CLASSE DE 2TSI

PREPARATION PLANCHES ORAUX PYTHON

CORRECTION

---

*Planche*01.*CCS*2018

1.
```
>>> def factorielle(n) :
        p = 1
        for k in range(1, n+1) :
            p = p*k
        return p
```
Remarque : on peut utiliser $mt.factorial$ à la place de $factorielle$, à condition d'avoir chargé $math\ as\ mt$
Puis on crée la fonction $BINOM(n,k)$
```
>>> def BINOM(n,k) :
        if not(0 <= k <= n) :
            return 0
        return factorielle(n)//(factorielle(k) * factorielle(n-k))
```
2.
```
>>> import math as mt
>>> def rep(t,n,p) :
        u = mt.floor(t)
        if u <= -1 :
            return 0
        elif u >= n :
            return 1
        else :
            return sum(BINOM(n,k) * (p**k) * ((1-p)**(n-k)) for k in range(0,u+1))
>>> rep(2,5,0.3), rep(6,5,0.3), rep(-2,5,0.3)
(0.836919999, 1, 0)
```
3. Tracons la fonction $t \mapsto rep(t,n,p)$ pour $t \in [-1, n+1]$ pour $(n,p) = (5, 0.3)$ et $(n,p) = (15, 0.7)$, côte à côte.
```
>>> import matplotlib.pyplot as plt ; import numpy as np
>>> X = np.arange(-1,6,0.001) ; XX = np.arange(-1,16,0.001)
>>> Y = [rep(t,5,0.3) for t in X] ; YY = [rep(t,15,0.7) for t in XX]
>>> plt.subplot(221) ; plt.plot(X, Y, color =' 0') ; plt.subplot(222) ;
plt.plot(XX, YY, color =' 0') ; plt.show()
```
[width=9cm]figure-planche01.eps
Remarque : On a tapé Y et YY ainsi car $rep(X,5,0.3)$ et $rep(XX,15,0.7)$ envoyent un message d'erreur.

*Planche*02.*CCS*2017

1. >>> *def f1(L) :*
neg = 0 ; pos = 0
for k in range(len(L)) :
if L[k] <= 0 :
neg += 1
else :
pos += 1
print('nombretermesnegatifsounuls', neg)
print('nombretermespositifs', pos)

On tape par exemple
>>> *f1([−4, 8, 9, 0, 1])*
nombre termes negatifs ou nuls 2 nombre termes positifs 3

2. >>> *def f2(L) :*
Lneg = [] ; Lpos = []
for k in range(len(L)) :
if L[k] <= 0 :
Lneg.append(L[k])
else :
Lpos.append(L[k])
SomCubNeg = sum((Lneg[k] ∗ ∗3) for k in range(len(Lneg)))
SomCarPos = sum((Lpos[k] ∗ ∗2) for k in range(len(Lpos)))
return SomCubNeg , SomCarPos

Puis on tape
>>> *f2([−4, 8, 9, 0, 1])*
(−64, 146)

3. >>> *def f3(L) :*
res = [] ; maxi = max(L)
for k in range(len(L)) :
if L[k] == maxi :
res.append(k)
return res

Puis on tape
>>> *f3([4, 4, 5, −2, 4, 0]) , f3([4, 4, −5, −2, 4, 0])*
[2], [0, 1, 4]

---

*Planche*03.*CCS*2019

1. >>> *import numpy as np*
>>> *def Newtrace(A) :*
n = np.size(A, 0)
return sum(A[i, i] for i in range(n))

On tape
>>> *Newtrace(np.array([[2, 1, 3], [1, 4, 1], [0, 1, 1]]))*
7

Remarque : on peut utiliser directement *np.trace(A)*

2. • Pour $\phi : M_n() \to M_n(),\ M \mapsto \mathrm{Tr}\,(M)\, I_n$
>>> *def phi(M) :*

n = np.size(M, 0)
return Newtrace(M) ∗ np.eye(n)

On tape par exemple
>>> *phi(np.array([[2, 1, 3], [1, 4, 1], [0, 1, 1]]))*
array([[7., 0., 0.], [0., 7., 0.], [0., 0., 7.]])

• Pour $\psi : M_n() \to M_n(),\ M \mapsto M - \phi(M)$
>>> *def psi(M) :*

return M − 2 ∗ phi(M)

On tape par exemple
>>> *psi(np.array([[2, 1, 3], [1, 4, 1], [0, 1, 1]]))*
array([[−12., 1., 3.], [1., −10., 1.], [0., 1., −13.]])

---

*Planche*04.*CCS*2019

1. >>> *import scipy.integrate as integr*
>>> *import numpy as np*
>>> *def f(t) :*

return t ∗ np.arctan(t)/((1 + t ∗ ∗2) ∗ ∗2)

>>> *def G(x) :*

```
        return integr.quad(f, 1/x, x)[0]
```
On tape :
```
>>> G(3)
```
$$0.31415926535897865$$
Cela ressemble à $\pi/10$.

2. On va tracer ensemble le graphe de G (en noir) avec la droite $y = \dfrac{\pi}{8}$ (toujours en noir) et le graphe de la fonction $y = \dfrac{\pi}{8}\,\dfrac{x^2-1}{1+x^2}$ (en rouge).
```
>>> import matplotlib.pyplot as plt ; import numpy as np

    >>> def A(x) :

            return np.pi/8
    >>> def h(x) :

            return np.pi/8 * (x**2 - 1)/(x**2 + 1)
    >>> X = np.linspace(0.001, 10, 500) ; Y = [G(x) for x in X] ; Z = [A(x) for x in X]
>>> plt.plot(X, Y, color =' 0') ; plt.plot(X, h(X), color =' r') ;
plt.plot(X, Z, color =' 0') ; plt.show( )
```
[width=8cm]figure-planche04.eps

Remarque : On a tapé Y et Z ainsi car si l'on tape à la place respectivement $G(X)$ et $np.pi/8$, on a droit à un message d'erreur.

---

*Planche*05.*CCS*2019

1. Posons $f_n(x) = \ln\left(2\sin\dfrac{x}{2}\right)\cos(nx)$ pour tout $n \in$. La fonction f_n est continue sur $]0,\,\pi]$ et le seul problème est donc en 0^+. On utilise un développement limité à l'ordre 3, au voisinage de 0^+,
$$f_n(x) = \ln\left(2\left(\frac{x}{2} - \frac{x^3}{48} + o(x^3)\right)\right)\left(1 - \frac{n^2x^2}{2} + o(x^2)\right)$$
$$= \ln x + \ln\left(1 - \frac{x^2}{24} + o(x^2)\right)\left(1 - \frac{n^2x^2}{2} + o(x^2)\right) = \ln x + o(x).$$
Comme $x \mapsto \ln x$ est intégrable sur $]0, a]$ avec $a > 0$, on en déduit que I_n existe pour tout n.

2.
```
>>> import scipy.integrate as integr
>>> import numpy as np
>>> def fn(t) :
        return np.log(2 * np.sin(t/2)) * np.cos(1000 * t)
>>> integr.quad(fn, 0, np.pi)[0]  # par la fonction Python integr.quad
    IntegrationWarning : The integral is probably divergent, or slowly convergent.
```
$$-0.008269585083549684$$
```
>>> integr.quad(fn, 0.001, np.pi)[0]
```
$$0.0005559484974756285$$
```
>>> def TRAPEZES(f, a, b, n) :
        h = (b - a)/float(n) ; z = 0.5 * (f(a) + f(b))
        for i in range(n) :
            z = z + f(a + i * h)
        return h * z

>>> TRAPEZES(fn, 0.000001, np.pi, 2000) , TRAPEZES(fn, 0.0000001, np.pi, 200000)
```
$$-0.027665727145126547 \qquad -0.0018480965132166626$$
La convergence vers 0 est en effet lente.

Remarque : On veut une précision de 0.001. Pour cela, on rappelle la majoration de l'erreur de la méthode des Trapèzes. C'est $\dfrac{(b-a)^3}{12n^2}M_2$, où $M_2 = \sup\limits_{t\in[a,b]} |f''(t)|$. Mais f est alors de classe C^2 sur $[a, b]$, ce n'est pas le cas de la fonction de l'énoncé. Donc on ne peut pas utiliser cette formule. D'ailleurs si l'on calcule la dérivée de $x \mapsto \ln\left(2\sin\dfrac{x}{2}\right)\cos(1000x)$, on remarque qu'elle n'est pas bornée en 0^+.

---

*Planche*06.*ENSAM*2019

1.
```
>>> importnumpyasnp
>>> def typeH(n) :

        if n == 1 :
            return True
        if n <= 0 or n != np.floor(n) :
            return False
        k = 0 ; l = 0 ; m = 0 ; n2 = n ; n3 = n ; n5 = n
        while n2 % 2 == 0 :
```

```
        n2 /= 2 ; k += 1
    while n3 % 3 == 0 :
        n3 /= 3; l += 1
    while n5 % 5 == 0 :
        n5 /= 5 ; m += 1
    if n == (2**k)*(3**l)*(5**m) :
        return True
    else :
        return False
```

Faisons quelques essais.

```
>>> typeH(2) , typeH(34) , typeH(1) , typeH(5.7)
                                    True False True False
```

Affichons les *compt* premiers nombres de type H.

```
>>> def DisplaytypeH(compt) :
        L = []; n = 0; newcompt = compt
        while newcompt >= 1 :
            n += 1
            if typeH(n) :
                L.append(n); newcompt -= 1
        return L
```

On fait des essais

```
>>> DisplaytypeH(10)
                                    [1, 2, 3, 4, 5, 6, 8, 9, 10, 12]
>>> DisplaytypeH(100)
                    [1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, 18, ...1440, 1458, 1500, 1536]
```

2. Faisons un programme qui renvoie sous forme d'une liste les puissances de 2, 3 et 5 si n est un nombre de type H.

```
>>> def PowertypeH(n) :

        k = 0; l = 0; m = 0; n2 = n; n3 = n; n5 = n
        if n == 1 :
            return [0,0,0]
        if n <= 0 or n != np.floor(n) :
            print('probleme impossible')
        while n2 % 2 == 0 :
            n2 /= 2 ; k += 1
        while n3 % 3 == 0 :
            n3 /= 3 ; l += 1
        while n5 % 5 == 0 :
            n5/ = 5 ; m+ = 1
        if n == (2**k)*(3**l)*(5**m) :
            return [k,l,m]
        else :
            print('ce nombre non de type H')
```

On fait des essais.

```
>>> PowertypeH(5) , PowertypeH(56) , PowertypeH(1500) , PowertypeH(1536)
                        [0,0,1] ce nombre non de type [2,1,3] , [9,1,0]
```

3.

```
>>> def followingTypeH(L) :

        valeur = L[len(L)-1] + 1
        while typeH(valeur) == False :
            valeur += 1
        return valeur
```

On fait des exemples.

```
>>> followingTypeH([2,3,5]) , followingTypeH([1500,1536]) , followingTypeH([1600])
                                    6  1600  1620
```

Ce programme permet donc de donner le successeur de n'importe quel nombre de type H.

---

*Planche*07.*CCINP*

1. Il n'est pas évident avec Python de tracer une surface définie implicitement, c'est-à-dire du type $F(x,y,z) = 0$ mais par contre, on sait mieux faire avec une surface définie par $z = F(x,y)$. Dans l'aide fournie à CCS, il y a d'ailleurs un exemple avec un tel type de surface. Donc on commence par transformer l'équation de notre surface. On suppose x et y fixés et z est l'inconnue. Notre équation devient une classique équation du second degré d'inconnue z.

$$z^2 + 2xyz + x^2 + y^2 - 1 = 0 \Rightarrow z = -xy \pm \sqrt{x^2y^2 - x^2 - y^2 + 1}.$$

On a par ailleurs : $x^2y^2 - x^2 - y^2 + 1 = (1-x^2)(1-y^2) = (x^2-1)(y^2-1)$.

```
>>> import numpy as np; import matplotlib.pyplot as plt
>>> from mpl_toolkits.mplot3d import Axes3D
>>> def f1(x,y) : return - xy + np.sqrt((1-x**2)*(1--y**2))
```

```
>>> def f2(x,y) : return - xy - np.sqrt((1-x**2)*(1--y**2))
>>> X = np.arange(-1,1,0.015) ; Y = np.arange(-1,1,0.015) ; X, Y = np.meshgrid(X,Y)
>>> ax = Axes3D(plt.figure()) ; Z1 = f1(X,Y) ; Z2 = f2(X,Y)
>>> ax.plot_surface(X,Y,Z1) ; ax.plot_surface(X,Y,Z2) ; plt.show()
```

[width=9cm]figure-planche07.eps

2. `>>> def appartient(M) :`

```
    return M[0]**2+M[1]**2+M[2]**2 + 2*M[0]*M[1]*M[2] - 1 == 0
```

On fait des exemples.

```
>>> appartien([0,0,0]), appartien([1,1,1]), appartien([-1,-1,-1])
(False, False, True)
```

---

*Planche*08.*CCINP*

1. L'idée est de définir une fonction égale à f sur $[-\pi,\pi]$ et qui est la fonction nulle ailleurs puis de tracer $t \mapsto f(t+2\pi)+f(t)+f(t-2\pi)$ sur $[-2\pi,2\pi]$. On obtiendra bien le graphe sur $[-2\pi,2\pi]$ de la fonction de période 2π de valeur $\pi^2/12 - t^2/4$ sur $[-\pi,\pi]$.

```
>>> import matplotlib.pyplot as plt ; import numpy as np
>>> def f(t) :
    if t >= -np.pi and t <= np.pi :
        return (np.pi**2)/12 - t**2/4
    else :
        return 0
>>> def newf(t) :

    return f(t+2*np.pi) + f(t) + f(t-2*np.pi)

    >>> X = np.arange(-2*np.pi,2*np.pi,0.001)
>>> plt.plot(X,newf(X),color='0') ; plt.show()
```

[width=9cm]figure-planche01.eps

2. On peut remarquer que f est paire et donc pour tout n entier non nul, $b_n = 0$. Il reste à calculer les coefficients a_n.

- $a_0 = \dfrac{1}{T}\displaystyle\int_{-T/2}^{T/2} f(t)\,dt = \frac{2}{T}\int_0^{T/2} f(t)\,dt = \frac{1}{\pi}\int_0^{\pi}\left(\frac{\pi^2}{12}-\frac{t^2}{4}\right)dt = \frac{1}{\pi}\left[\frac{\pi^2 t}{12}-\frac{t^3}{12}\right]_0^{\pi} = 0.$
- Pour tout $n \in^{\star}$, $a_n = \dfrac{2}{T}\displaystyle\int_{-T/2}^{T/2} f(t)\cos(nt)\,dt = \frac{4}{T}\int_0^{T/2} f(t)\cos(nt)\,dt = \frac{2}{\pi}\int_0^{\pi}\left(\frac{\pi^2}{12}-\frac{t^2}{4}\right)\cos(nt)\,dt.$

Ici $\omega = \dfrac{2\pi}{2\pi} = 1$. Et il reste à faire une double intégration par parties. Posons $I_n = \displaystyle\int_0^{\pi}\frac{t^2}{4}\cos(nt)\,dt$ et trouvons I_n par deux intégrations par parties successives.

$$I_n = -\int_0^{\pi}\frac{t}{2}\frac{\sin(nt)}{n}\,dt + \left[\frac{t^2}{4}\right)\frac{\sin(nt)}{n}{}_0^{\pi}.$$

Le terme entre crochets est nul. On réintègre par parties,

$$I_n = -\int_0^{\pi}\frac{1}{2}\frac{\cos(nt)}{n^2}\,dt + \left[-\frac{t}{2}\right)\frac{\cos(nt)}{n^2}{}_0^{\pi}.$$

Ce qui donne $I_n = -\dfrac{1}{2}\left[\dfrac{\sin(nt)}{n^3}\right]_0^{\pi} - \dfrac{\pi}{2}\dfrac{(-1)^n}{n^2} = \dfrac{\pi(-1)^{n+1}}{2n^2}$.

Alors $a_n = \dfrac{2}{\pi}\displaystyle\int_0^{\pi}\left(\frac{\pi^2}{12}-\frac{t^2}{4}\right)\cos(nt)\,dt = \frac{2}{\pi}\left(\left[\frac{\pi^2\sin(nt)}{12n}\right]_0^{\pi} + \frac{\pi(-1)^{n+1}}{2n^2}\right) = \frac{(-1)^{n+1}}{n^2}$.

3. On utilise le théorème de Dirichlet car f est de classe C^1 par morceaux et est continue sur $[-\pi,\pi]$. On a alors immédiatement, pour tout $t \in [-\pi,\pi]$, $\dfrac{\pi^2}{12}-\dfrac{t^2}{4} = \displaystyle\sum_{n=1}^{+\infty}(-1)^{n+1}\frac{\cos(nt)}{n^2}$.

Puis on applique cette égalité avec $x = 0$ et : $\displaystyle\sum_{n=1}^{+\infty}\frac{(-1)^{n+1}}{n^2} = \frac{\pi^2}{12}$.

4. Écrivons une fonction *CalculSom* prenant en paramètre un entier N non nul et renvoyant la valeur de la somme partielle $\displaystyle\sum_{k=1}^{N}\frac{(-1)^{k+1}}{k^2}$.

```
>>> def CalculSom(N) :

    S = 0
    for k in range(1,N+1) :
        S = S + (-1)**(k+1)/(k**2)
    return S
    >>> CalculSom(10) , CalculSom(100) , CalculSom(1000)
        (0.8179621756109852, 0.8224175333741286, 0.8224665339241114)
>>> importnumpyasnp ; np.pi**2/12
```

0.8224670334241132

---

*Planche*09.*CCINP*

1. On voit pour commencer que x est définie et dérivable sur $\mathbb{R}\setminus\{-1,1\}$ et que y est définie et dérivable sur $\mathbb{R}\setminus\{1\}$.

Puis pour tout $t\in\mathbb{R}\setminus\{-1,1\}$, $x'(t)=\dfrac{-2t^2+2t-2}{(t^2-1)^2}$ et pour tout $t\in\mathbb{R}\setminus\{1\}$, $y'(t)=\dfrac{t^2-2t-3}{(t-1)^2}$.

L'équation $-2t^2+2t-2=0$ a pour discriminant $\Delta=-12<0$ donc n'a pas de solutions réelles et comme le coefficient devant t^2 est négatif, $x'(t)<0$. Donc x est strictement décroissante sur chacun des intervalles $]-\infty,-1[$, $]-1,1[$ et $]1,+\infty[$. Puis on a les limites :

$\lim\limits_{t\to-\infty}x(t)=0$, $\lim\limits_{t\to-1^-}x(t)=-\infty$, $\lim\limits_{t\to-1^+}x(t)=+\infty$, $\lim\limits_{t\to1^-}x(t)=-\infty$, $\lim\limits_{t\to1^+}x(t)=+\infty$, $\lim\limits_{t\to+\infty}x(t)=0$

L'équation $t^2-2t-3=0$ a pour discriminant $\Delta=16$ et a pour solution $t=-1$ et $t=3$ et comme le coefficient devant t^2 est positif, $y'(t)$ est négatif pour $t\in[-1,1[\cup]1,3]$ et positif pour $t<-1$ ou pour $t>3$. Puis on a les limites :

$\lim\limits_{t\to-\infty}y(t)=-\infty$, $\lim\limits_{t\to1^-}y(t)=-\infty$, $\lim\limits_{t\to1^+}y(t)=+\infty$, $\lim\limits_{t\to+\infty}y(t)=+\infty$

TAFEL in x und y miteinander ?

2. On remarque que $\lim\limits_{t\to-1^-}x(t)=-\infty$, $\lim\limits_{t\to-1^+}x(t)=+\infty$ et $y(-1)=-2$ donc la droite $y=-2$ est une asymptote horizontale pour (Γ). Puis $\lim\limits_{t\to\pm\infty}x(t)=0$, $\lim\limits_{t\to\pm\infty}y(t)=\pm\infty$, donc $x=0$ est une asymptote verticale pour (Γ).

3. `>>> def abscisse(t) :`

`return (2*t - 1)/(t**2 - 1)`
`>>> def ordonnee(t) :`

`return (t**2 + 3)/(t - 1)`

4. Pour le tracé, on va éviter de prendre $t=1$ car x et y ne sont pas bornées en cette valeur. On prendra $1-\alpha$ à la place de 1 pour I_1 et $1+\alpha$ à la place de 1 pour I_2. Enfin, α devant être positif et proche de 0, prenons 10^{-5} soit en Python $1E-5$

Tracer avec Python la courbe $t\mapsto\dfrac{y(t)}{x(t)}$ sur $I_1=\left[\dfrac{2}{3},1\right[$. Faire de même sur $I_2=\left]1,\dfrac{4}{3}\right]$.

Tracer la courbe $t\mapsto y(t)-8x(t)$ sur I_1 et sur I_2. Que constate t-on ? En déduire une asymptote oblique à (Γ).

Faire le tracé à la main de (Γ). Illustrer avec Python en tracant l'arc (Γ) et en incluant dans le dessin les diverses asymptotes.

---

*Planche*10.*CCINP*

1. On rappelle que si l'on charge *numpy.random* avec l'alias *rd*, *rd.random*( ) renvoie un réel entre 0 et 1. Ainsi *rd.random*( ) $<p$ quantifie la probabilité p et donc si *rd.random*( ) $<p$ alors la personne ment. L'idée est de partir d'un compteur *compt* qui initialement vaut 1 puis on lance la boucle. À chaque incrémentation de la boucle, si la personne ne ment pas, *compt* ne change pas et si la personne ment 1 devient 0 et 0 devient 1 donc *compt* est remplacé par $1-$*compt*. En sortie de boucle, si *compt* vaut 1, c'est que l'on retrouve le message initial et si par contre *compt* vaut 0, le message final est le contraire du message initial.

```
>>> import numpy.random as rd
>>> def Chaine(n,p) :
        compt = 1
        for i in range(1,n+1) :
            if rd.random( ) p :
                compt = 1 - compt
        return compt
>>> Chaine(5,0.9) , Chaine(15,0.5) , Chaine(3.0.9)
 (1, , 0 , 0 )
```

2. Si un message est transmis correctement, $Chaine(n,p)$ renvoie 1 et sinon 0, donc on somme N fois $chaine(n,p)$, ce qui renvoie le nombre de messages transmis correctement. Pour avoir la proportion, il suffit de diviser le résultat par N.

```
>>> def simul(N,n,p) :
        S = 0
        for i in range(1,N+1) :
            S = S + chaine(n,p)
        return S/N
```

Faisons tourner la bestiole.

```
>>> simul(3000,200,0.2) , simul(3000,200,0.8)
(0.5073333333333333, 0.49733333333333335)
```

Il semble que l'on tende vers 1/2, quelquesoit la valeur de p.

3. On note A_n l'événement : la $n$ème personne transmet le message initial alors $\left\{A_n,\overline{A_n}\right\}$ est un système complet d'événements et on a $p_n=P(A_n)$ pour tout n. Avec la convention $p_0=1$, on peut démarrer à $n=0$.

$$p_{n+1} = P(A_{n+1}) = P(A_n)P_{A_n}(A_{n+1}) + P(\overline{A_n})P_{\overline{A_n}}(A_{n+1}) = p_n(1-p) + (1-p_n)p.$$
En effet, $P_{A_n}(A_{n+1}) = p$ et $P_{\overline{A_n}}(A_{n+1}) = 1-p$.
Il reste : $p_{n+1} = (1-2p)p_n + p$. Puis si l'on pose $q_n = p_n - 1/2$,
$$q_{n+1} + 1/2 = (1-2p)(q_n + 1/2) + p \Rightarrow q_{n+1} + 1/2 = q_n + 1/2 - 2pq_n \Rightarrow q_{n+1} = (1-2p)q_n.$$
Ainsi (q_n) est une suite géométrique de raison $1-2p$. Alors pour tout $n \in$, $q_n = (1-2p)^n q_0$ et comme $q_0 = 1/2$,
$$p_n - \frac{1}{2} = (1-2p)^n \frac{1}{2} \Rightarrow p_n = \frac{1}{2} + (1-2p)^n \frac{1}{2}.$$
Il reste à remarquer que comme $p \in]0,1[$, $1-2p \in]-1,1[$ et $\lim\limits_{n\to+\infty}(1-2p)^n = 0$ et finalement, $\lim\limits_{n\to+\infty} p_n = \frac{1}{2}$.

---

*Planche*11.*CCINP*

1. • Conception de la procédure $monProb(a,b)$ qui renvoie le produit scalaire de a par b.

```
>>> def monProd(a,b) :
        resul = , 0
        for i in range(0,3) :
            resul = resul + a[i]*b[i]
        return resul
```

Puis on fait un essai en comparant avec *np.vdot*

```
>>> import numpy as np; u = [1,2,-1]; v = [1,1,1]
>>> monProd(u,v), np.vdot(u,v)
(2, 2)
```

2. Attention, si l'on tape :

```
>>> C1 = 1/9*[8,-4,1]; C2 = 1/9*[1,4,8]; C3 = 1/9*[-4,-7,4]
```

Python ne sera pas content car on ne peut pas multiplier dans une liste. On préfere taper : `>>> C1 = 1/9*np.array([8,-4,1]); C2 = 1/9*np.array([1,4,8]); C3 = 1/9*np.array([-4,-7,4])`

```
(array([0.88888889, -0.44444444, 0.11111111]), (array([0.11111111, 0.44444444, 0.88888889]), array([-0.44444444, -0.77
>>> monProb(C1,C2), monProb(C1,C3), monProb(C2,C3)
(0.0, -5.551115123125783e-17, 5.551115123125783e-17)
>>> monProb(C1,C1), monProb(C2,C2), monProb(C3,C3)
(1.0, 1.0, 0.9999999999999998)
```

Les colonnes forment une base orthonormale de ^{3}.

3. On commence par rentrer M.

```
>>> import numpy as np; M = (1/9)*np.array([[8,1,-4],[-4,4,-7],[1,8,4]])
>>> np.dot(np.transpose(M),M)
array([[1.00000000e+00, 8.39520497e-18, -4.43746548e-17],
[8.39520497e-18, 1.00000000e+00, 3.35808199e-17],
[-4.43746548e-17, 3.35808199e-17, 1.00000000e+00]])
```

On reconnait la matrice identité car pour Python $8.39520497e-18$ ou $-4.43746548e-17$, c'est 0.

4. Il suffit de vérifier que le déterminant de M vaut 1.

```
>>> import numpy.linalg as alg; alg.det(M)
0.9999999999999999
```

No comment !

---

*Planche*12.*CCS*2019.*PSI*

1.

```
>>> def A(n) :
        A = np.zeros((n,n))
        for i in range(n) :
            for j in range(n) :
                A[i,j] = max(i,j)
        return A
>>> A(3), A(4), A(5)

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

2.

```
>>> import numpy as np; import numpy.linalg as alg
>>> alg.inv(A(3)), alg.inv(A(4)), alg.inv(A(5))

(array([[-1., 1., 0.], [1., -2., 1.], [0., 1., -0.5]]),
array([[-1.00000000e+00, 1.00000000e+00, 1.48029737e-16, 2.22044605e-16], [1.00000000e+
00, -2.00000000e+00, 1.00000000e+00, -4.44089210e-16], [1.66533454e-16, 1.00000000e+
00, -2.00000000e+00, 1.00000000e+00], [-1.11022302e-16, 1.11022302e-16, 1.00000000e+
00, -6.66666667e-01]]),
array([[-1., 1., 0., 0., 0.], [1., -2., 1., 0., 0.], [0., 1., -2., 1., 0.], [0., 0., 1., -2., 1.], [0., 0., 0., 1., -0.75]]))
```

Cela donne plus clairement :

$$(A(3))^{-1} = \begin{pmatrix} -1 & 1 & 0 \\ 1 & -2 & 1 \\ 0 & 1 & -1/2 \end{pmatrix},\ (A(4))^{-1} = \begin{pmatrix} -1 & 1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 1 & -2/3 \end{pmatrix},\ (A(5))^{-1} = \begin{pmatrix} -1 & 1 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 1 & -3/4 \end{pmatrix}$$

On conjecture que $(A(n))^{-1}$ est une matrice carrée d'ordre n de diagonale $\left(-1,\ -2,\ -2,\ ...,\ -2,\ -\dfrac{n-2}{n-1}\right)$, la première sur-diagonale et la première sous-diagonale ne sont composées que de 1 et les autres coefficients sont 0. Si l'on note $(a'_{i,j})$ les coefficients de $(A(n))^{-1}$, alors

$$a'_{1,1} = -1, \text{ pour } i \in [\![2,\ n-1]\!],\ a'_{i,i} = -2 \text{ et } a'_{n,n} = -\frac{n-2}{n-1}$$
$$\text{Pour } i \in [\![1,\ n-1]\!],\ a'_{i+1,i)=a'_{i,i+1}=1} \text{ et tous les autres coefficients } a'_{i,j} = 0$$

3. On peut déjà remarquer que A étant symétrique réelle est diagonalisable dans $_n()$.

$>>>$ $alg.eig(A(3))\,,\ alg.eig(A(4))\,,\ alg.eig(A(5))$

$((array([4.61185871, -1.27053402, -0.34132469])\,,\ array([[-0.43184431, -0.7916823, 0.43214537], [-0.52548211, -0.1685$

$(array([8.99393442, -2.18009438, -0.51909045, -0.29474959]), array([[-0.37688428, -0.71517028, -0.51860583, -0.2784$

$(array([14.76516338, -3.34546096, -0.75606707, -0.38658106, -0.27705429]), array([[-0.33905186, -0.65185149, 0.52612$

On remarque que toutes les valeurs propres sont simples et la dimension de ses sous-espaces propres pour $n \in \{3, 4, 5\}$ est donc 1 puis on conjecture que la dimension des sous-espaces propres dans le cas général est toujours 1.

$Planche13.CCS2019.PSI$

Soient $h > 0$, suffisamment petit, et la suite (t_i) définie pour tout $i \in$ par $t_i = ih$. L'équation différentielle linéaire (E) : $y'' + \omega^2 y = 0$ régit le comportement d'un oscillateur harmonique.
Montrer que l'ensemble des solutions de (E) est en bijection avec l'ensemble des solutions de (E') :

$$X' = \begin{pmatrix} 0 & 1 \\ -\omega^2 & 0 \end{pmatrix} X.$$

Justifier que les suites (y_i) et (z_i) définies par :

$$\begin{cases} y_{i+1} = & y_i + hz_i \\ z_{i+1} = & z_i - h\omega^2 y_i \end{cases}$$

fournissent une approximation des solutions de (E') puis de (E).
Écrire une fonction $Euler$ qui affiche la solution obtenue par la méthode d'Euler de (E). On tracera aussi la solution exacte de (E). Que remarque t-on après plusieurs périodes ?

*Planche*14.*ENSAM*2018.*PSI*

On donne $a_0 = u$, $b_0 = v$, $a_{n+1} = \sqrt{a_n b_n}$ et $b_{n+1} = \dfrac{1}{\frac{1}{a_n} + \frac{1}{b_n}}$.

Définir une fonction *iterer*(p) qui renvoie $[a_{n+1}, b_{n+1}]$ à partir de $p = [a_n, b_n]$. Tester cette fonction pour $p = [3, 2]$.

Définir une fonction *suite* (non récursive) qui prend (a_0, b_0, n) en argument et qui renvoie a_n et b_n. La tester pour $(3, 2, 3)$ et pour $(3, 2, 5)$. Que peut-on dire ?

On admet que (a_n) et (b_n) sont adjacentes, convergentes vers L tel que pour tout $n \in$,

$$a_n L b_n.$$

Créer une fonction *moyenne*(a_0, b_0) qui renvoie L à 10^{-10} près. Quel est le n correspondant ? Tester cette fonction pour $(3, 2)$.

Tracer *moyenne*$(1, x)$ pour x allant de 1 à 10 avec un pas de 0.1.

Écrire une fonction récursive *Recsuite* qui renvoie a_n et b_n. Comparer la vitesse d'execution de *Recsuite*$(3, 2, 10)$ et de *suite*$(3, 2, 10)$.

---

*Planche*15.*ENSAM*2018.*PSI*

1. On tape :

$>>>$ *import numpy.random as rd*
$>>>$ $LR = rd.random(6)\,;\ LR < 0.5$

$array([False, False, True, True, True, False])$

$>>>$ $1 * (LR < 0.5)$

$array([0, 0, 1, 1, 1, 0])$

2. Définissons la fonction *tirer*(n) réalisant n expériences de Bernoulli de paramètre $1/2$.

3. Écrivons une fonction *temps*(Seq) qui, à partir d'une séquence de 0 et de 1, renvoie le nombre de tirages nécessaires pour que la séquence Seq apparaisse. Par exemple, si $Seq = [1, 0, 1, 1]$, et si les tirages sont $[0, 1, 0, 0, 1, 0, 1, 0, 1, 1]$ alors la fonction renvoie 10.