Corrigé des exercices types oraux de la banque PT

Ce document donne les codes de programmes permettant de répondre aux différents exercices de la liste.

Exercice 0

```
def somcube(n) :
    '''renvoie la somme des cubes des chiffres d'un nombre par une méthode
arithmétique'''
    nombre = n // 10
    chiffre = n % 10
    res = pow(chiffre, 3)
    while nombre != 0 :
        chiffre = nombre % 10
        nombre = nombre // 10
        res += pow(chiffre, 3)
    return(res)

def entiers_somcube(N) :
    '''renvoie les entiers inférieurs ou égaux à N qui sont égaux à la somme
des cubes de leurs chiffres.'''
    res = [k for k in range(N+1) if k == somcube(k)]
    return (res)
```

Les résultats donnent : 0, 1, 153, 370, 371, 407.

```
def somcube2(n) :
    '''renvoie la somme des cubes des chiffres d'un nombre en utilisant
les chaînes de caractères'''
    nombre = str(n)
    res = 0
    for chiffre in nombre:
        res += int(chiffre) ** 3
    return(res)
```

Exercice 1.

```
import csv
import numpy as np

## creation du fichier de données
# def fonc(x):
#     return (1.00988282142 + (1.07221264497 - 1.00988282142) / 0.1 * x + \
#             x * (x - 0.1) * (3 * x ** 3) )
#
# x = np.linspace(0, 1.5, 16)
#
# donnees = [[val, fonc(val)] for val in x]
#
# with open('ex_001.csv', 'w', newline = '') as file :
#     ecrire = csv.writer(file)
#     for ligne in donnees:
#         ecrire.writerow(ligne)
#
# file.close()

import matplotlib.pyplot as plt
import scipy.integrate as scint

## recuperation des donnees
with open('ex_001.csv', 'r', newline = '') as file :
    lire = csv.reader(file)
    LX, LY = [],[]
    for ligne in lire :
        LX.append(float(ligne[0]))
```

```
        LY.append(float(ligne[1]))

file.close()

plt.plot(LX, LY)
plt.title('Courbe exercice 1')
plt.show()

def trapeze(lx, ly):
    '''lx et ly liste de meme longueur'''
    I = 0
    for i in range(1, len(lx)):
        I += (lx[i] - lx[i - 1]) * (ly[i] + ly[i - 1]) / 2
    return(I)

print('fonction trapeze', trapeze(LX, LY))
print('module scipy', scint.trapz(LY, x = LX))
```

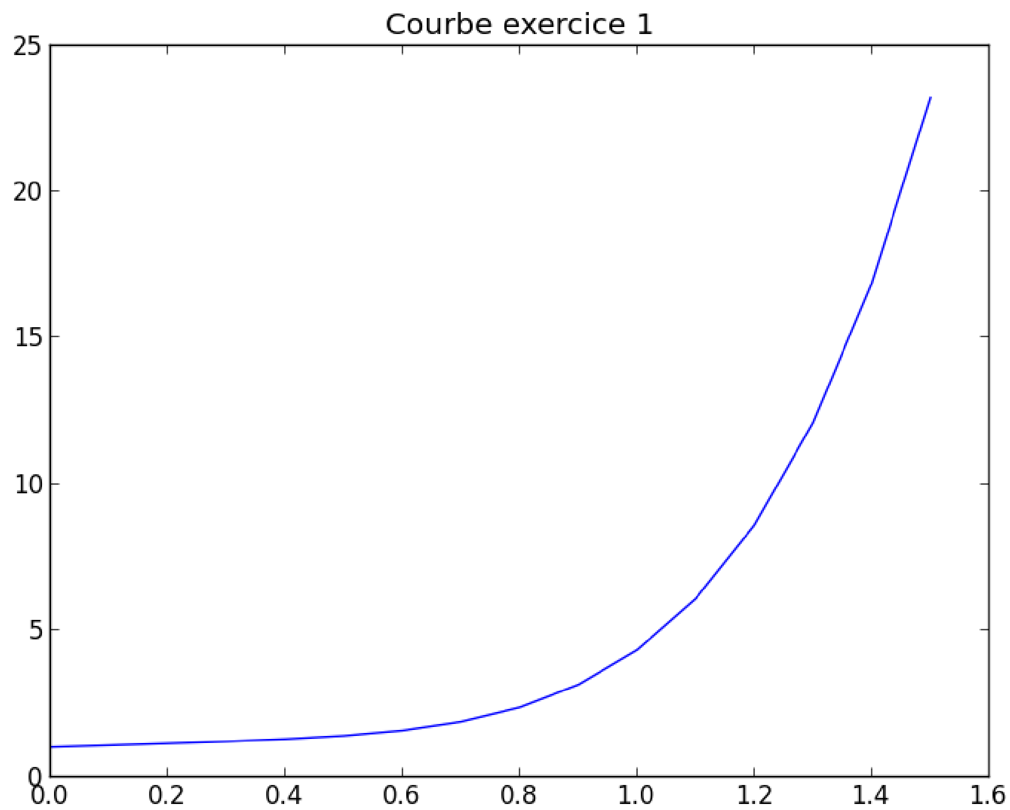

FIGURE 1 – Q2 Ex1

Exercice 2.

```
## Définition de la matrice de distance

M = [[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]]

## Définitions des fonctions

def voisins(M, s):
    '''renvoie la liste des voisins du sommet de Mij'''
```

```
    Som = []
    for k in range(len(M)):
        if M[s][k] > 0:
            Som.append(k)
    return(Som)

def degre(M, s):
    '''renvoie le nombre d'arètes issues du sommet M[s]'''
    return(len(voisins(M, s)))

def longueur(M, L):
    '''renvoie la longueur du trajet décrit par la liste L des sommets de M'''
    long = 0
    for k in range(len(L) - 1):
        if M[L[k]][L[k + 1]] == -1:
            return(-1)
        long += M[L[k]][L[k + 1]]
    return(long)
```

Exercice 3.

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

def nombreZeros(t, i):
    '''renvoie le nombre de zéros consécutifs dans le tableau t à partir
du rang i s'il est non nul.'''
    if t[i] == 1:
        return(0)
    res = 1
    T = len(t)
    if i == T - 1:
        return(1)
    j = i + 1
    while j < T and t[j] == 0:
        res += 1
        j = j + 1
    return(res)

def nombreZerosMax_naif(t):
    '''complexité en O(N**2) dans le pire des cas'''
    res = 0
    indice_max = 0
    for i in range(len(t)):
        nZ = nombreZeros(t, i)
        if nZ> res:
            res = nZ
            indice_max = i
    return(res, indice_max)

def nombreZerosMax(t):
    '''renvoie le nombre maximal de 0 contigus d'une liste t non vide,
et l'indice de départ de cette série de 0.'''
    res = 0
    indice_max = 0
    i = 0
    while i < len(t):
        nZ = nombreZeros(t, i)
        if nZ > res:
            res = nZ
            indice_max = i
        i += 1 + nZ
    return(res, indice_max)
```

Exercice 4.

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

def Arrang(k, n):
    '''arrangement de k éléments dans un ensemble de n éléments'''
```

```python
    return(m.factorial(n) / m.factorial(n - k))

def Comb(k, n):
    '''combinaison de k éléments dans un ensemble de n éléments'''
    return(Arrang(k, n) / m.factorial(k))

def Px(k, n, p):
    '''loi de Poisson, approximation de la loi binomiale pour n grand,
d'espérance n*p'''
    Lambda = n * p
    return(pow(Lambda, k) * m.exp(- Lambda) / m.factorial(k))

x = list(range(31))
Lpx = [Px(k, 30, 0.1) for k in x]

plt.figure(1, figsize = (12, 6))

plt.subplot(121)
plt.plot(x, Lpx, linestyle = '*', marker = '+', markerfacecolor = 'red')
plt.title('Loi de Poisson \n pour n = 30 et p = 0.1')

def Py(k, n, p):
    '''loi binomiale, d'espérance n*p'''
    return(Comb(k, n) * pow(p, k) * pow(1 - p, n - k))

Lpy = [Py(k, 30, 0.1) for k in x]

plt.subplot(122)
plt.plot(x, Lpy, linestyle = '*', marker = '+', markerfacecolor = 'cyan')
plt.title('Loi binomiale \n pour n = 30 et p = 0.1')

plt.show()
```

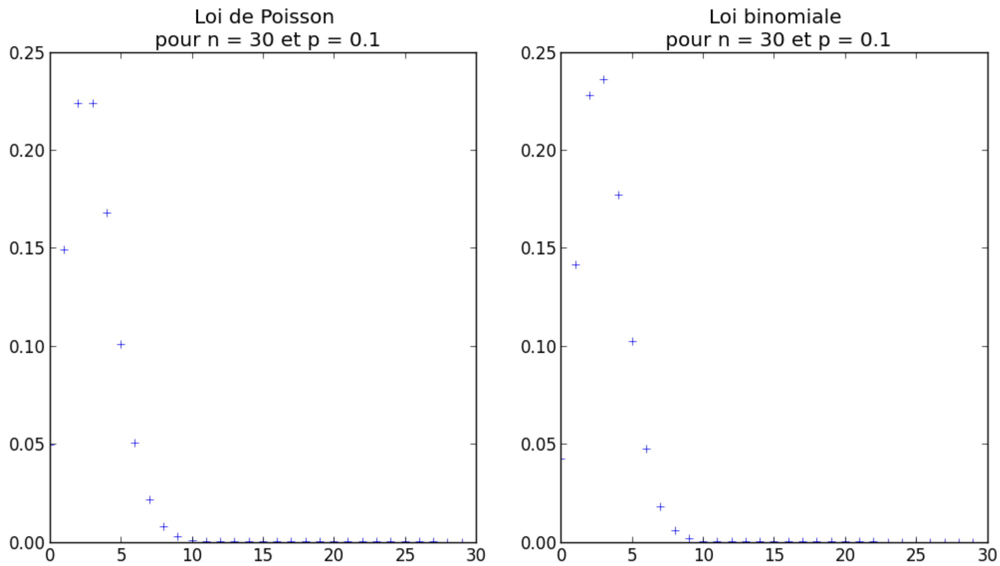

Figure 2 – Q1 & Q2 Ex4

```python
def Ecart(n, p):
    res = 0
    for k in range(n + 1):
        ecart = abs(Py(k, n, p) - Px(k, n, p))
        if ecart > res:
            res = ecart
    return(res)
```

```
Lecart1 = [Ecart(k, 0.1) for k in x]

plt.figure(2, figsize = (12, 6))

plt.subplot(121)
plt.plot(x, Lecart1, linestyle = '*', marker = '+', markerfacecolor = 'black')
plt.title('Ecart Lois binomiale/Poisson \n pour 0 <= n <= 30 et p = 0.1')

Lecart2 = [Ecart(k, 0.075) for k in x]

plt.subplot(122)
plt.plot(x, Lecart2, linestyle = '*', marker = '+', markerfacecolor = 'black')
plt.title('Ecart Lois binomiale/Poisson \n pour 0 <= n <= 30 et p = 0.075')

plt.show()
```

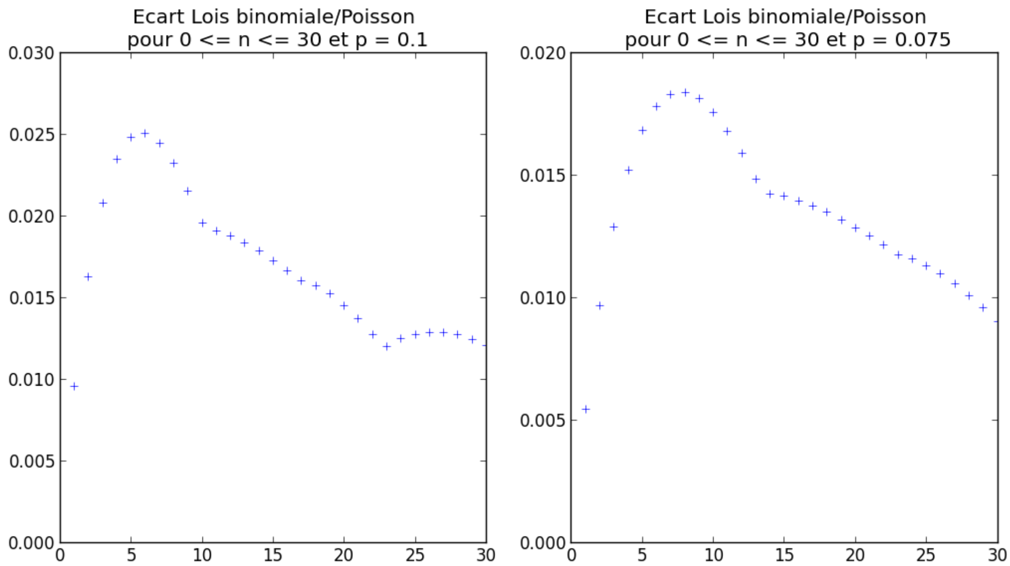

Figure 3 – Q5 Ex4

```
def N(e, p):
    res = 0
    while Ecart(res, p) > e:
        res += 1
    return(res)

Lp = [0.075, 0.1]
Le = [0.008, 0.005]
```

Exercice 5.

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

def g(x):
    '''renvoie x pour 0 <= x < 1, et 1 pour 1 <= x <= 2'''
    if 0 <= x and x < 1:
        return(x)
    if 1<= x and x < 2:
        return(1)
    return(False)

x = np.arange(0, 2, 0.01)
y = [g(k) for k in x]
```

```
plt.figure(1, figsize = (12, 6))

plt.subplot(121)
plt.plot(x, y, 'r')
plt.title('Courbe représentative \n de la fonction g')

def f(x):
    '''renvoie g(x) pour 0 <= x < 2 et sqr(x) * f(x - 2) pour x >= 2'''
    if 0 <= x and x <2:
        return(g(x))
    return(pow(x, 0.5) * f(x - 2))

x3 = np.arange(0, 6, 0.01)
y3 = [f(k) for k in x3]

plt.subplot(122)
plt.plot(x3, y3, 'b')
plt.title('Courbe représentative \n de la fonction f')

plt.show()
```

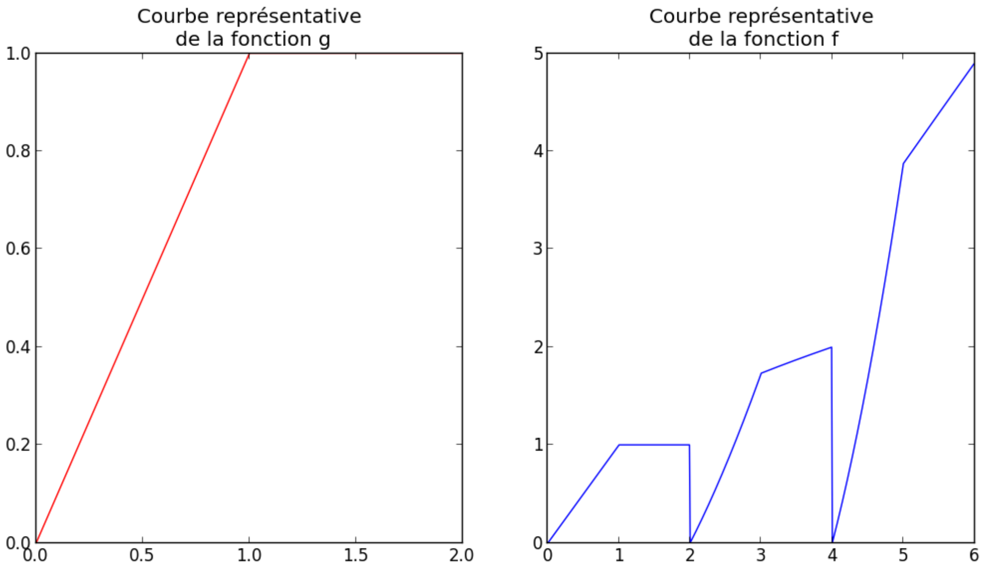

FIGURE 4 – Q1 & Q3 Ex5

```
def alpha(f, val, eps):
    '''renvoie à eps près, la plus petite valeur alpha > 0
telle que f(alpha) > val'''
    res = 0
    while f(res) <= val  :
        res += eps
    return(res)
```

On trouve finalement alpha(f, 4, 1e-2) = 5.13.

Exercice 6.

```
def d(n):

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

d(4) renvoie [1, 2, 4], et d(10) renvoie [1, 2, 5, 10].

```
def DNT(n) :
    '''renvoie la liste des diviseurs non triviaux de n
(différents de 1 et de n)'''
    L = []
    for nombre in range(2, n) :
        if n % nombre == 0 :
            L.append(nombre)
    return (L)

def sommeCarresDNT(n) :
    '''renvoie la somme des carrés des diviseurs non triviaux de l'entier n'''
    return ( sum([k ** 2 for k in DNT(n)]) )

def entiers_sommeCarresDNT(N) :
    '''affiche tous les nombres entiers inférieurs à N et égaux à la
somme des carrés de leurs diviseurs non triviaux'''
    return( [k for k in range(N + 1) if k == sommeCarresDNT(N)] )
```

`entiers\_sommeCarresDNT(1000)\ renvoie [], et \verb`entiers_sommeCarresDNT(10000)| également. Au delà, il faudrait être patient !

En effet le calcul de complexité donne :

$$C(entiers\_sommeCarresDNT) = (N+1) * C(SommeCarrresDNT)$$

$$C(entiers\_sommeCarresDNT) = (N+1) * C(DNT) = (N+1) * O(N) = O(N^2)$$

On pourrait améliorer cela avec un procédé de mémoïsation sur DNT(n)

Exercice 7.

```
na = ord('a')

alphabet = ''

for n in range(na, na + 26) :
    alphabet += chr(n)

def decalage(n) :
    '''renvoie l'alphabet décalé de n lettres'''
    alphabet_dec = ''
    for k in range(na + n, na + n + 26) :
        alphabet_dec += chr((k - na) % 26 + na)
    return (alphabet_dec)

def indices(x, phrase) :
    '''renvoie la liste des indices de x dans phrase'''
    ind = []
    for i in range(len(phrase)) :
        if phrase[i] == x :
            ind.append(i)
    return(ind)

def codage(n, phrase) :
    '''renvoie phrase codé avec un décalage de n lettres'''
    # la solution demandée est plutôt lourde...

def codage_perso(chaine, dec) :
    '''renvoie la chaine de caractère décalée de dec'''
    res = ''
    for k in chaine :
        res += chr((ord(k) - na + dec) % 26 + na)
    return(res)
```

Exercice 8.

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

M = 20
m = 10
```

```
def f(c, m = 10, M = 20):
    '''m et M définis par défaut'''
    uk = 0
    for k in range(m + 1):
        uk = uk ** 2 + c
        if abs(uk) > M:
            return(k)
    return(m + 1)

LX = np.linspace(-2, 2, 401)
LY = [f(c) for c in LX]

plt.figure(1, figsize = (12, 6))

plt.subplot(121)
plt.plot(LX, LY, 'r')
plt.ylim([0, 11])
plt.title('Courbe représentative de f sur [-2;2]')


LXZ = np.linspace(-2, 0.5, 101)
LYZ = np.linspace(-1.1, 1.1, 101)

LZ = [ [complex(x, y) for y in LYZ] for x in LXZ ]

Image = [ [f(complex(x, y)) for y in LYZ] for x in LXZ ]

plt.subplot(122)
plt.title('Image du tableau de valeurs')
plt.imshow(Image)

plt.show()
```

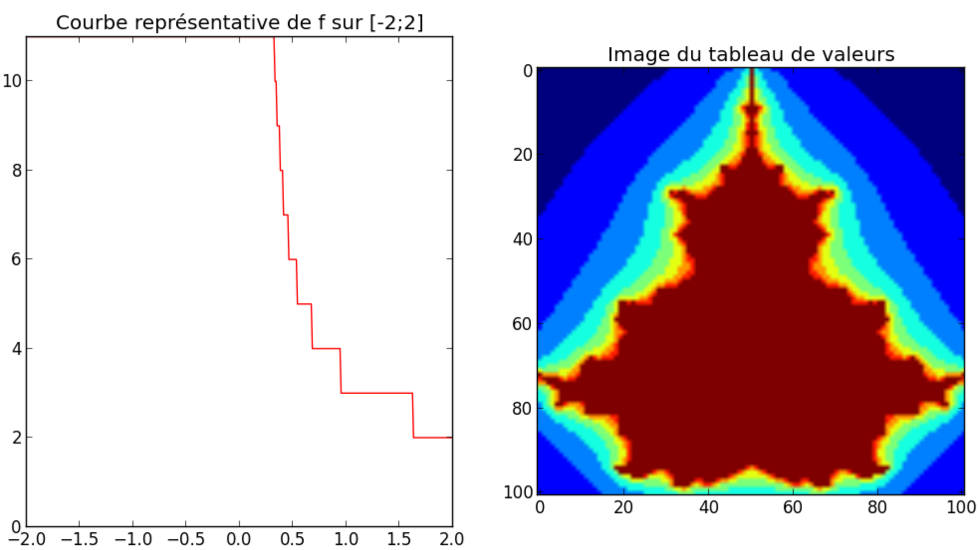

FIGURE 5 – Q2 & Q4 Ex8

On "reconnait" la fractale d'un ensemble de Julia.

Exercice 9.

```
import numpy as np
import numpy.linalg as npl

## Génération fichier ex_009.txt
##import random as rd
```

```python
##
##n = 5
##vals_propres = [rd.random() for i in range(n)]
##M_diag = np.diag(vals_propres)
##P = np.random.rand(n,n)
##M = np.dot(np.dot(np.linalg.inv(P), M_diag), P)
##
##file = open('Ex_009.txt', 'w')
##
##for ligne in M:
##    for elt in ligne:
##        file.write(str(elt) + ' ')
##    file.write('\n')
##
##file.close()

R1 = np.array([[1, 2, 3], [4, 5, 6]])
R2 = np.array([[i + 3 * j for i in range(1, 4)] for j in range(2)])
S1 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
S2 = np.array([[i + 3 * j for i in range(1, 4)] for j in range(3)])

def test(M):
    '''test si la matrice M est carrée, renvoie alors sa dimension, 0 sinon'''
    (l, c) = np.shape(M)
    if l == c:
        return(l)
    return(0)

Lambda, vecteurs = npl.eig(S1)

print("Valeurs propres : ", Lambda)
print("Vecteurs propres : ", vecteurs)

def dansIntervalle(L, a, b):
    '''renvoie True si tous les éléments de la liste L sont dans
l'intervalle [a, b] et False sinon'''
    m = min(a, b)
    M = max(a, b)
    for elt in L:
        if elt < m or elt > M:
            return False
    return True

## Récupération des données de Ex_009.txt

file = open('Ex_009.txt', 'r')
Tableau = file.readlines()
L = []
j = 0
for Ligne in Tableau:
    Liste_ligne = Ligne.strip().split()
    L.append([])
    for elt in Liste_ligne:
        L[j].append(float(elt))
    j += 1
file.close()

MAT = np.array(L)
print(MAT)

print(dansIntervalle(npl.eig(MAT)[0], 0, 1))
```

Exercice 10.

```python
import random as rd

L_test = [rd.randint(0, 5) for k in range(20)]

print(L_test)

def comptage(L, N):
```

```
    '''renvoie une liste P dont le k-ième élément désigne le nombre
d'occurences de l'entier k dans la liste L'''
    P = [0 for k in range(N + 1)]
    for elt in L:
        if elt <= N :
            P[elt] += 1
    return(P)

def tri(L, N):
    '''renvoie la liste L triée en utilisant la fonction comptage'''
    R = []
    LC = comptage(L, N)
    for i in range(len(LC)):
        R += [i for k in range(LC[i])]
    return (R)
```

Étude de la Complexité :

$$C(tri) = C(comptage) + N * O(N) = [O(N+1) + O(n)] + O(N^2) = O(n) + O(N^2)$$

$$C(tri\_insertion) = O(n^2) \text{au pire des cas}, O(n) \text{au meilleur des cas}$$

$$C(tri\_fusion) = O(n.log(n))$$

Exercice 11.

```
import math as m

def p(t, b = 0.5, w = 6.):
    '''renvoie la position dans le plan d'une masse ponctuelle mobile
au cours du temps'''
    return( m.cos(t) + b * m.cos(w * t), m.sin(t) + b * m.sin(w * t) )

def v(t, b = 0.5, w = 6.):
    '''renvoie la vitesse dans le plan d'une masse ponctuelle mobile
au cours du temps'''
    return( -m.sin(t) - b * w * m.sin(w * t), m.cos(t) + b * w * m.cos(w * t) )

def a(t, b = 0.5, w = 6.):
    '''renvoie l'accélération dans le plan d'une masse ponctuelle mobile
au cours du temps'''
    return( -m.cos(t) - b * pow(w, 2) * m.cos(w * t), \
            -m.sin(t) - b * pow(w, 2) * m.sin (w * t) )

## tests sur un exemple
t = [0, m.pi / 2, m.pi]
for val in t:
    print('p(', val, ')=', p(val))
    print('v(', val, ')=', v(val))
    print('a(', val, ')=', a(val))

import numpy as np

L = [p(t) for t in np.arange(-np.pi, np.pi, 0.01 * np.pi)]

import matplotlib.pyplot as plt

x = [elt[0] for elt in L]
y = [elt[1] for elt in L]

plt.figure(1, figsize = (12, 6))

plt.subplot(121)
plt.plot(x, y, 'red')
plt.title('Tracé de la position du mobile au cours du temps')


def c(t, b = 0.5, w = 6.):
    '''renvoie le couple des coordonnées du centre de courbure'''
    x, y = p(t, b, w)
    dx, dy = v(t, b, w)
    d2x, d2y = a(t, b, w)
```

```
    d = (pow(dx, 2) + pow(dy, 2)) / (dx * d2y - dy * d2x)
    return(x - d * dy, y + d * dx)

Lc = [c(t) for t in np.arange(-np.pi, np.pi, 0.01 * np.pi)]

cx = [elt[0] for elt in Lc]
cy = [elt[1] for elt in Lc]

plt.subplot(122)
plt.plot(x, y, 'red')
plt.plot(cx, cy, 'blue')
plt.title('Tracé de la position du mobile au cours du temps \navec les centres\
de courbure')

plt.show()
```

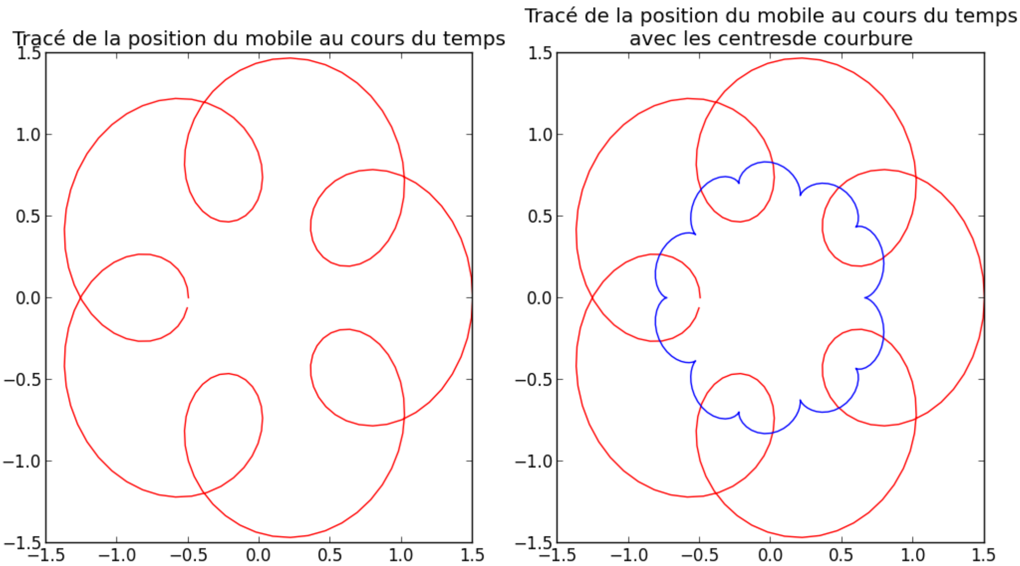

FIGURE 6 – Q3 & Q5 Ex11

```
def longueur(L_points):
    '''renvoie la longueur de la ligne polygonale reliant les points passés
en arguments'''
    return( sum( [np.linalg.norm( \
            np.array(L_points[i + 1]) - np.array(L_points[i])) \
                for i in range(len(L_points) - 1)] ) )

def evolution_longueurs(n_min, n_max):
    '''renvoie une liste de longueurs pour différentes valeurs de pas de temps
telles que dt = pi * 1e-exp pour exp compris entre n_min et n_max'''
    long = []
    for exp in range(n_min, n_max):
        dt = np.pi * pow(10, -exp)
        L = [p(t) for t in np.arange(-np.pi, np.pi, dt)]
        long.append(longueur(L))
    return(long)
```

Résultats pour `evolution_longueurs(2, 6)` :
[19.286913503237365, 19.370341677646834, 19.376265406836875, 19.376833582034649]