Corrigé 3 du sujet 0 de SI (proposé pour le programme de 2015)
La fonction ω_m est solution de l'équation scalaire d'ordre 2

$$\frac{1}{\omega_0^2}\frac{d^2}{dt^2}\omega_m(t) + \frac{2m}{\omega_0}\frac{d}{dt}\omega_m(t) + \omega_m(t) = K\omega_{cons}(t) \tag{4.3}$$

ssi la fonction $Y : t \to \begin{bmatrix} \omega_m(t) \\ \frac{d}{dt}\omega_m(t) \end{bmatrix}$ est, quant à elle, solution de l'équation vectorielle d'ordre 1

$$\frac{d}{dt}Y(t) \qquad = F(t, Y(t)) = \begin{bmatrix} \frac{d}{dt}\omega_m(t) \\ -2m\omega_0\frac{d}{dt}\omega_m(t) - \omega_0^2\omega_m(t) + K\omega_0^2\omega_{cons(t)}(t) \end{bmatrix} \tag{4.4}$$

avec (4.5)

$$F(t, Y) \quad = \begin{bmatrix} Y_2 \\ -2m\omega_0 Y_2 - \omega_0^2 Y_1 + K\omega_0^2\omega_{cons(t)} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\omega_0^2 & -2m\omega_0 \end{bmatrix} Y + \begin{bmatrix} 0 \\ K\omega_0^2\omega_{cons(t)} \end{bmatrix} \tag{4.6}$$

Si le cours de maths donne les moyens d'exprimer les solutions sous forme intégrale (fin d'année), il reste judicieux vu le besoin pratique (temps de réponse à 5%) de rechercher des solutions numériques d'emblée. Si nous prenons $w_{cons} = u$ (fonction de Heaviside (notation ?)) et nous restreignons à $T \geq 0$, on retrouve l'équation de l'énoncé.

Q 26 Calcul du tableau T :

On propose 3 solutions (fixez vous en une et gardez la)

```
import numpy as np

def subdivision(Tmax, N):
    '''D'après l'enonce, N points, N-1 intervalles'''

    T =  np.zeros(N, dtype = float)
    pas = Tmax/(N-1)
    for k in range(1,N):
        T[k] = T[k-1] + pas
    return T

>>> subdivision(10, 11)
array([  0., 1., 2., 3.,4., 5., 6., 7., 8., 9.,,10.])

def subdivision2(Tmax, N):
    pas = Tmax/(N-1)
    return  np.array([k*pas for k in range(0,N)])

>>> subdivision2(10, 11)
array([  0., 1., 2., 3.,4., 5., 6., 7., 8., 9.,,10.])

>>> np.linspace(0, 10, 11)
array([  0., 1., 2., 3.,4., 5., 6., 7., 8., 9.,,10.])
```

Q 27

On évite les calculs inutiles dans f1

```
K,  m, w0  =  1, 1,1
a, b, c    =  -w0**2, -2*m*w0, K*w0**2

def  f1(ti, Yi):
    '''ti flottant, ti n'intervient pas ici;
       Y np.array de taille 2 (matrice 1 lignes, 2 colonnes)'''
                    '''
    return np.array([Yi[0,1], c +a*Yi[0,0]+b*Yi[0,1]])
```

Q 28 On exprime Y_{i+1} en fonction de Y_i selon le schéma d'Euler (ce n'est pas précisé mais *what else ?*) :

$$\boxed{Y_{i+1} = Y_i + pas * F(t_i, Y_i)}$$

Q 29

Le schéma d'Euler (explicite)

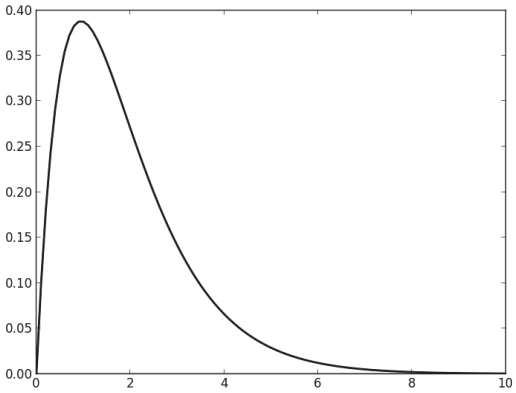
```
def EulerExplicite(Yini, h, Tmax, F):
    ''' D'après l'enonce, N points, N-1 intervalles'''
    ''' pas = T/(N-1)                                '''

    N        =   int(np.floor(Tmax/h)) +1
    T        =   subdivision(Tmax, N)
    SY       =   np.matrix(np.zeros((N,2), dtype=float))
    SY[0, :] =   Yini

    for k in range(0, N-1):
        SY[k+1, :] = SY[k,  :]+ h*f1(T[k],  SY[k,  :])
    return  SY

Y0      = np.array([0,0])
pas     = 0.1
Tmax    = 2
SY      = EulerExplicite(Y0, pas, Tmax, f1)
print(SY)

>>> [[ 0.          0.        ]
 [ 0.          0.1       ]
 [ 0.01        0.18      ]
 [ 0.028       0.243     ]
 [ 0.0523      0.2916    ]
 .../...
```



1000 pas

La question n'est pas vraiment en cohérence avec le choix fait en Q 26 pour définir la subdivision à pas constant (c'est le nombre de points qui est fixé). Mieux vaudrait donner N que *pas* en argument. Nous choisirons pour nous conformer à la signature de chacune des deux fonctions, de prendre un pas constant « proche » de celui donné en argument et de conserver scrupuleusement les bornes 0 et Tmax.

Q 30 L'erreur théorique ou l'erreur d'approximation en flottants ?

Premier cas : l'erreur théorique est $e_N \sim O\left(\frac{1}{N}\right)$ donc en divisant la pas par 10 on multiplie N par 10 et l'erreur diminue d'un facteur 10. Hors programme me semble-t-il.

Second cas : l'erreur d'approximation en tenant compte des calculs en flottants ? Elle diminue comme la précédente jusqu'à une certaine valeur puis augmente à cause des erreurs de calcul. Pour vous, c'est de l'expérimental...

Q 31 Nous allons considérer le nombre d'appels à f1 pour notre mesure de complexité : il y a $N-1$ appels à f1 ; or $N_h = [Tmax/h]$. La complexité est donc

$$C(h) \underset{h\to 0}{\sim} [Tmax/h] \underset{h\to 0}{\sim} N_h \underset{h\to 0}{=} O\left(\frac{1}{h}\right).$$

En divisant h par 10 on multiplie par 10 le nombre d'opérations et, comme la complexité est linéaire, on multiplie le temps de calcul par 10.

En choisissant de dénombrer les opérations on obtient 3 multiplications, 3 additions par itération (si les constantes qui interviennent dans f1 sont précalculées) ce qui ne change rien aux réponses apportées.

Q 32 Un flottant en double précision est codé sur 64 bits, soit 8 octets.

SY contient $2 \times N$ flottants et N de plus pour stocker T. Cela fait en tout $3 \times N \times 8$ octets soit $\boxed{24 \times 10^4 \text{ octets (240 Koctets ou 240 Kbytes)}}$.

Q 33 De quoi c'est-y-qu'on cause ?

W contient les valeurs approchées des $\omega(t_k)$ pour $t_0 = 0, t_1 = h, t_2 = 2h, ..., Tmax = (N-1)h$. C'est la première colonne de SY.

La comparaison des valeurs de $\omega(t)$ à la valeur $\omega(Tmax)$ semble indiquer que l'on regarde **si l'on s'approche de la limite de la fonction, de l'état stationnaire.** On suppose donc qu'il y a un état stationnaire.

Pas grand chose à voir avec une convergence d'algorithme [3], mais on se comprend.

3. On a préféré ne pas modifier l'énoncé.

Sommes nous proches de l'état stationnaire ?

```
def TestConvergence(t,w):
    n       =  len(t)
    p       =  int(np.floor(0.9*n))
    w_lim   = w[n-1,0] # tableaux de meme longueur

    r, k    =  True, p
    while r and k<n:
        r = r  and  (abs(w[k,0]- w_lim) <= abs(w_lim)/100)
        k += 1

    return r

N= 1000
SY = EulerExplicite(Y0, Tmax/(N-1), Tmax, f1)

print(TestConvergence(subdivision(Tmax, N), SY[:,0]))
```

Q 34

Temps de réponse à 5%

```
def CalculT5(Y0, N, Tmax, F):
    '''de droite à gauche en prenant garde de ne pas dépasser'''

    T,  SY = EulerExplicite(Y0, Tmax/(N-1),  Tmax, F)
    if  TestConvergence(T,  SY[:,0]):
        t       = N-1
        w_lim   = SY[t,0]

        while t > 0 and abs(SY[t,0]-w_lim)<=abs(w_lim)/20:
            t = t - 1
        return t+1

    else:
        return -1

print(CalculT5(Y0, N, Tmax, f1))
```