Corrigés BEOS UPS

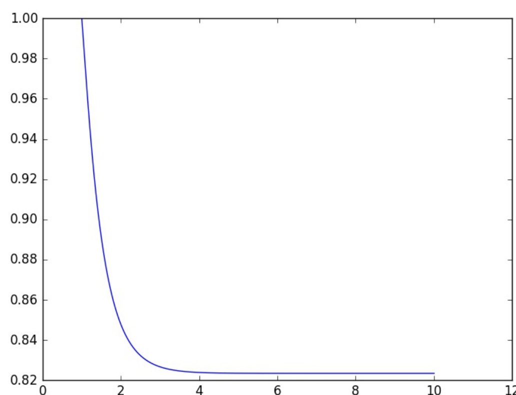
1872.

1. a) Équation différentielle linéaire d'ordre 1. Étude sur $\mathbb{R}_+^*$ et $\mathbb{R}_-^*$ (et éventuels recollements en 0).

b) (EH) : $y' = -\dfrac{2\operatorname{ch} x}{(\operatorname{sh} x)^3} y$, d'où SGH : $y = \lambda \exp\left(\dfrac{1}{(\operatorname{sh} x)^2}\right)$.

2. Méthode d'Euler : Le principe est, pour trouver une valeur approchée d'une solution de $y' = f(x, y)$, de construire à partir des conditions initiales une liste d'abscisses x et d'images $y(x)$, avec $y(x+h) \simeq y(x) + y'(x)h = y(x) + f(x, y)h$.

```
from math import cosh, sinh
def f(x,y):
    return -2*cosh(x)/(sinh(x)**3)*y+(cosh(x))**3/(sinh(x)**5)

h=0.01
a=1.0
b=10.0
x=a
y=1.
Lx=[x]
Ly=[y]
while x<b:
    y=y+f(x,y)*h
    x=x+h
    Lx.append(x)
    Ly.append(y)

import matplotlib.pyplot as plt

plt.plot(Lx,Ly)
plt.show()
```

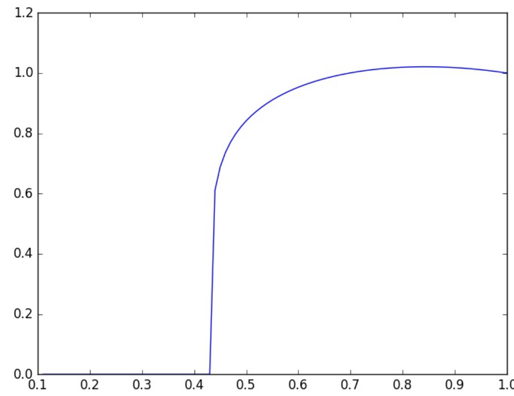


On conjecture que la solution tend vers $+\infty$ en 0.

b.

On note que odeint demande que la condition initiale apparaisse en tête de la liste, mais pas que celle-ci soit rangée dans l'ordre croissant ! On va donc utiliser un pas négatif.

```
from scipy.integrate import odeint
import numpy as np

X=np.arange(1.00,0.1,-0.01)
Y=odeint(f,1.,X)
plt.plot(X,Y)
plt.show()
```



En fait, la solution change de monotonie vers 0.8. et semble tendre vers 0 en 0 ?

3.a. $t \mapsto (t-2)e^t$.

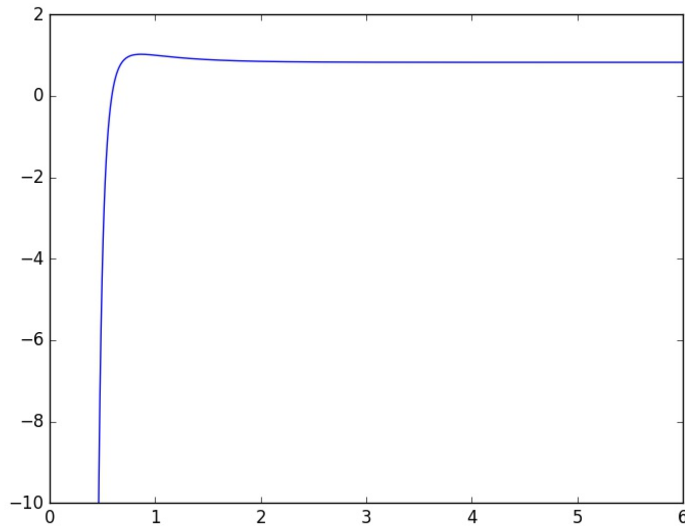
b. La variation de la constante mène au calcul d'une primitive de $\dfrac{(\operatorname{ch} x)^3}{(\operatorname{sh} x)^5}\exp(1/(\operatorname{sh} x)^2)$. Par le changement de variable $t = -\dfrac{1}{(\operatorname{sh} x)^2}$, on est ramené à une primitive de $(t-1)e^t$, et on obtient la solution générale :

$$y = \frac{1}{2}\left(\frac{1}{(\operatorname{sh} x)^2} + 2\right) + C\exp\left(\frac{1}{(\operatorname{sh} x)^2}\right).$$

La condition $y(1) = 1$ fournit : $C = -\alpha/2)e^{-\alpha}$, avec $\alpha = 1/(\operatorname{sh} 1)^2$, d'où la solution :

$$x \mapsto \left(1 + \frac{1}{2\operatorname{sh}^2 x}\right) - \frac{1}{2\operatorname{sh}^2(1)}\exp\left(\frac{1}{\operatorname{sh}^2 x} - \frac{1}{\operatorname{sh}^2 1}\right).$$

```
from math import exp

alpha=1/sinh(1)**2
C=-alpha/2*exp(-alpha)

def y0(x):
    return .5*(1/sinh(x)**2+2)+C*exp(1/sinh(x)**2)

t=np.linspace(0.1,10,1000)
y=[y0(x) for x in t]
plt.plot(t,y)

plt.axis([0,6,-10,2])
plt.show()
```



3.c. Formulation bizarre. La question est : existe-t-il une solution définie sur $\mathbb{R}$ (les solutions sont toujours dérivables !) La réponse est NON, car si on fait $x = 0$ dans l'équation, on obtient $0 = 1$, ce qui est absurde.

1940 PSI.

1. OK

2. Th. spectral : $M^T M = PD^2P^T = S^2$ avec $S = PDP^T$. On verifie que MS^{-1} est orthogonale.

3. On calcule les éléments propres avec numpy.linalg.eig. On applique np.real car les résultats bruts renvoyés par eig sont sous forme complexe.

```
import numpy as np
from numpy import sqrt

m=np.array([[1.,0,0],[0,1,-sqrt(2)],[0,2,0]])

import numpy.linalg as alg
from numpy.linalg import eig

tmm=m.dot(m.transpose())

p=np.real(eig(tmm)[1])
d=np.diag(sqrt(np.real(eig(tmm)[0])))

s=p.dot(d).dot(p.transpose())
omega=m.dot(alg.inv(s))

>>> s
array([[ 1.        ,  0.        ,  0.        ],
       [ 0.        ,  1.63828133,  0.56216928],
       [ 0.        ,  0.56216928,  1.91936596]])

>>> omega
array([[ 1.        ,  0.        ,  0.        ],
       [ 0.        ,  0.95968298, -1.01789752],
       [ 0.        ,  1.35719669, -0.39751371]])
```

4. Il s'agit de l'expression matricielle du procédé de Schmidt. On considère que M est la matrice d'une base B dans la base canonique (orthonormale). Le procédé de Schmidt construit une base orthonormale B' dans laquelle la matrice de B est triangulaire supérieure, à coefficients diagonaux > 0 (matrice T). En notant Ω' la matrice de passage de la base canonique à la base B', il vient (formule de changement de base pour une famille de vecteurs) : $M = \Omega' T$.

5. On applique le procédé de Schmidt, on obtient alors Ω'. On a alors $T = M^t\Omega'$.
NB : on orthonormalise les lignes de ${}^tM$ (donc de M, qui est symétrique).

```
m=np.ones((4,4))+np.eye(4)
omega=m.copy() # indispensable pour une copie indépendante!

# on retire les projections sur les premières directions
for j in range(3):
    for k in range(j+1,4):
        omega[k]=omega[k]-omega[k].dot(omega[j])*omega[j]/omega[j].dot(omega[j])
# on norme les lignes
for j in range(4):
    omega[j]=omega[j]/omega[j].dot(omega[j])
# on transpose pour avoir les vecteurs en colonnes
omega=omega.transpose()
t=omega.transpose().dot(m)

>>> omega
array([[ 0.28571429, -0.38461538, -0.26315789, -0.2        ],
       [ 0.14285714,  0.61538462, -0.26315789, -0.2        ],
       [ 0.14285714,  0.07692308,  0.73684211, -0.2        ],
       [ 0.14285714,  0.07692308,  0.05263158,  0.8        ]])

>>> np.round(t,2) # pour rendre les coefficients plus lisibles.
array([[ 1.  ,  0.86,  0.86,  0.86],
       [ 0.  ,  1.  ,  0.46,  0.46],
       [ 0.  ,  0.  ,  1.  ,  0.32],
       [ 0.  ,  0.  ,  0.  ,  1.  ]])
```

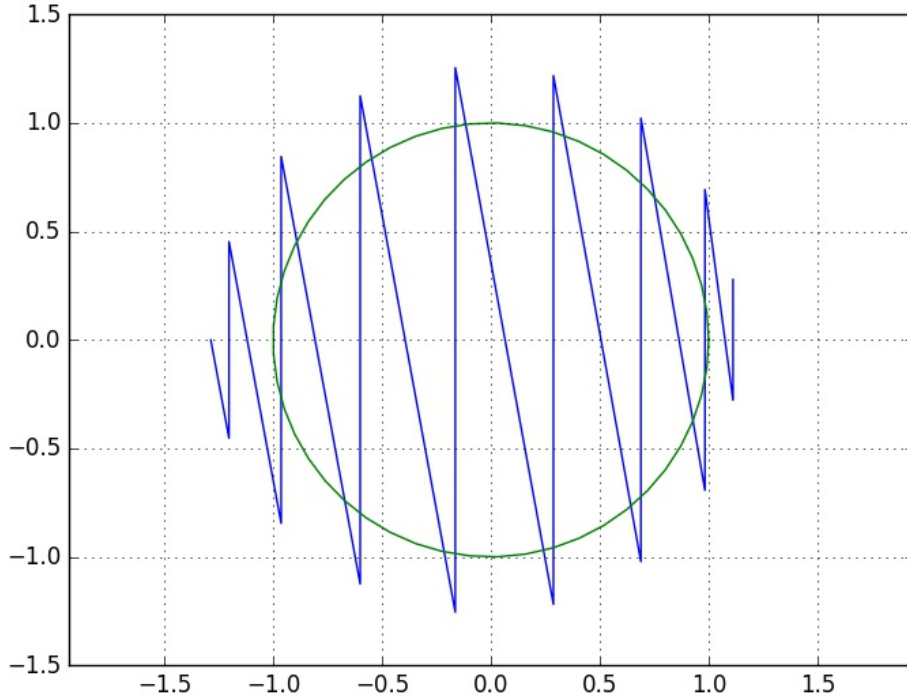
1860 MP

1.a.

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

def racinesR(n,p):
    R=Polynomial(np.ones(n+1))+p-1
    listeracines=R.roots()
    partiesreelles=np.real(listeracines)
    partiesimag=np.imag(listeracines)
    return partiesreelles,partiesimag
```

1.b.

```
import matplotlib.pyplot as plt

def grapheR(n,p):
    #racines de Rnp
    x,y= racinesR(n,p)
    plt.plot(x,y)
    #cercle unité
    t=np.linspace(0,2*np.pi)
    plt.plot(np.cos(t),np.sin(t))
    plt.axis('equal') # repère ON
    plt.grid() # quadrillage
    plt.show()

>>> grapheR(17,41)
```



À ce stade, la conjecture n'est pas évidente (on voit mieux après le 2.b, qu'il faut conjecturer que toutes les racines sont de module > 1).

2.a. On réduit : $F_{n,p}(X) = -pX^n + (p-1)X^n + 1$, donc pour z racine, $p|z|^n \leq (p-1)|z|^{n-1} + 1$. On étudie $Q(r) = pr^n - (p-1)r^{n-1} - 1$ pour $r \geq 0$. $Q(0) = -1$, $Q(1) = 0$, les variations de Q montrent que $Q(r) \leq 0$ ssi $r \in [0, 1]$.

b. De 2.a, on déduit que $Q_{n,p}$ a ses racines de module ≤ 1.

Supposons $|z| = 1$ et z racine. Alors $p = p|z|^n = |1 + z + \cdots + z^{n-1}| \leq |1| + |z| + \cdots + |z|^{n-1} = p$. D'après le cas d'égalité de l'inégalité triangulaire, 1, z, etc. ont même argument, donc $z \in \mathbb{R}_+$. On a alors $z = 1$, or $Q_{n,p}(1) = p + n \neq 0$, contradiction.

Finalement, toutes les racines de $Q_{n,p}$ sont de module < 1.

c. Comme $Q_{n,p}(X) = X^n R_{n,p}(1/X)$, on en déduit que toutes les racines de $R_{n,p}$ sont de module > 1 (c'était donc ça ? ! !)

Supposons que $R_{n,p}$ soit le produit de deux polynômes non constants de $\mathbb{Z}[X]$. Comme $R_{n,p}$ est unitaire et p est premier, l'un des facteurs est de la forme $F = X^m + \cdots + 1$ et le second : $G = X^q + \cdots + p$. Les racines de F sont des racines de $R_{n,p}$, donc de module > 1. Or leur produit vaut $(-1)^m$, ce qui est une contradiction. Conclusion : $R_{n,p}$ est irréductible dans $\mathbb{Z}[X]$.

1802 PC.

1. $a_n + b_n = 1$

2. a. FPT : $P(Z_n = 1) = P(Z_n = 1 | Z_{n-1} = 1)P(Z_{n-1} = 1) + P(Z_n = 1 | Z_{n-1} = -1)P(Z_{n-1} = -1)$, et comme Z_n et Z_{n-1} sont indépendantes : $b_n = (1-p)b_{n-1} + pa_{n-1}$. On en déduit :

$$\begin{pmatrix} a_n \\ b_n \end{pmatrix} = \begin{pmatrix} 1-p & p \\ p & 1-p \end{pmatrix} \begin{pmatrix} a_{n-1} \\ b_{n-1} \end{pmatrix}.$$

b.

```
p=0.2
import numpy as np

def loi(n):
    probas=np.array([p,1-p])
   m=np.array([[1-p,p],[p,1-p]])
    for i in range(1,n):
        probas=m.dot(probas)
    return probas
```

```
>>> loi(3)
array([ 0.392,   0.608])
```

c. On diagonalise la matrice, il vient :

$$\begin{pmatrix} a_n \\ b_n \end{pmatrix} = \frac{1}{2}\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & (1-2p)^{n-1} \end{pmatrix}\begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}\begin{pmatrix} p \\ 1-p \end{pmatrix} = \frac{1}{2}\begin{pmatrix} 1-(1-2p)^n \\ 1+(1-2p)^n \end{pmatrix}.$$

```
>>> for n in range(1,11):
        print(loi(n),(1-(1-2*p)**n)/2)
[ 0.2   0.8] 0.2
[ 0.32   0.68] 0.32
[ 0.392   0.608] 0.392
[ 0.4352   0.5648] 0.43520000000000003
[ 0.46112   0.53888] 0.46112000000000003
[ 0.476672   0.523328] 0.476672
[ 0.4860032   0.5139968] 0.4860032
[ 0.49160192   0.50839808] 0.49160192
[ 0.49496115   0.50503885] 0.494961152
[ 0.49697669   0.50302331] 0.4969766912
```

2.d. $E(X_1) = 1-2p$ et comme les X_i sont indépendantes, $E(Z_n) = (1-2p)^n$.

3. a. $Z_n\dot{Z}_{n+1} = X_{n+1}$, donc $E(Z_nZ_{n+1}) = 1-2p$. On en déduit : $\mathrm{Cov}(Z_n, Z_{n+1}) = (1-2p)-(1-2p)^{2n+1}$.

b. $\mathrm{Cov}(Z_1, Z_2) = (1-2p)(1-(1-2p)^2) = 4(1-2p)p(1-p)$. Les variables sont corrélées, posivement si $p < 1/2$, négativement si $p > 1/2$. Si $p = 1/2$, les variables sont décorrélées, et elles sont même indépendantes !

En effet $P(Z_1 = a, Z_2 = b) = P(X_1 = a, X_2 = b/a) = P(X_1 = a)P(X_2 = b/a) = P(X_1 = a)P(Z_2 = b)$ car $P(Z_2 = b) = P(X_2 = b/a) = 1/2$.

1797.

1. Préférer une version itérative (complexité linéaire), car une version récursive sans "mémoïsation" a une complexité exponentielle !

```
def F(n):
    if n==0:
        return 0
    u,v=0,1
    for k in range(n):
        u,v=v,u+v
    return u
>>> F(50)
12586269025
```

2.

```
def RF(n,m):
    return F(n)%m
>>> RF(100,5), RF(100,7), RF(100, 43)
(0, 3, 15)
```

b. On a :

$$\begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} F_{n-1} \\ F_n \end{pmatrix}$$

donc

$$\begin{pmatrix} F_{n-1} \\ F_n \end{pmatrix} = \begin{pmatrix} -1 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix}$$

On en déduit l'équivalence proposée.

3. Les couples $c_n = (F_n(m), F_{n+1}(m))$ ne prennent qu'au plus m^2 valeurs distinctes, donc il existe $a \in \mathbb{N}$ et $p \in \mathbb{N}^*$ tel que $c_a = c_{a+p}$, ce qui équivaut d'après 2.b. à $c_0 = c_p$, puis à $c_n = c_{n+p}$ pour tout $n \in \mathbb{N}$, donc en particulier $(F_n(m))$ est de période p.

On teste si $c_p = c_0$ à partir de $p = 1$, ce qui permet d'obtenir la (plus petite) période de $(F_n(m))$:

```
def periode(m):
    p=1
    while RF(p,m),RF(p+1,m) != 0,1:
        p+=1
    return p

>>> for j in range(2,7):
...     periode(j)
3
8 # période de F_n(3)
6
20
24
```

1764. A. 1.

```
import numpy as np

def J(n):
    j=np.zeros((n,n))
    for i in range(n):
        j[i,n-1-i]=1
    return j

>>> J(4)
array([[ 0.,  0.,  0.,  1.],
       [ 0.,  0.,  1.,  0.],
       [ 0.,  1.,  0.,  0.],
       [ 1.,  0.,  0.,  0.]])
```

2.

```
from numpy.random import randint

def Randomat(n):
    m=np.zeros((n,n))
    for i in range(n):
        for j in range(n):
            m[i,j]=randint(0,100)
    return m

>>> a=Randomat(4)
array([[ 76.,  63.,  98.,  24.],
       [ 63.,  69.,  56.,   7.],
       [ 99.,  67.,  36.,  85.],
       [ 36.,  19.,  84.,  31.]])
```

3.

```
def centroMatrix(A):
    n=A.shape[0]
    m=np.zeros((n,n))
    for i in range(n):
        for j in range(n):
```

```
            m[i,j]=a[n-1-i,n-1-j]
    return m
>>> centroMatrix(a)
array([[ 31.,  84.,  19.,  36.],
       [ 85.,  36.,  67.,  99.],
       [  7.,  56.,  69.,  63.],
       [ 24.,  98.,  63.,  76.]])
```

4. Si on considère que A est la matrice de l'endo f dans la base $(e_1, \cdots, e_n)$, alors $\hat(A)$ est la matrice de f dans la base $(e_n, \cdots, e_1)$. Comme $J(n)$ est la matrice de passage et $J(n)^{-1} = J(n)$, on en déduit :

$$\hat{A} = J_n A J_n.$$

B. 1. Évident (car $J_n J_n A J_n J_n = A$).

2. Notons $J = J_n$. $J(AB)J = JAJJBJ$ donc $\widehat{AB} = \hat{A}\hat{B}$.

Si A est inversible, $(JAJ)^{-1} = JA^{-1}J = \hat{A^{-1}}$.

3. J est symétrique, donc ${}^t(JAJ) = J^t AJ = {}^t\hat{A}$.

4. A et $\hat{A}$ sont semblables donc ont même déterminant.

C. 1. Notons $\varphi : A \mapsto \hat{A}$. Comme φ est une symétrie d'après B.1, $M_n(\mathbb{R}) = \mathrm{Ker}(\varphi - \mathrm{Id}) \oplus \mathrm{Ker}(\varphi + \mathrm{Id})$, cqfd.

2. Notons $T : A \mapsto {}^tA$. D'après B.3, T et φ commutent, donc les espaces propres de φ sont stables par T. Comme T est diagonalisable et a pour sous-espaces propres S_n et A_n, on en déduit le résultat.

1743 PSI (Le problème de l'ascenseur)

1.

```
def simul(personnes,etages):
    demandes=[]# liste des arrêts demandés (numérotés àpd 0)
    for i in range(personnes):
        demandes.append(randint(etages))
    arrets=[0]*etages
    for x in demandes:
        arrets[x]=1 # on coche les étages demandés
    return sum(arrets)
```

Méthode erronée : on pourrait penser à procéder, à chaque étage, à tirer au hasard le nombre de passagers qui descendent. Dans ce cas, la probabilité que personne ne descende à l'étage 1 est $1/(n+1)$ et les arrêts ne seraient pas équiprobables !

2. a. X_i suit une loi de Bernoulli $B(p_i)$. Il y a p étages, donc la probabilité qu'une personne donnée s'arrête à i est $1/p$, donc $P(X_i = 0) = (1 - 1/p)^n$, donc paramètre $p_i = 1 - (1 - 1/p)^n$ (ne dépend pas de i, ce qui était prévisible).

b. $X = \sum_{i=1}^{p} X_i$.

c. Linéarité de l'espérance : $E[X] = pE[X_1] = p\,(1 - (1 - 1/p)^n)$.

d.

```
for etages in range(3,21):
    s=0
    personnes=10
    for j in range(1000):
        s+=simul(personnes,etages)
    print(s/1000,etages*(1-(1-1/etages)**personnes))
>>> (executing)
2.939 2.947975410252502
3.764 3.7747459411621094
4.429 4.463129088
5.002 5.030966502660926
5.521 5.501591790790845
```

```
5.868  5.895395390689373
6.246  6.228484671083053
6.543  6.513215599
6.652  6.7590238162751515
6.898  6.973153345384887
7.16  7.161217607155774
7.382  7.3276133929594405
7.448  7.475822621159781
7.625  7.608632399220369
7.682  7.7282965151685215
7.888  7.83665500764351
7.931  7.935223588631142
8.052  8.025261215232426
```

La simulation confirme donc les calculs...

1711 MP

Le sujet est mal écrit (et hors programme en PC/PSI). Il faut lire : on définit un sous-ensemble A de $I_N = [\![0, N]\!]$ par une liste L de booléens telle que L[i]=True ssi $i \in A$.

1.a. Écrire une fonction Python qui prend en argument N, et les listes de booléens L_1 et L_2 représentant les sous-ensembles X_1 et X_2 de I_N et qui renvoie une liste L des éléments de I_N sommes d'un élément de X_1 et d'un élément de X_2.

```
def sommes(N,L1,L2):
    L=[False]*(2*N+2)
    for i in range(N+1):
        for j in range(N+1):
            if L1[i] and L2[j]:
                L[i+j]=True
    return L[:N+1]
```

1.b. Je comprends qu'il faut déterminer l'ensemble des sommes de 2,3 puis 4 carrés d'entiers pour des nombres de 0 à 100.

```
Carres=[False]*101
for i in range(11):
    Carres[i**2]=True
C2=sommes(100,Carres,Carres)
C3=sommes(100,Carres,C2)
C4=sommes(100,Carres,C3)

X1=[i for i in range(101) if Carres[i]]
X2=[i for i in range(101) if C2[i]]
X3=[i for i in range(101) if C3[i]]
X4=[i for i in range(101) if C4[i]]
```

```
>>> len(X2),len(X3),len(X4)
44,86,101
```

On constate que tous les nombres de 0 à 100 sont sommes de 4 carrés.

2. L'ensemble des sommes de 4 carrés d'entier est stable par multiplication.

3. La propriété signifie que si $b = 2m + 1$, a est congru à un élément de $[-m, m]$ (dit comme ça, c'est plus simple!)

Soient x, y entre 0 et $(p-1)/2$ tels que $1 + x^2 \equiv 1 + y^2$. On en déduit $(x - y)(x + y) \equiv 0$ (modulo p), donc $x \equiv y$ ou $x \equiv -y$ (car p est premier). Si $x + y \equiv 0$, alors $x = y = 0$, car $0 \le x + y < p$. On a donc toujours $x \equiv y$, donc $x = y$, ce qui montre que les classes des éléments de la forme $1 + x^2$ sont distinctes.

4. On obtient donc $(p + 1)/2$ classes distinctes. On procède de même pour montrer que les $-y^2$, $y \in \{0, \dots, (p-1)/2\}$ forment $(p + 1)/2$ classes distinctes. Or $\mathbb{Z}/p\mathbb{Z}$ ne comporte que p classes, d'où le résultat.

b. L'objectif est ici de montrer qu'un nombre premier est la somme de 4 carrés. Comme tout entier naturel non nul est le produit de nombres premiers, on aura démontré que tout entier est la somme de 4 carrés. Pour $2 = 1+1+0+0$, c'est clair. Reste à démontrer le résultat pour les premiers impairs.

D'après a., on a $m \in \mathbb{N}$ tel que $1 + x^2 + y^2 + 0^2 = mp$. On vérifie facilement $mp \leq 1 + (p-1)^2/2$ donc $m \leq (p - 2 + 3/p)/2 \leq (p-1)/2$.

Soit m minimal tel que $mp = x_1^2 + \cdots + x_4^2$. On suppose $m > 1$.

Si m est pair alors, quitte à réindexer, on se ramène à l'une des situations suivantes :

– les x_i sont tous pairs
– x_1, x_2 pairs ; x_3, x_4 impairs
– les x_i sont tous impairs

Dans tous les cas, $(x_1 + x_2)$, $(x_1 - x_2)$, $(x_3 + x_4)$, $(x_3 - x_4)$ sont pairs et :

$$\frac{m}{2}p = (x_1 + x_2)^2/4 + (x_1 - x_2)^2/4 + (x_3 + x_4)^2/4 + (x_3 - x_4)^2/4$$

et m n'est pas minimal (car $m/2 < m$).

On en déduit m impair, donc pour tout i il existe y_i entre 0 et $(m-1)/2$ tel que $y_i \equiv x_i$ modulo m. La somme $s = \sum_i y_i^2$ est, comme $\sum_i x_i^2$, multiple de m et $0 \leq s < 4m^2/4 = m^2$. On note $s = mr$ avec $0 \leq r \leq m-1$.

On peut alors écrire $mrmp = \sum_i z_i^2$ (d'après 2.), avec par ex. $z_1 = x_1y_1 + \cdots + x_4y_4$ divisible par m (formules données au 2.) de même pour z_2, z_3, z_4, donc en posant $z_i = mt_i$, il vient :

$$rp = t_1^2 + \cdots + t_4^2$$

ce qui est contradictoire car $r < m$. On en déduit $m = 1$. (preuve de Hardy et Wright p.389, attention la preuve de Wikipédia est un peu fausse, il me semble).

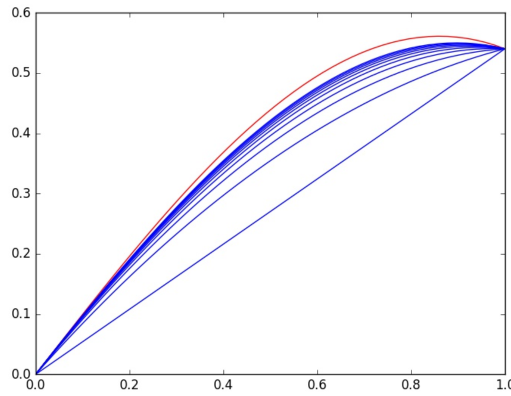
1215 MP. Énoncé : la somme qui définit B_n est de 0 à n. Il manque 4. Montrer que B_n converge uniformément vers f sur $[0,1]$.

1. Linéarité de E : $E[M_n(x)] = x$. Indépendance des X_n : $V[M_n(x)] = \frac{1}{n}x(1-x) \leq \frac{1}{4n}$ (majoration classique de $x(1-x)$ sur $[0,1]$).

2. Transfert : $E[f(M_n(x))] = \sum_{k=0}^{n} f(k/n)\binom{n}{k}x^k(1-x)^{n-k} = B_n(x)$.

3.

```
from scipy.misc import comb
import numpy as np
import matplotlib.pyplot as plt

def tracer(f,n):
    t=np.linspace(0,1)
    ft=np.array([f(ti) for ti in t])
    plt.plot(t,ft,'r')
    for m in range(1,n+1):
        b=np.array([sum(comb(m,k)*f(k/m)*ti**k*(1-ti)**(m-k) \
        for k in range(m+1)) for ti in t])
        plt.plot(t,b,'b')
    plt.legend()

>>> f=lambda t:t*np.cos(t)
>>> tracer(f,10)
```



4. Soit $\varepsilon > 0$. f est continue sur $[0, 1]$ donc uniformément continue (th. de Heine), donc on fixe $\alpha > 0$ tel que $|a - b| \leq \alpha \Rightarrow |f(a) - f(b)| \leq \varepsilon$ (en PC/PSI, on peut se contenter du cas particulier où f est lipschitzienne, voire C^1 sur $[0, 1]$).

On observe :

$$|f(x) - B_n(x)| \leq \sum_{k=0}^{n} \binom{n}{k} |f(x) - f(k/n)| x^k (1-x)^{n-k} = E[|f(x) - f(M_n)|].$$

On vérifie facilement :

$$|f(x) - f(M_n)| \leq \varepsilon \mathbb{1}_{|x-M_n| \leq \alpha} + 2||f||_\infty \mathbb{1}_{|x-M_n|>\alpha},$$

et en passant à l'espérance :

$$E[|f(x) - f(M_n)|] \leq \varepsilon + 2||f||_\infty P(|x - M_n| > \alpha) \leq \varepsilon + 2||f||_\infty \frac{1}{4n\alpha^2}$$

(avec Bienaymé-Tchebychev et la majoration de 1.) Pour n assez grand, on a donc la majoration : $E[|f(x) - f(M_n)|] \leq 2\varepsilon$, indépendamment de x, ce qui justifie la convergence uniforme de (B_n) vers f sur $[0, 1]$.

1193A PSI (=771 oraux 2015 RMS)

1. Définition de f puis tracé de sa courbe représentative.

```
def f(x):
    a1=m.floor(10*x)
    a2=m.floor(100*x)-a1*10
    y=x-a1/10-a2/100+a2/10+a1/100
    return y

import numpy as np
import matplotlib.pyplot as plt
t=np.linspace(0,1,200)
plt.plot(t,[f(u) for u in t])
plt.axhline()
plt.axvline()
```

2. f n'est pas continue. Par ex, si $x \in [0, .01[$, alors $x = 0.00\ldots$ donc $f(x) = x$ mais $f(0.01) = 0.1$

3. f est continue par morceaux sur $[0, 1]$ car 1. montre que f est combinaison linéaire de fonctions cpm. On note que les points de discontinuités (à gauche) sont les $k/100$, avec k entier et $1 \leq k \leq 100$.

4. On utilise une méthode des rectangles. Elle est lente pour une fonction cpm, mais converge (ceci n'est plus au programme, d'ailleurs).

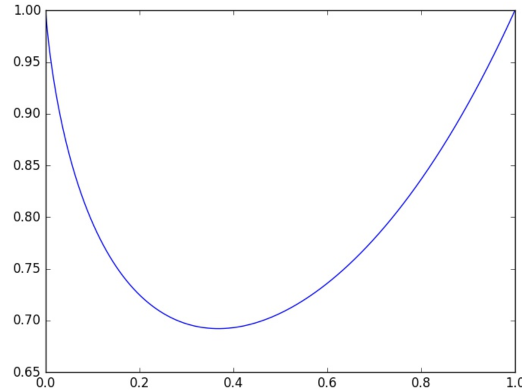
```
def intf(n):
    return 1/n*sum([f(k/n) for k in range(n)])

>>> intf(10000)
0.49992299999999945
```

1193 B PSI.

1. $g(x) = e^{x\ln x}$ donc $g(x) \to 1$ quand $x \to 0$.

2.

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

t=np.linspace(0,1,1000)
plt.plot(t,t**t)
```



3. La dérivée $g'(x) = (1 + \ln(x))x^x$ s'annule en $1/e$, donc le minimum est $g(1/e) = e^{-1/e}$. En 0, $g'(x) \to -\infty$ donc la courbe de g présente une tangente verticale.

4. Méthode des rectangles à droite (pour éviter la singularité en 0).

```
g=lambda t:t**t
def intg(n):
    return 1/n*sum(g(k/n) for k in range(n))

def trapg(n):
    return 1/n*(sum(g(k/n) for k in range(1,n))+g(0.)/2+g(1.)/2)
```

```
>>> intg(1000000)
0.7834305107135209
```

La méthode des rectangles est très lente (en $1/n$), celle des trapèzes un peu moins (en $1/n^2$). La question suivante permet de diminuer l'incertitude sur la valeur de l'intégrale.

5. $e^{x\ln x} = \displaystyle\sum_{n=0}^{+\infty} \frac{x^n(\ln x)^n}{n!}$.

On calcule par récurrence $\displaystyle\int_0^1 x^p(\ln x)^q dx = \frac{(-1)^q q!}{(p+1)^{q+1}}$, et en particulier :

$$\int_0^1 \frac{|x^n(\ln x)^n|}{n!} dx = \frac{1}{(n+1)^{n+1}}.$$

On peut donc appliquer le théorème d'intégration terme à terme, et $\displaystyle\int_0^1 x^x dx = \sum_{n=0}^{+\infty} \frac{(-1)^n}{(n+1)^{n+1}}$.

D'après le TSSA, la valeur absolue du reste d'ordre n est majorée par $\dfrac{1}{(n+2)^{n+2}}$, la convergence est donc très rapide.

```
>>> sum((-1)**k/(k+1)**(k+1) for k in range(14))
0.7834305107121344
```

Comme $15^{15} > 4\dot{\imath}10^{17}$, on excède déjà la précision de calcul.

En revanche, il ne faut pas espérer trouver une valeur *exacte* de cette intégrale.