```
# -*- coding: utf-8 -*-
"""
Created on Sun Feb  8 08:37:02 2015

@author: stephane
"""

import numpy as np
import matplotlib.pyplot as pypl
from scipy.integrate import trapz
from math import factorial, exp, sqrt
from scipy.misc import comb
from scipy.optimize import fsolve
from random import random, randint
from numpy import cos, sin

#
# Exo 0
#


def somcube(n):
    todo = n # ce qui reste
    somme = 0
    while todo > 0:
        somme += (todo % 10)**3
        todo //= 10
    return somme

"""
>>> somcube(1234)
100
"""

res = []
for i in range(1001):
    if i == somcube(i):
        res.append(i)

res2 = [i for i in range(1001) if i == somcube(i)]

"""
>>> res, res2
([0, 1, 153, 370, 371, 407], [0, 1, 153, 370, 371, 407])
"""

"""
>>> list(str(1234))
['1', '2', '3', '4']
"""

def somcube2(n): # je ne pense pas avoir compris l'énoncé
    return sum(int(car)**3 for car in  list(str(n)))

"""
>>> somcube2(1234)
100
"""


#
# Exo 1
#

fichier = open('ex_001.csv', 'w')
for i in range(101):
    fichier.write("%.10f;%.10f\n"%(np.pi*i/100, np.sin(np.pi*i/100)))
fichier.close()

# retenons-nous de faire du oneliner...
```



```
LX, LY = [], []
for ligne in open('ex_001.csv'):
    casse = ligne.rstrip().split(';')
    LX.append(float(casse[0]))
    LY.append(float(casse[1]))

"""
>>> LX[:4], LY[:4]
([0.0, 0.0314159265, 0.0628318531, 0.0942477796], [0.0, 0.0314107591, 0.0627905195, 0.0941083133])
"""

pypl.plot(LX, LY)

def trapezes(LY, LX):
    return sum((LX[i]-LX[i-1])*(LY[i]+LY[i-1])/2 for i in range(1, len(LX)))

"""
>>> trapezes(LY, LX)
1.999835503892275

>>> trapz(LY, LX)
1.9998355038922742
"""



#
# Exo 2
#

M = np.array([[0,9,3,-1,7], [9,0,1,8,-1], [3,1,0,4,2],\
              [-1,8,4,0,-1], [7,-1,2,-1,0]])

v4 = [i for i in range(len(M)) if M[4][i] > 0]

"""
>>> v4
[0, 2]
"""

def degre(i):
    return len([j for j in range(len(M)) if M[i][j] > 0])
"""
>>> [degre(i) for i in range(5)]
[3, 3, 4, 2, 2]
"""

def longueur(L):
    longueurs = [M[L[i], L[i+1]] for i in range(len(L)-1)]
    if -1 in longueurs:
        return -1
    return sum(longueurs)

"""
>>> longueur([1, 3, 2, 4, 0, 2])
24
>>> longueur([1, 3, 2, 4, 0, 3])
-1
"""


#
# Exo 3
#

t1 = [0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0]

def nombreZeros(t, i):
    nb = 0
    j = i
    while j < len(t) and t[j] == 0:
```

```
        nb += 1
        j+= 1
    return nb

def nombreZeroMax(t): # quadratique...
    return max(nombreZeros(t, i) for i in range(len(t)))

def nombreZeroMaxLineaire(t):
    # C'est niais de s'imposer l'utilisation de nombreZeros, mais bon...
    maxi = 0 # le maximum courant
    debut = 0 # on regarde à partir de cet indice
    while debut < len(t):
        courant = nombreZeros(t, debut)
        if courant > maxi:
            maxi = courant
        debut += 1+courant # On passe au suivant
    return maxi

"""
>>> [nombreZeros(t1, x) for x in [4, 1, 8]]
[3, 0, 1]
>>> nombreZeroMax(t1)
4
>>> nombreZeroMaxLineaire(t1)
4
"""

#
# Exo 4
#

def Px(k, n, p):
    return exp(-n*p) * (n*p)**k / factorial(k)


def Py(k, n, p):
    return comb(n, k) * p**k * (1-p)**(n-k)

pypl.clf()
pypl.plot(range(31), [Px(k, 30, 0.1) for k in range(31)])
pypl.plot(range(31), [Py(k, 30, 0.1) for k in range(31)])
pypl.xlabel('$k$')
pypl.ylabel('$P(V=k)$')
pypl.legend(['${\\cal P}(nk)$', '${\\cal B}(n,p)$'])
pypl.savefig('poisson_binomiale.pdf')

"""
>>> [Px(k, 30, 0.1) for k in range(8)]
[0.049787068367863944, 0.14936120510359183, 0.22404180765538775,
0.22404180765538775, 0.16803135574154082, 0.10081881344492448,
0.05040940672246225, 0.0216040314524838 2]
>>> [Py(k, 30, 0.1) for k in range(8)]
[0.042391158275216237, 0.14130386091738778, 0.22765622036689948,
0.23608793223233993, 0.17706594917425619, 0.10230477063401491,
0.047363319737969395, 0.018043169423988526]
"""

def Ecart(n, p):
    return max(abs(Px(k, n, p)-Py(k, n, p)) for k in range(n+1))

def Nseuil(e, p): # on ne va pas finasser : quadratique plutot que n.ln(n)...
    n = 1
    while Ecart(n, p) > e:
        n += 1
    return n

"""
>>> Nseuil(0.008, 0.075), Nseuil(0.005, 0.075)
(1, 124)
```



```
>>> Nseuil(0.008, 0.1), Nseuil(0.005, 0.1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/stephane/documents/info-pour-tous/zero/pt/algo_simu.py", line 197, in Nseuil
    while Ecart(n, p) > e:
  File "/home/stephane/documents/info-pour-tous/zero/pt/algo_simu.py", line 193, in Ecart
    return max(abs(Px(k, n, p)-Py(k, n, p)) for k in range(n+1))
  File "/home/stephane/documents/info-pour-tous/zero/pt/algo_simu.py", line 193, in <genexpr>
    return max(abs(Px(k, n, p)-Py(k, n, p)) for k in range(n+1))
  File "/home/stephane/documents/info-pour-tous/zero/pt/algo_simu.py", line 167, in Px
    return exp(-n*p) * (n*p)**k / factorial(k)
OverflowError: long int too large to convert to float
>>> Nseuil(0.008, 0.1), Nseuil(0.0055, 0.1)
(71, 151)
>>> Nseuil(0.008, 0.1), Nseuil(0.0052, 0.1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/stephane/documents/info-pour-tous/zero/pt/algo_simu.py", line 197, in Nseuil
    while Ecart(n, p) > e:
  File "/home/stephane/documents/info-pour-tous/zero/pt/algo_simu.py", line 193, in Ecart
    return max(abs(Px(k, n, p)-Py(k, n, p)) for k in range(n+1))
  File "/home/stephane/documents/info-pour-tous/zero/pt/algo_simu.py", line 193, in <genexpr>
    return max(abs(Px(k, n, p)-Py(k, n, p)) for k in range(n+1))
  File "/home/stephane/documents/info-pour-tous/zero/pt/algo_simu.py", line 167, in Px
    return exp(-n*p) * (n*p)**k / factorial(k)
OverflowError: long int too large to convert to float
>>> Nseuil(0.008, 0.1), Nseuil(0.0053, 0.1)
(71, 162)

Si p est trop grand, il ne faut pas espérer une trop bonne approximation.

Oui, bon, il faudrait regarder ça de plus près...
"""

#
# Exo 5
#

pypl.clf()

def g(x):
    return min(x,1)
gg = np.vectorize(g)
les_x = np.arange(0, 2, 0.01)
pypl.plot(les_x, gg(les_x), linewidth=2)

def f(x):
    if x < 2:
        return g(x)
    return sqrt(x)*f(x-2)
ff = np.vectorize(f)
les_x = np.linspace(0, 6, 1000)
pypl.plot(les_x, ff(les_x), linewidth=1)

pypl.ylim(-0.1, 5.1)
pypl.xlim(-0.1, 6.1)
pypl.axhline(color='black')
pypl.axvline(color='black')
pypl.legend(['$g$', '$f$'], loc='upper center')
pypl.grid()
pypl.savefig('f_et_g.pdf')

alpha = fsolve(lambda x:f(x)-4, 5)
"""
>>> alpha
array([ 5.12310563])
```

Ce n'est pas une approximation du minimum (qui n'existe pas) mais de la borne
inférieure... et f(borne inf) = 4 !!!

Bon, à la main ?
"""

```
gauche, droite = 5, 5.5
while droite-gauche > 0.01:
    milieu = (gauche+droite)/2
    if f(milieu) < 4:
        gauche = milieu
    else:
        droite = milieu
```

"""
>>> milieu
5.1171875
"""

```
#
# Exo 6
#

def d(n):
    L = [1]
    for nombre in range(2, n+1):
        if n % nombre == 0:
            L.append(nombre)
    return L

"""
```

>>> d(4), d(10)
([1, 2, 4], [1, 2, 5, 10])
Peut-être la liste des nombre de [2,n] divisant n ?
"""

```
def DNT(n):
    if n <= 1:
        return []
    return d(n)[1: -1]

"""
```

>>> [DNT(n) for n in range(16)]
[[], [], [], [], [2], [], [2, 3], [], [2, 4], [3], [2, 5], [], [2, 3, 4, 6], [], [2, 7], [3, 5]]
"""

```
def sommeCarresDNT(n):
    return sum(x**2 for x in DNT(n))

#magiques = [x for x in range(1001) if x == sommeCarresDNT(x)]

"""
```

>>> magiques
[0, 4, 9, 25, 49, 121, 169, 289, 361, 529, 841, 961]

Hum... carrés de nombres premiers ?
"""

```
#
# Exo 7
#

alphabet = "".join([chr(x) for x in range(ord('a'), 1+ord('z'))]) # héhé

def decalage(n):
    assert n <= 26 # j'imagine...
    return alphabet[n:] + alphabet[:n]

"""
```

Je ne sais pas si je reste dans l'esprit de l'énoncé...


```
>>> decalage(3)
'defghijklmnopqrstuvwxyzabc'
>>> decalage(24)
'yzabcdefghijklmnopqrstuvwx'
"""

def indices(x, phrase):
    liste = []
    for i in range(len(phrase)): # on va éviter l'enumerate, mais bon...
        if phrase[i] == x:
            liste.append(i)
    return liste

def codage(n, phrase):
    # Je ne suis pas certain d'être dans l'esprt de l'énoncé...
    return "".join(chr(((ord(phrase[i])-ord('a')+n)%26)+ord('a'))\
                   for i in range(len(phrase)))

"""
>>> codage(10, 'blabla')
'lvklvk'
>>> codage(1, 'blabla')
'cmbcmb'
>>> codage(20, 'blabla')
'vfuvfu'
"""

def decodage(n, phrase):
    return codage(26-n, phrase)

"""
>>> decodage(10, codage(10, 'turlututu'))
'turlututu'
"""

#
# Exo 8
#

M, m = 20, 10

def f(c):
    un = 0
    for k in range(m+1): # oui, vraiment
        if abs(un) > M:
            return k
        un = un**2 + c
    return m+1

pypl.clf()
LX = np.linspace(-2, 2, 401)
LY = np.vectorize(f)(LX)
pypl.xlim(-2.2, 2.2)
pypl.ylim(-1, m+2)
pypl.plot(LX, LY)
pypl.axhline(color='black')
pypl.axvline(color='black')

#pypl.show()

#for m in [10, 20, 50]:
#    pypl.clf()
#    images = np.array([[f(complex(x,y)) for x in np.linspace(-2, 0.5, 1001)]\
#                           for y in np.linspace(-1.1, 1.1, 1001)])
#
#    pypl.imshow(images, extent = [-2, 0.5, -1.1, 1.1])
#    pypl.savefig('mandelbrot%i.pdf'%m)
#    pypl.savefig('mandelbrot%i.jpg'%m)
```

```
# Quelle surprise !

#
# Exo 9
#

R = np.array([[1, 2, 3], [4, 5, 6]])
S = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

"""
>>> print(R)
[[1 2 3]
 [4 5 6]]
>>> print(S)
[[1 2 3]
 [4 5 6]
 [7 8 9]]
"""

def test(M): # sans connaitre shape...
    if len(M) == len(M[0]):
        return len(M)
    return 0

"""
>>> test(R), test(S)
(0, 3)

>>> R.shape
(2, 3)

Et maintenant, un générateur de matrices...
"""

def genere():
    A = np.zeros((5,5))
    for i in range(5):
        A[i, i] = 0.4+0.2*random()
        for j in range(i+1, 5):
            A[i, j] = 0.1*random()
            A[j, i] = A[i, j]
    return A

def ecriture(M, but):
    fichier = open(but, 'w')
    for ligne in M:
        for x in ligne:
            fichier.write(str(x)+" ")
        fichier.write('\n')
    fichier.close()

for i in range(1, 11):
    ecriture(genere(), 'ex_009_%i.txt'%i)

# Retournons à nos moutons, avec un oneliner

M1 = np.array([map(float,ligne.strip().split(' ')) for ligne in open('ex_009_1.txt'
)])

vapvep = np.linalg.eig(M1)
vap = vapvep[0]

"""
>>> vapvep
(array([ 0.73699759,  0.53115466,  0.48103329,  0.36679228,  0.38748681]),
 array([[-0.31631231, -0.44534528,  0.45269259,  0.63451388, -0.30671756],
        [-0.48781898, -0.57075601, -0.03561051, -0.34735115,  0.5606686 ],
        [-0.29923524,  0.19773546,  0.57426107, -0.61526943, -0.40376568],
        [-0.50413014, -0.02476047, -0.67680843, -0.04399143, -0.5340738 ],
```



```
        [-0.56417217,  0.66044891,  0.07717437,  0.31023798,  0.37856771]]))
>>> vap
array([ 0.73699759,  0.53115466,  0.48103329,  0.36679228,  0.38748681])
"""

def dansIntervalle(L, a, b):
    for x in L:
        if x < a or x > b:
            return False
    return True

"""
>>> dansIntervalle(vap, 0, 1)
True
>>> dansIntervalle(vap, 0.5, 1)
False
"""
#
# Exo 10
#

def comptage(L, N):
    resultat = [0] * N
    for x in L:
        resultat[x] += 1
    return resultat

def tri(L, N):
    foo = comptage(L, N)
    resultat = []
    for i in range(N): # un extend pour changer. Ça ira en amorti
        resultat.extend([i] * foo[i])
    return resultat

def liste_alea(nb, maxi):
    return [randint(0, maxi) for _ in range(nb)]
"""
>>> liste_alea(10, 3)
[3, 0, 0, 1, 0, 0, 0, 3, 2, 0]

>>> bar = liste_alea(20, 5)
>>> bar
[1, 3, 3, 2, 4, 2, 4, 3, 4, 4, 4, 3, 1, 2, 2, 3, 1, 4, 5, 1]
>>> tri(bar, 6)
[1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 5]
>>> bar.sort()
>>> bar
[1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 5]

OK

On obtient un tri en O(N*n) vs. n^2 pour l'insertion et n.ln(n) pour la fusion
"""

#
# Exo 11
#

b, w = 0.5, 6

def p(t): # position
    return (cos(t) + b*cos(w*t), sin(t) + b*sin(w*t))
def v(t): # vitesse
    return (-sin(t) - b*w*sin(w*t), cos(t) + b*w*cos(w*t))
def a(t): # accélération
    return (-cos(t) - b*w**2*cos(w*t), -sin(t) - b*w**2*sin(w*t))

"""
>>> p(1), v(1), a(1)
```

```
((1.0203874491933227, 0.7017632357084336),
 (−0.0032244902111189244, 3.4208131658192378),
 (−17.823367465574727, 4.188007982772769))
"""

L = np.linspace(−np.pi, np.pi, 201)
Lx, Ly = [], []
for t in L:
    (x, y) = p(t)
    Lx.append(x)
    Ly.append(y)

pypl.clf()
pypl.plot(Lx, Ly, linewidth=2)

def c(t):
    xx, yy = p(t)
    vx, vy = v(t)
    ax, ay = a(t)
    d = (vx**2+vy**2)/(vx*ay−vy*ax)
    return (xx−d*vy, yy+d*vx)

Lx, Ly = [], []
for t in L:
    (x, y) = c(t)
    Lx.append(x)
    Ly.append(y)
pypl.plot(Lx, Ly, linewidth=2)

pypl.axhline(color='black')
pypl.axvline(color='black')

pypl.savefig('developpee.pdf')

def distance((x1,y1), (x2,y2)):
    return sqrt((x2−x1)**2+(y2−y1)**2)

def longueur(dt):
    lg = 0
    courant = p(0)
    for i in range(int(np.rint(2*np.pi/dt))):
        suivant = p( (i+1) * dt )
        lg += distance(courant, suivant)
        courant = suivant
    return lg

#longueurs = [longueur(2*np.pi/10**k) for k in range(2,7)]

"""
>>> longueurs
[19.26847819516613, 19.37581017276328, 19.37688557820326,
 19.37689633246676, 19.376896440008714]
"""
```