```
# -*- coding: utf-8 -*-
"""
Correction du DM1

"""

## Question 1

""" Comme x est deux fois dérivables en
t, on peut lui appliquer deux fois la
formule de Taylor-Young à l'ordre 2,
entre x et x+h d'ne part, entre x et x-h
d'autre part. Ceci donne :
x(t+h)=x(t)+h*x'(t)+h**2/2*x"(t)+o(h**2)
x(t-h)=x(t)-h*x'(t)+(-
h)**2/2*x"(t)+o(h**2)

En sommant les lignes, il vient :
x(t+h)+x(t-h)=2*x(t)+h**2*x"(t)+o(h**2)
D'où :  x"(t) = (x(t+h)+x(t-
h)-2*x(t))/h**2 + o(1)
Ceci équivaut exactement à la limite
demandée.
"""

## Question 2

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

```
def euler2DF(f,x0,v0,t0,tf,n):
    h=(tf-t0)/n
    Lt=np.linspace(t0,tf,n+1)
    ''' remarque : linspace à
privilégier devant np.arange(t0,tf+h,h),
qui ne   marche pas pour n>43, à cause
d'erreurs successives dans les
approximations de  Python, qui mènent à
ne pas tomber rigoureusement sur tf pour
le dernier nœud de discrétisation
    '''
    Lx=np.zeros(n+1)
    Lx[0]=x0
    Lx[1]=v0*h+x0
    for k in range(1,n):
        Lx[k+1]=2*Lx[k]-
Lx[k-1]+h**2*f(Lt[k],Lx[k],(Lx[k]-
Lx[k-1])/h)
    pl.grid(True)
    pl.plot(Lt,Lx,label="Euler pour
n="+str(n)+" et pour x0="+str(x0))
    pl.legend()
    pl.show()
    return (Lt,Lx)

### Question 3 : test

def f1(t,y,yp):
    return -np.sin(y)

pl.figure()
```

```
euler2DF(f1,3,0,0,30,100)


### Question 4 : solution linéarisée

def sol(t,x0,v0):
    return x0*np.cos(t)+v0*np.sin(t)

def graphelineaire(x0,v0,t0,tf,n):
    Lt=np.linspace(t0,tf,n+1)
    Lx=sol(Lt,x0,v0)  # c'est là qu'on
voit la force de Numpy : comme les
fonctions de sol sont dans Numpy, alors
on peut les appliquer directement à des
tableaux.
    pl.grid(True)
    pl.plot(Lt,Lx,'r',label='solution
linéarisée')
    pl.legend()
    pl.show()

graphelineaire(3,0,0,30,100)

""" On peut affirmer que les 2 courbes
vont débuter identiquement, mais se
différencier très rapidement du fait que
l'angle initial de 3 rad n'est pas assez
proche de 0 pour que l'approximation sin
x ~ x soit valide.

Remarque : on peut s'amuser à réduire
```

```
l'angle initial x0, poru voir à partir
de quand l'approximation donne, sur
[0,30], des courbes similaires.
"""

### Question 5 : approximation de la
vitesse

""" Si on utilise l'approximation
décentrée (à droite) : x'(t) ~ (x(t+h)-
x(t))/h, on aura un problème pour le
calcul de la dernière vitesse (en tf),
puisque eulerDF ne donne pas une
approximation de x(tf+h).

Pour résoudre cette difficulté,
plusieurs alternatives.
On choisit ci-dessous d'approcher la
vitesse par le taux rétrograde (à
gauche) : x'(t) ~ (x(t)-x(t-h))/h

RMQ : on peut prouver qu'il serait plus
précis de prendre l'approximation
centrée (à 2 pas) : x'(t) ~ (x(t+h)-x(t-
h))/(2*h).
Mais ceci poserait toujours un pb à la
fin..."""

def vitesseDF(f,x0,v0,t0,tf,n):
    Lt,Lx=euler2DF(f,x0,v0,t0,tf,n)
    # Avec les listes :
```

```
    Lv=[v0]
    for k in range(1,len(Lt)):
        h=Lt[k]-Lt[k-1]    # par exemple
! Au cas où le pas ne soit pas constant.
        Lv.append((Lx[k]-Lx[k-1])/h)
    return Lv

    """ Pour ceux qui ont adopté Numpy :
    Lv= np.zeros(n+1)
    Lv[0]=v0
    for k in range(1,len(Lt)):
        h=Lt[k+1]-Lt[k]
        Lv[k]=(Lx[k+1]-Lx[k])/h
    return Lv
    """

# Test (non demandé) :

print("La liste des vitesse approchée
est : ", vitesseDF(f1,3,0,0,30,100))
```