16^n n^{4n} e^{-4n}(2\pi n)^2} = \frac{2n+1}{\pi n} \underset{n\to+\infty}{\longrightarrow} \frac{2}{\pi}$$

3. a_n est bornée, donc le rayon de convergence est au moins 1. Comme $(-1)^n a_n \underset{n\to+\infty}{\sim} \frac{1}{n}$, terme général d'une série divergente, la série entière ne converge pas en -1, donc le rayon de convergence est exactement 1.

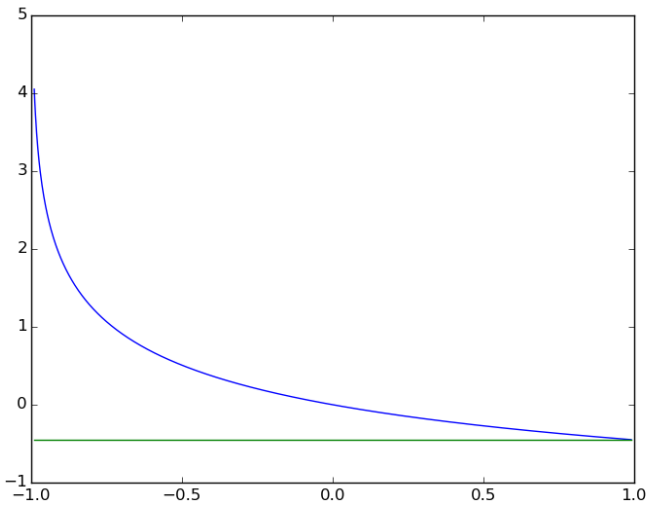
4. Le code suivant permet de tracer un graphique faisant apparaître la série entière et la constante $\ln(2/\pi)$. Une série entière converge vite (sauf au bord du disque de convergence), 1000 termes sont amplement suffisants pour estimer $f(x)$. On restreint le domaine à $[1-\varepsilon, 1+\varepsilon,]$, pour éviter les problèmes sur les bords du disque de convergence, et on prend 1000 points dans cet intervalle pour tracer le graphe.

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

A=[0]+[a(n) for n in range(1,1000)]
def f(x):
    return sum([A[i]*x**i for i in range(1,1000)])

T=np.linspace(-0.99,0.99,1000)
plt.plot(T,[f(x) for x in T])
plt.plot(T,[log(2/pi) for x in T])
plt.show()
```

Voici la figure obtenue :



Il semble bien que $f(x) \underset{x\to 1^-}{\to} \ln\left(\frac{2}{\pi}\right)$. Montrons le : il suffit de montrer que la série entière converge uniformément vers f sur l'intervalle $[1/2, 1]$ (par exemple). Or pour tout $x \in [1/2, 1]$, la série des $(-1)^n a_n x^n$ est une série alternée, dont le reste est majorée par le premier terme en valeur absolue. D'où

$$\left|\sum_{n=1}^{N} a_n x^n - f(x)\right| \leq |a_{N+1}|\, x^{N+1} \leq |a_{N+1}| \underset{N\to+\infty}{\longrightarrow} 0$$

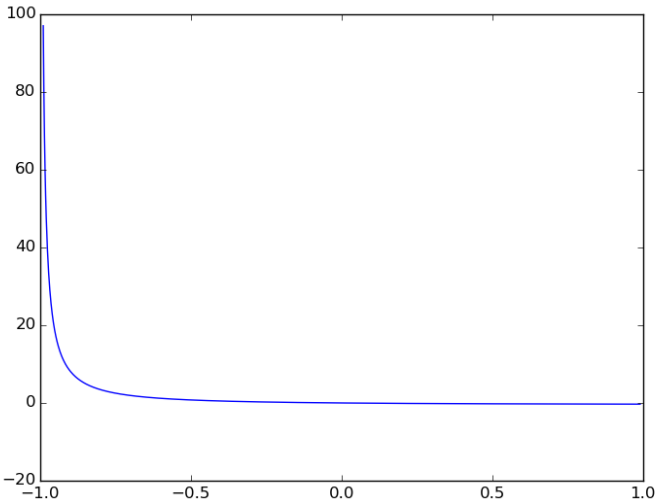
Ainsi, la convergence uniforme assure que f est continue sur $[1/2, 1]$, et donc $f(x) \underset{x\to 1^-}{\to} f(1) = \ln\left(\frac{2}{\pi}\right)$

5. On reprend l'approche précédente :

```
def g(x):
    return sum([A[i]*i*x**i for i in range(1,1000)])

T=np.linspace(-0.99,0.99,1000)
plt.plot(T,[g(x) for x in T])
plt.show()
```

Ce code produit :



f' semble donc bien avoir une limite en 1. Pour le montrer, on effectue un développement limité : $na_n = (-1)^n\left(1 - \frac{1}{2n} + O\left(\frac{1}{n^2}\right)\right)$. Ainsi, on découpe la série entière $\sum_{n=1}^{+\infty} na_n x^{n-1}$ en trois morceaux :

— $\sum_{n=1}^{+\infty}(-1)^n x^n = \frac{-x}{1+x}$ admet bien une limite en 1 ;

— $\sum_{n=1}^{+\infty}\frac{(-1)^n}{2n}x^n$ admet une limite en 1 pour les mêmes raisons qu'à la question précédente (on a une série entière convergente en 1, alternée) ;

— le dernier morceau converge normalement sur $[1/2, 1]$ et a donc bien une limite en 1 également.

Exercice 9. *Tracé, intégration.*

On considère les fonctions f et g définies sous condition de convergence par :

$$f : x \mapsto \int_0^{+\infty} \frac{\sin(t)}{x+t}\,\mathrm{d}t \qquad \text{et} \qquad g : x \mapsto \int_0^{+\infty} \frac{\mathrm{e}^{-xt}}{1+t^2}\,\mathrm{d}t$$

1. Montrer que f est définie sur $\mathbb{R}_+$.

2. Que donne l'outil informatique pour le calcul de $f(0)$, $f(1)$ et $f(10)$? Commenter.

3. Soit $A \in \mathbb{R}_+$, on pose $f_A : x \mapsto \int_0^A \frac{\sin(t)}{x+t}\,\mathrm{d}t$.

(a) Écrire une fonction Python de paramètres A et x qui calcule $f_A(x)$.

(b) Tracer simultanément les courbes représentatives de f_{50}, f_{100}, f_{200} et f_{500} sur l'intervalle $[0, 5]$. Commenter.

4. Montrer que la fonction g est définie et continue sur $\mathbb{R}_+$ et de classe $\mathcal{C}^2$ sur $\mathbb{R}_+^*$.

5. Tracer simultanément les courbes représentatives de g et f_{200}. Que peut-on conjecturer ?

6. Après avoir changé son expression à l'aide d'un changement de variable, montrer que la fonction f est de classe $\mathcal{C}^2$ sur $\mathbb{R}_+^*$.

7. Donner une équation différentielle d'ordre 2 vérifiée par f et g. Conclure.

Corrigé. Cet exercice est très classique : on vérifie que f et g sont solutions de la même équation différentielle d'ordre 2 sur $\mathbb{R}_+^*$, avec $g''(x) + g(x) = 1/x$ (et de même pour f). f et g diffèrent donc d'une solution de l'équation homogène (combinaison linéaire de sin et cos), mais comme elles ont même limite (zéro) en $+\infty$, elles sont égales.

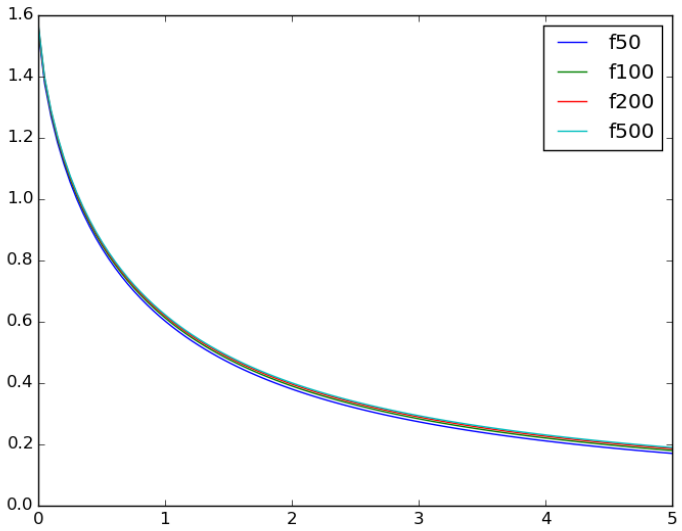
En Python, on utilise `quad` pour intégrer (il est possible de donner la borne $+\infty$ avec `np.inf`). Cette fonction renvoie un couple (I, e), où I est la valeur de l'intégrale et e une majoration de l'erreur commise.

```
def f(x):
    def h(t):
        return sin(t)/(x+t)
    return integr.quad(h,0,np.inf)[0] #on ne garde que l'intégrale, pas le terme d'erreur
```

On remarque que Python émet un message d'erreur lorsqu'on demande des valeurs de f, et donne un terme d'erreur grand, car la fonction $t \mapsto \frac{\sin(t)}{x+t}$ n'est pas intégrable (l'intégrale n'est que semi-convergente).

```
def f2(A,x):
    def h(t):
        return sin(t)/(x+t)
    return integr.quad(h,0,A)[0]

X=np.linspace(0,5,100) #100 points sur [0,5]
for A in [50, 100, 200, 500]:
    plt.plot(X,[f2(A,x) for x in X],label="f"+str(A)) #on trace chacune des f_A
plt.legend(loc="upper right") #légende à droite
plt.show()
```



On trace de même f_{200} et g, qui sont quasiment confondues.

Exercice 10. *Tracé, intégration, résolution d'équations différentielles.*

On considère $a \in \mathbb{R}$ et x_a une fonction de $\mathcal{C}^1(\mathbb{R}_+, \mathbb{R})$ vérifiant $x_a(0) = a$ et $\forall\, t \in \mathbb{R}_+$, $x_a'(t) = \cos(t) + \cos(x_a(t))$.

1. Utiliser la méthode d'Euler pour tracer le graphe de x_a sur $[0, T]$ pour diverses valeurs de a et T. *Note : je trouve assez aberrant de demander une méthode d'Euler, utiliser odeint !*
2. Si $x \in \mathcal{C}^0([0, T], \mathbb{R}) = E$ on pose $\phi(x) : t \mapsto a + \sin t + \int_0^t \cos(x(s))\,\mathrm{d}s$. Montrer que :

$$\forall n \in \mathbb{N},\ \forall (x, y) \in E^2,\ \forall t \in [0, T],\ |\phi^n(x)(t) - \phi^n(y)(t)| \leq \frac{t^n}{n!} \|x - y\|_\infty$$

3. Avec les notations précédentes, soit $x \in E$. Montrer que $(\phi^n(x))$ converge uniformément vers un élément $z \in E$, et que z vérifie les propriétés de x_a.
4. Pour x la fonction constante égale à 1, tracer les premiers termes de la suite $(\phi^n)_n$ pour $a = 1$ et $T = 1$.

Corrigé. Rappel : `odeint` permet de résoudre une équation de la forme $x'(t) = f(x(t), t)$ avec condition $x(t_0) = x_0$. On lui donne `x0` et un tableau de temps `tps`, et elle renvoie une approximation de la solution aux temps donnés par `tps`, le premier élément de `tps` étant t_0.

1. Le code suivant convient pour tracer x_a (en fonction de a et T) :

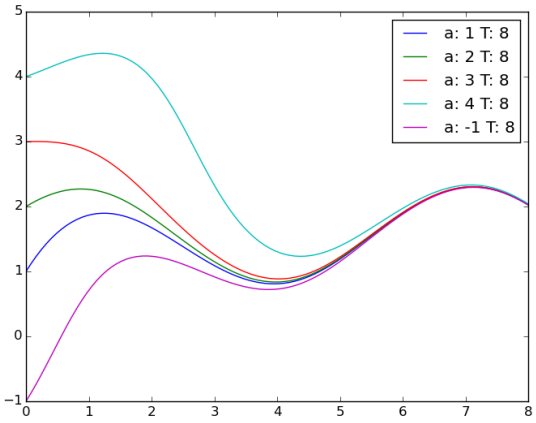
```
import scipy.integrate as integr
from math import *
import matplotlib.pyplot as plt
import numpy as np

def trace_sol(a,T):
    X=np.linspace(0, T, 100)
    def f(x,t):
        return cos(t)+cos(x)
    plt.plot(X, integr.odeint(f, a, X), label = "a: "+str(a)+" T: "+str(T))
```

Voici par exemple comment obtenir des tracés pour $a \in \{-1, 1, 2, 3, 4\}$ et $T = 8$:

```
for a,T in [(1,8), (2,8), (3,8), (4,8), (-1,8)]:
    trace_sol(a,T)
plt.legend()
plt.show()
```

On obtient :



Remarque : une version plus forte du théorème de Cauchy-Lipschitz au programme en classes préparatoires donne l'existence et l'unicité de x_a.

2. C'est assez imédiat par récurrence.

— Le cas $n = 0$ est évident car $\phi^0(x)(t) = x(t)$, en on a bien $|x(t) - y(t)| \leq \|x - y\|_\infty$.

— Pour l'hérédité, si la propriété est vrai au rang $n-1 \geq 0$, on a en utilisant le fait que cos est 1-lipshitzienne :

$$\begin{array}{rcl} |\phi^n(x)(t) - \phi^n(y)(t)| & = & \left|\int_0^t \cos(\phi^{n-1}(x)(s)) - \cos(\phi^{n-1}(y)(s))\,\mathrm{d}s\right| \\ & \leq & \int_0^t \left|\phi^{n-1}(x)(s) - \phi^{n-1}(y)(s)\right|\mathrm{d}s \\ |\phi^n(x)(t) - \phi^n(y)(t)| & \leq & \frac{t^n}{n!}\|x - y\|_\infty \end{array}$$

3. La série de fonctions définie par $u_0 = x$ et $u_n = \phi^n(x) - \phi^{n-1}(x)$ pour $n \geq 1$ converge normalement sur E vers une fonction z qui est continue (car les $\varphi^i(x)$ le sont). Or $\sum_{i=0}^n u_i = \phi^n(x)$. D'où la convergence uniforme de $\phi^n(x)$ vers z. Puisque $\phi^{n+1}(x)(t) = a + \sin(t) + \int_0^t \cos(\phi^n(x)(s))\,\mathrm{d}s$, on en déduit en faisant tendre n vers l'infini (légitime par convergence uniforme) que z vérifie $z(t) = a + \sin(t) + \int_0^t \cos(z(s))\,\mathrm{d}s$. z est donc dérivable et vérifie la même équation différentielle que x_a. (Remarque : on a en fait égalité mais c'est hors programme !).

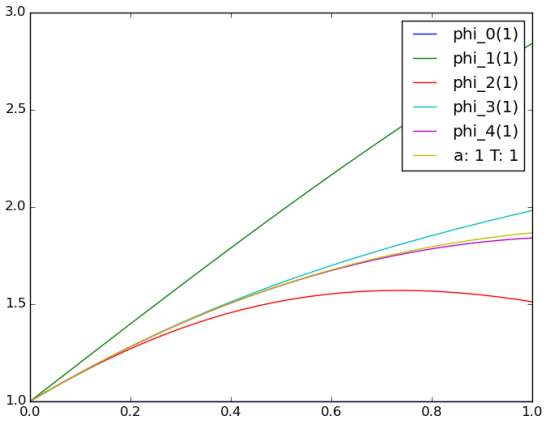
4. La fonction suivante trace $\phi^n(1)$ sur $[0, 1]$ (le deuxième cas « économise une application de `quad` ») :

```
def trace_phi(n):
    a,T=1,1
    def phi(n,x):
        if n==0:
            return 1
        elif n==1:
            return a+sin(x)+x
        else:
            return a+sin(x)+integr.quad(lambda t: cos(phi(n-1,t)),0,x)[0]
    X=np.linspace(0,T,100)
    plt.plot(X,[phi(n,x) for x in X],label="phi_"+str(n)+"(1)")
```

Le code suivant effectue un tracé des premières fonctions $\phi^i(1)$:

```
for n in range(5):
    trace_phi(n)
trace_sol(1,1)
plt.legend()
plt.show()
```

On obtient :



Remarque : il est très couteux de poursuivre le calcul, la complexité étant naturellement exponentielle.

Exercice 11. *Probabilités.*

On s'intéresse à la première apparition du motif « PF » dans un tirage infini de pile ou face, indépendants et non truqués. On note A_i l'événement « le motif PF apparaît pour la première fois au rang i ». On convient que le rang numéro 0 correspond au premier lancer (on a donc $\mathbb{P}(A_0) = 0$). On note $q_i = \mathbb{P}(A_i)$ pour $i \geq 0$, et T la variable aléatoire donnant le rang d'apparition du motif PF pour la première fois.

1. Écrire une fonction Python donnant le résultat d'une expérience.
2. En réitérant l'expérience un grand nombre de fois, conjecturer une valeur pour l'espérance de T (l'existence de cette espérance sera montrée plus tard).
3. Montrer que $\mathbb{P}(\cup_{i\geq 1} A_i) = 1$.
4. Décrire A_n pour $n \geq 1$ et en déduire la valeur de q_n.
5. Montrer que T est d'espérance finie, et donner $\mathbb{E}(T)$.
6. Reprendre l'exercice avec le motif « PP ».

Corrigé.

1. La fonction `pf()` tire à pile ou face (renvoie 0 ou 1 avec probabilité 1/2). La fonction `Exp()` réalise l'expérience : `a` est le dernier lancer, on en réalise un nouveau (`x`), et on s'arrête si `(x,a)` vaut `(0,1)`.

```
import numpy.random as rd

def pf():
```

```
    return rd.randint(2)

def Exp():
    a=pf()
    n=0
    while True:
        x=pf()
        n+=1
        if x==0 and a==1:
            return n
        else:
            a=x
```

2. Avec 100000 expériences :

```
>>> N=100000 ; print(sum([Exp() for _ in range(N)])/N)
2.99608
```

On conjecture que l'espérance de T est 3.

3. Soit $n \in \mathbb{N}$. $(\cup_{i\geq 1} A_i)^c = \cap_{i\geq 1}(A_i)^c \subset \cap_{i=0}^{n} B_i$, où B_i est l'évènement : « $a_{2i} \neq P$ ou $a_{2i+1} \neq F$ », avec a_i le résultat du lancer numéro i. Les B_i sont indépendants, de même probabilité 3/4. Ainsi $\mathbb{P}\left((\cup_{i\geq 1} A_i)^c\right) \leq (3/4)^{n+1} \underset{n\to+\infty}{\to} 0$.
Donc $\mathbb{P}(\cup_{i\geq 1} A_i) = 1$.

4. On note toujours a_i le résultat du i-ème lancé.

$$A_n = \cup_{i=0}^{n-1}\{a_0 = \cdots = a_{i-1} = F, a_i = a_{i+1} = \cdots = a_{n-1} = P, a_n = F\}$$

Les évènements intervenant dans la décomposition ci-dessus sont disjoints, et ont même probabilité $(1/2)^{n+1}$. Il y en a n, donc $q_n = \frac{n}{2^{n+1}}$.

5. La série des $\frac{n^2}{2^{n+1}}$ est sommable, donc

$$\mathbb{E}(T) = \sum_{n=1}^{+\infty} \frac{n^2}{2^{n+1}}$$

En introduisant la série entière $f(x) = \sum_{k=1}^{+\infty} n^2 x^n$, de rayon de convergence 1, on peut écrire :

$$f(x) = x\sum_{n=1}^{+\infty} n(n+1)x^{n-1} - x\sum_{n=1}^{+\infty} nx^{n-1} = x\left(\sum_{i=0}^{+\infty} x^i\right)'' - x\left(\sum_{i=0}^{+\infty} x^i\right)' = \frac{2x}{(1-x)^3} - \frac{x}{(1-x)^2}$$

Et $\mathbb{E}(T) = \frac{1}{2} f\left(\frac{1}{2}\right) = 3$.

6. On montre de même que l'espérance du rang x d'apparition du premier motif PP est finie (majorer $\mathbb{P}(x \geq n)$). En conditionnant suivant les résultats du premier ou des deux premiers lancers (évènements de la forme $PP*$, $F*$ ou $PF*$), on peut écrire que x vérifie :

$$x = \frac{1}{4} + \frac{1}{2}(1+x) + \frac{1}{4}(2+x)$$

dont la solution $x = 5$ est confirmée par l'expérience :

```
def Exp2():
    a=pf()
    n=0
    while True:
        x=pf()
        n+=1
        if x==1 and a==1:
            return n
        else:
            a=x
```

```
>>> N=100000 ; print(sum([Exp2() for _ in range(N)])/N)
4.96774
```

Exercice 12. *Probabilités.*

Soit n un entier supérieur ou égal à 2.

1. (i) On considère X et Y deux variables aléatoires indépendantes suivant la loi uniforme dans $\mathbb{Z}/n\mathbb{Z}$. Conjecturer la loi de $X+Y$.
   (ii) Conjecturer la loi de $X+Y$, X et Y étant toujours supposées indépendantes, X suit la loi uniforme dans $\mathbb{Z}/n\mathbb{Z}$, et Y est une variable aléatoire à valeurs dans $\mathbb{Z}/n\mathbb{Z}$.
   (iii) Sous les hypothèses de (i), conjecturer la valeur de $\mathbb{P}(XY=1)$.
   (iv) Démontrer la conjecture faite en (i).
2. (i) Soit $(G,.)$ un groupe fini. On considère X et Y deux variables aléatoires indépendantes à valeurs dans G, X suit la loi uniforme. Déterminer la loi de XY. Le résultat demeure-t-il sans l'hypothèse d'indépendance ?
   (ii) Démontrer la conjecture faite en (1.ii).
3. Démontrer la conjecture faite en (1.iii).
4. Soit p un nombre premier, X et Y deux variables aléatoires indépendantes à valeurs dans $\mathbb{Z}/p\mathbb{Z}$, X suivant la loi uniforme. Donner la loi de XY.

Corrigé. Cet exercice est facile, et il n'y a même pas besoin de Python pour avoir une intuition du résultat... Je vous laisse vous amuser à simuler si vous le souhaitez !

Exercice 13. *Probabilités.*

Soit (Ω, T, P) un espace probabilisé, X une variable aléatoire à valeurs dans $\mathbb{N}$, $(X_n)_{n\geq 1}$ une suite de variables aléatoires i.i.d suivant la loi de X et N une variable aléatoire indépendante des X_i et à valeurs dans $\mathbb{N}$. Pour $\omega \in \Omega$, on pose $S(\omega) = \sum_{k=1}^{N(\omega)} X_k(\omega)$.

1. Soient G_X, G_S, G_N les séries génératrices de X, S et N. Montrer :
$$\forall\, t \in [0,1], G_S(t) = G_N \circ G_X(t)$$
2. On suppose que X et N possèdent une espérance. Montrer que S possède une espérance et la calculer.
3. On suppose que X et N ont un moment d'ordre 2. Montrer que S possède un moment d'ordre 2 et calculer $\mathbb{E}(S^2)$.
4. On étudie la transmission du nom de famille au cours des générations dans une société patriarcale. On suppose que le nombre de descendants masculins d'un individu suit une loi de Poisson de paramètre $\lambda \in]0, +\infty[$. On note Z_0 le nombre d'individus masculins au début de l'étude, Z_n le nombre de descendants à la n-ième génération. On suppose que $Z_0 = 1$.
   (i) Écrire une fonction python renvoyant le nombre de descendants masculins à la n-ième génération.
   (ii) Fixer λ et n. Calculer une moyenne, sur un grand nombre de mesures, du nombre de descendants masculins. Comparer à $\mathbb{E}(Z_n)$.

Corrigé. C'est classique et a déja été vu en cours ! Je donne simplement un code possible pour la simulation.

```
import numpy.random as rd

def nb_descendants(n, lamb):
    a=1 #à la génération 0
    for i in range(n):
        a=sum(rd.poisson(lamb,a)) #a tirages de P(lamb), qu'on somme.
    return a
```

Testons avec λ ni trop grand ni trop petit (on prend 1.2), et 20 générations :

```
>>> sum([nb_descendants(20,1.2) for _ in range(1000)])/1000
38.837
>>> 1.2**20
38.33759992447472
```

C'est cohérent !