Lycée Louis-Le-Grand, Paris
MPSI 4 – Informatique commune
N. Carré, A. Troesch

2018/2019

# TP n° 11 : Équations différentielles

**Correction de l'exercice 1 –**

1. Ici, la résolution de l'équation en $y_{k+1}$ peut se faire mathématiquement de façon explicite, et aboutit à l'identité :

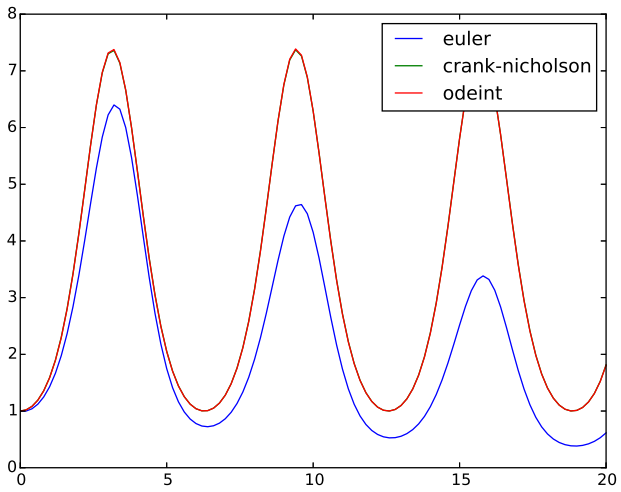
$$y_{k+1} = \frac{1 + \sin(t_k)\frac{t_{k+1}-t_k}{2}}{1 - \sin(t_{k+1})\frac{t_{k+1}-t_k}{2}}.$$

2. On implémente les deux méthodes Euler et Crank-Nicholson, et on trace les courbes correspondantes, ainsi que celle obtenue avec odeint avec le même nombre de pas :

```
import scipy.integrate as si
import numpy as np
from math import sin
import matplotlib.pyplot as plt

def F(y,t):
    return y * sin(t)

def euler(F,y0,t0,t1,n):
    """euler entre t0 et t1 initialisé avec y0, et n pas"""
    T = np.linspace(t0,t1,n+1)
    Y = [y0]
    for k in range(len(T)-1):
        Y.append(Y[-1] + F(Y[-1],T[k]) * (T[k+1]-T[k]))
    return T, Y

def cranknicholsonpart(y0,t0,t1,n):
    """CN entre t0 et t1 initialisé avec y0 et n pas pour la fonction
    de l'énoncé uniquement"""
    T = np.linspace(t0,t1,n+1)
    Y = [y0]
    for k in range(len(T)-1):
        h = (T[k+1]-T[k])/2
        Y.append(Y[-1] * (1+sin(T[k]) * h)/(1-sin(T[k+1])*h))
    return T,Y


def trace1(y0,t0,t1,n):
    T1,Y1 = euler(F,y0,t0,t1,n)
    T2,Y2 = cranknicholsonpart(y0,t0,t1,n)
    T3 = np.linspace(t0,t1,n+1)
    Y3 = si.odeint(F,y0,T3)
    plt.plot(T1,Y1, label = 'euler')
    plt.plot(T2,Y2, label = 'crank-nicholson')
    plt.plot(T3,Y3, label = 'odeint')
```

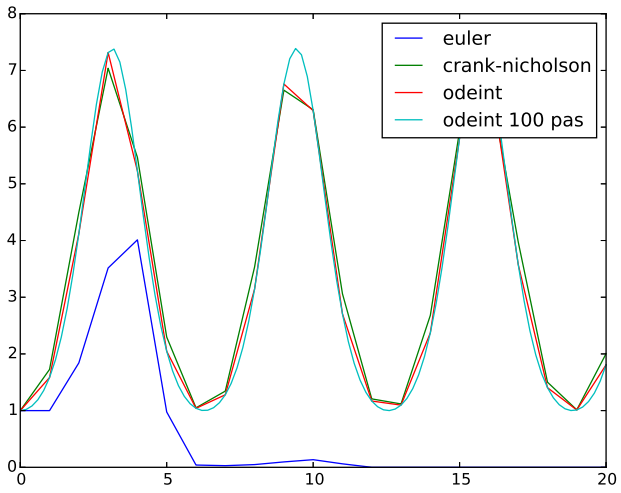
```
    plt.legend()
%   plt.savefig('py088-2.eps')
    plt.show()

trace1(1,0,20,20)
```

Avec 100 pas, on obtient les graphes suivants :




Les courbes de odeint et de Crank-Nicholson sont superposées.

3. Si on diminue le nombre de pas à 20, le résultat pour le schéma de Crank-Nicholson reste bon. On a également conservé le tracé de odeint avec n=100 pour comparer.

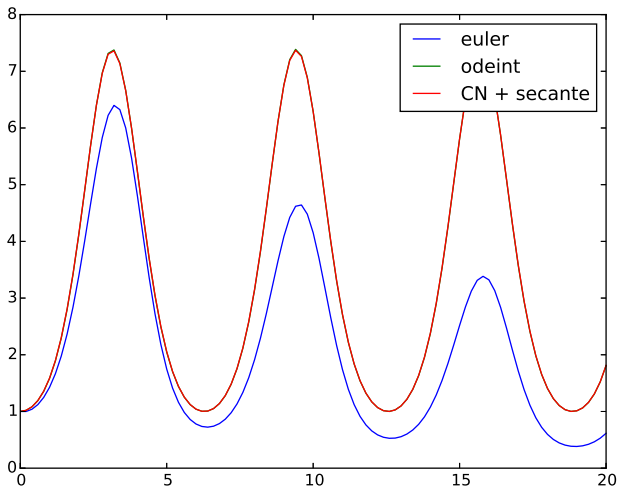



4. On résout l'équation avec la méthode de la sécante, la plus efficace ne cécessitant pas de connaître la dérivée de l'équation.

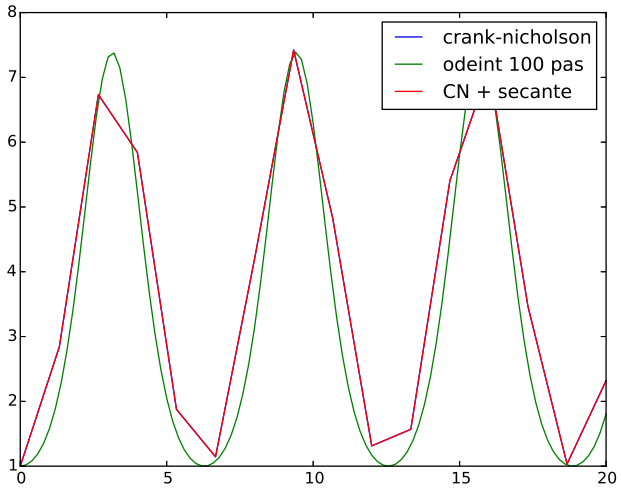
```
def secante(f,a,b,e):
    while b-a > e:
        c = a - (b-a)*f(a)/(f(b)-f(a))
        a,b=b,c
    return c

def cranknicholson(F,y0,t0,t1,n):
```

```
    """CN entre t0 et t1 initialisé avec y0 et n pas pour la fonction
    de l'énoncé uniquement"""
    T = np.linspace(t0,t1,n+1)
    Y = [y0]
    for k in range(len(T)-1):
        f = lambda x: Y[-1]+ (F(Y[-1],T[k])+ F(x,T[k+1]))/2 * (T[k+1]-T[k]) -x
        Y.append(secante(f,Y[-1],Y[-1]+1, 1e-10))
    return T,Y
```

Pour la fonction précédente et 100 pas, on obtient :



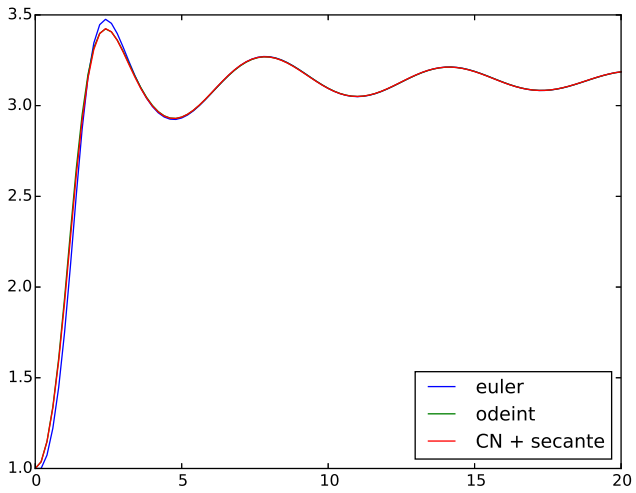

Et avec 20 pas :



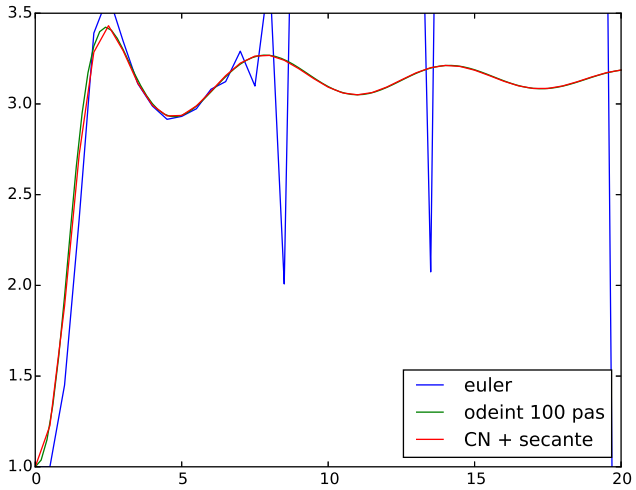

La courbe est confondue avec la courbe obtenue par résolution « mathématique » de l'équation.

On peut l'expérimenter sur d'autres équations différentielles pour lesquelles une résolution mathématique est plus délicate. Par exemple l'équation différentielle $y' = tsin(y) + sin(t)$.

Les courbes obtenues pour 100 pas sur l'intervalle $[0, 20]$ :




Et pour 40 pas, Euler renvoie n'importe quoi, alors que Crank-Nicholson tient la route :

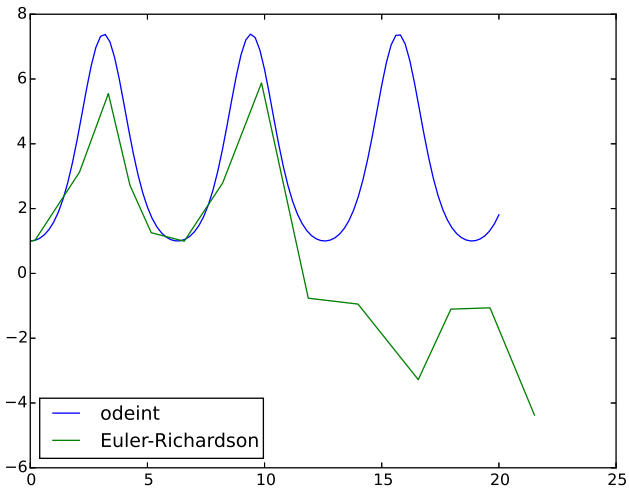



**Correction de l'exercice 2** –
Le code demandé, avec les tracés, par cette méthode, et avec odeint pour comparer.

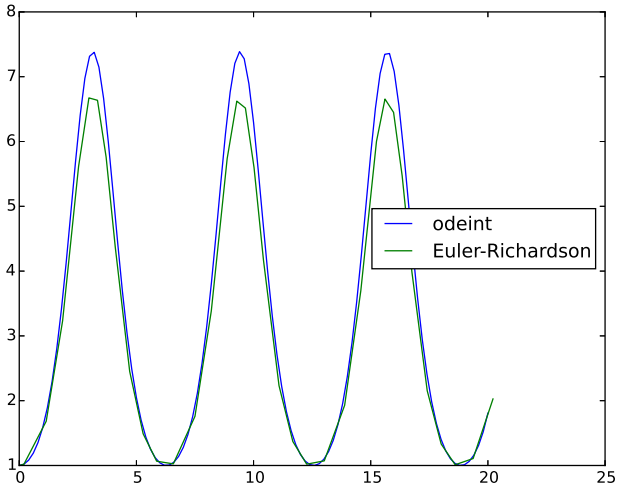
```
from math import sqrt,sin
import numpy as np
import matplotlib.pyplot as plt
import scipy.integrate as si

def eulerrichardson(F,y0,t0,t1,eps,h):
    T=[t0]
    Y=[y0]
    k = 0
    while T[-1] < t1:
        p1 = F(Y[-1],T[-1])
        p2 = F(Y[-1] + p1 * (h/2), T[-1] + (h/2))
        e = (h/2) * abs(p2-p1)
        while e > eps:
            h = 0.95 * h * sqrt(eps/e)
            p1 = F(Y[-1],T[-1])
```

```
            p2 = F(Y[-1] + p1 * (h/2), T[-1] + (h/2))
            e = (h/2) * abs(p2-p1)
        T.append(T[-1]+ h)
        Y.append(Y[-1]+ p2 * h)
        h *= 0.95 * sqrt(eps/e)
    return T,Y

def F(y,t):
    return y * sin(t)

def trace1(F,y0,t0,t1,n,eps,h):
    T1 = np.linspace(t0,t1,n+1)
    Y1 = si.odeint(F,y0,T1)
    T2,Y2 = eulerrichardson(F,y0,t0,t1,eps,h)
    print('Nombre de subdivisions: {}'.format(len(T2)))
    plt.plot(T1,Y1, label = 'odeint')
    plt.plot(T2,Y2, label = 'Euler-Richardson')
    plt.legend(loc = 'best')
    plt.savefig('py089-2.eps')
    plt.show()
```

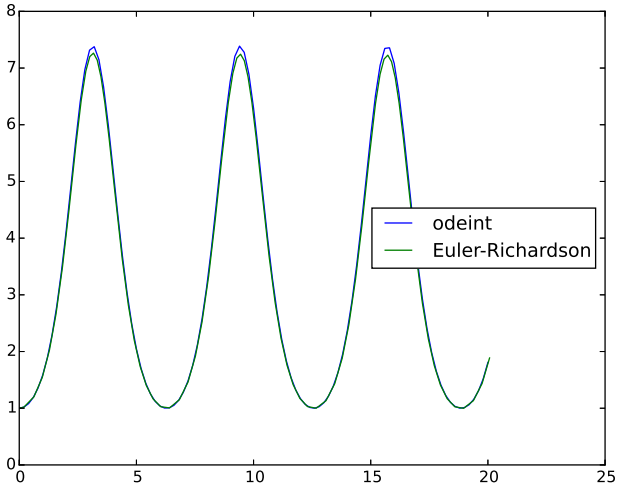
Le graphe obtenu avec $h = 0.2$ (pas initial correspondant à 100 subdivisions si on considère la subdivision régulière), et $\varepsilon = 1$ :




La précision n'est clairement pas suffisante. Le nombre de pas est de 15.
Avec $\varepsilon = 0.25$, c'est déjà beaucoup mieux avec seulement 36 pas :




Avec $\varepsilon = 0.05$, le résultat est presque parfait, avec 77 pas.

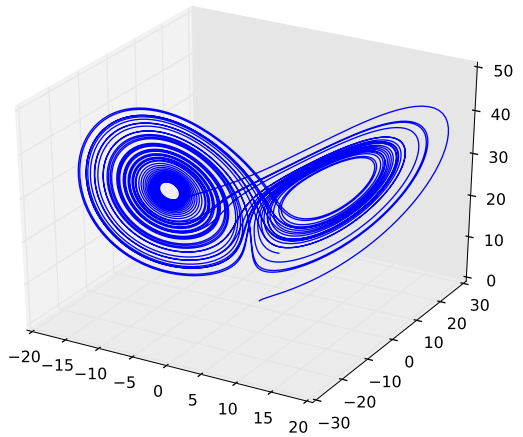



**Correction de l'exercice 3** –
Pour le calcul et le tracé :

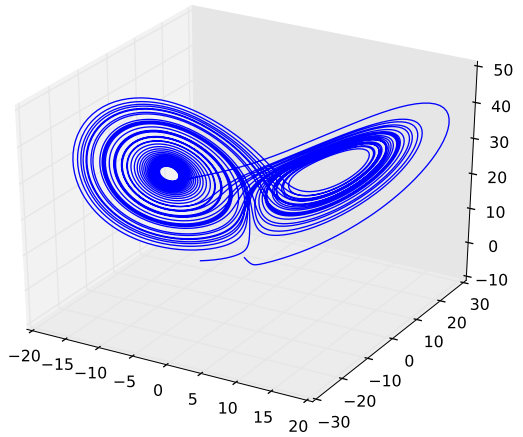
```
import scipy.integrate as si
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt

def F(X,t,sig,rho,beta):
    x1= sig *(X[1]-X[0])
    x2=rho*X[0]-X[1]-X[0]*X[2]
    x3=X[0]*X[1] - beta*X[2]
    return [x1,x2,x3]

def lorenz(sig,rho,beta,x0,y0,z0,t0,t1,n):
    T= np.linspace(t0,t1,n+1)
    V= si.odeint(lambda X,t: F(X,t,sig,rho,beta),[x0,y0,t0],T)
    X= [v[0] for v in V]
```

```
    Y= [v[1] for v in V]
    Z= [v[2] for v in V]
    return X,Y,Z

X1,Y1,Z1 = lorenz(10,28,8/3, -2,3,-5,0,50,10000)
plt.axes(projection='3d').plot(X1,Y1,Z1)
plt.savefig('py090-2.eps')
plt.show()
```

Pour la condition initiale $(1, 1, 1)$, on obtient :




Pour la condition initiale $(-2, 3, -5)$ :




Pour tester le caractère chaotique, on rajoute la fonction suivante :

```
def chaos(V,dV):
    X,Y,Z=lorenz(10,28,8/3,V[0],V[1],V[2],0,50,10000)
    print('CI en V : {}'.format((X[-1],Y[-1],Z[-1])))
    X,Y,Z=lorenz(10,28,8/3,V[0]+ dV[0],V[1]+dV[1],V[2]+dV[2],0,50,10000)
    print('CI en V+dV : {}'.format((X[-1],Y[-1],Z[-1])))

chaos((1,1,1),(0.0000001,0.0000001,-0.0000001))
```

Après quelques essais, on se rend compte que même avec des valeurs très petites de `dV`, on obtient des résultats très différents. Par exemple, pour l'exemple donné dans le code, on obtient :

```
CI en V : (2.5200778925909653, 4.5438074771160473, 10.66460526673729)
CI en V+dV : (-1.5054992331291279, -2.6983087728036148, 17.73656306364666)
```