```python
#Correction_info4

#EX1

#Dans les deux premières questions, la somme de Riemann représente l'aire
des rectangles construits à partir des points retenus.

def somme(n:int)->float:
    """renvoie la somme des aires des rectangles à droite"""
    f=lambda t:1/(1+t**2)
    S=0
    for i in range(1,n+1):
        S=S+(1/n)*f(i/n)
    return S

def y(n:int)->list:
    """renvoie la liste des valeurs approchées aux points de la subdivision
par la méthode d'Euler"""
    h=1/n
    T=[k/n for k in range(0,n+1)]
    y0=1
    L=[y0]
    for k in range(1,n+1):
        y=L[k-1]+h*tan(T[k-1])*L[k-1] #attention au schéma qui tient compte
de t_{k-1} au rang k
        L.append(y)
    return L

#On applique la méthode habituelle, et on trouve que f:t->1/cos(t) désigne
l'unique solution du problème de Cauchy.

from pylab import *
def euler(n:int)->None:
    T=[k/n for k in range(0,n+1)]
    Y=y(n)
    Z=[1/cos(t) for t in T]
    plot(T,Z,label='solution exacte')
    plot(T,Y,label='solution approchée')
    legend()
    show()

#EX2
def somme(n:int)->float:
    """renvoie la somme des aires des rectangles à droite"""
    f=lambda t:1/(1+t**2)
    S=0
    for i in range(1,n+1):
        S=S+(1/n)*f(i/n)
    return S

def trapezes(n:int)->float:
    """renvoie la somme des aires des trapèzes"""
    f=lambda t:1/(1+t**2)
    S=0
    for i in range(0,n):
        a=i/n
        b=(i+1)/n
        S=S+(1/n)*(f(a)+f(b))/2
    return S
```



```python
def simpson(n:int)->float:
    """renvoie la somme des aires des paraboles qui interpolent la
fonction"""
    f=lambda t:1/(1+t**2)
    S=0
    for i in range(0,n):
        a=i/n
        b=(i+1)/n
        m=(a+b)/2 #le point milieu
        S=S+(1/n)*(f(a)+4*f(m)+f(b))/6
    return S

from math import *
def comparaison(eps:float)->tuple:
    """renvoie les seuils à partir duquel on a appoximation par nos 3
méthodes"""
    n1=1
    while abs(somme(n1)-pi/4)>eps:
        n1=n1+1
    n2=1
    while abs(trapezes(n2)-pi/4)>eps:
        n2=n2+1
    n3=1
    while abs(simpson(n3)-pi/4)>eps:
        n3=n3+1
    return n1,n2,n3

#EX3
def bornes(n:int)->tuple:
    f=lambda x:x**2-5*x/2+1
    a0,b0=1,4
    for k in range(1,n+1):
        m=(a0+b0)/2
        if f(a0)*f(m)<0:
            a1,b1=a0,m #dans ce cas, on définit les nouveaux curseurs
        else:
            a1,b1=m,b0 #dans ce cas, on définit les nouveaux curseurs
        a0,b0=a1,b1 #on réinjecte pour l'étape suivante
    return a0,b0

def approx(p:int)->int:
    n=0
    while abs(bornes(n)[1]-bornes(n)[0])>10**(-p):
        n=n+1
    return n

from pylab import *
def vitesse(N:int)->None:
    P=[k for k in range(1,N+1)]
    L=[approx(p) for p in P]
    plot(P,L)
    show()

#Les curseurs représentent des suites adjacentes car (an) est croissante et
(bn) décroissante, et à chaque étape la longuer de l'intervalle est divisé
par 2, autrement dit bn-an=(b0-a0)/2**n qui tend vers 0. Comme an<x<bn, on a
par passage à la limite dans l'encadrement et en notant l la limite commune
: l=x.
```