CORRIGÉ DU DEVOIR À LA MAISON 9
Résolution d'une équation différentielle d'ordre 2 : oscillateur linéaire

Étude d'un oscillateur harmonique amorti : gestion du recul d'un canon (cf. TD MI2 – exercice 3)

1. Système : canon de masse M assimilé à son centre de gravité G

- Référentiel : terrestre supposé galiléen, base cartésienne
- Bilan des forces
  - Poids : $\vec{P}$: force conservative dérivant de $E_{P,pes} = cste$
  - Réaction normale du sol : $\overrightarrow{R_N}$ force non conservative telle que $W\left(\overrightarrow{R_N}\right) = 0$
  - Force de rappel élastique : $\vec{F} = -k_2\left(l - L_0\right)\vec{u}_{\text{sortant}} = -k_2\left(OG - L_0\right)\overrightarrow{u_x} = -k_2 x\overrightarrow{u_x}$: force conservative dérivant de $E_{P,élas} = \dfrac{1}{2}k_2 x^2$
  - Force de frottements fluides visqueux : $\overrightarrow{F_f} = -\lambda\vec{v} = -\lambda\dot{x}\overrightarrow{u_x}$: force non conservative
- Système non conservatif à un degré de liberté : x
- Énergie cinétique : $E_C = \dfrac{1}{2}M\dot{x}^2$
- Énergie mécanique : $E_m = E_C + E_{P,élas} + E_{P,pes} = \dfrac{1}{2}M\dot{x}^2 + \dfrac{1}{2}k_2x^2 + cste$
- Théorème de la puissance mécanique :

$$\frac{dE_m}{dt} = \mathcal{P}^{NC} = \mathcal{P}\left(\overrightarrow{R_N}\right) + \mathcal{P}\left(\overrightarrow{F_f}\right) = -\lambda\vec{v}\cdot\vec{v} = -\lambda\dot{x}^2$$

$$\frac{d}{dt}\left(\frac{1}{2}M\dot{x}^2\right) + \frac{d}{dt}\left(\frac{1}{2}k_2x^2\right) = -\lambda\dot{x}^2 \Leftrightarrow \frac{d}{d\dot{x}}\left(\frac{1}{2}M\dot{x}^2\right)\frac{d\dot{x}}{dt} + \frac{d}{dx}\left(\frac{1}{2}k_2x^2\right)\frac{dx}{dt} = -\lambda\dot{x}^2$$

$$M\dot{x}\ddot{x} + k_2x\dot{x} = -\lambda\dot{x}^2 \Leftrightarrow M\ddot{x} + k_2x = -\lambda\dot{x} \Leftrightarrow \ddot{x} + \frac{\lambda}{M}\dot{x} + \frac{k_2}{M}x = 0$$

$$\boxed{\frac{d^2x(t)}{dt^2} + 2\xi\omega_0\frac{dx(t)}{dt} + \omega_0^2x(t) = 0}$$

$$\text{avec } \boxed{\omega_0 = \sqrt{\frac{k_2}{M}}} \text{ et } 2\xi\omega_0 = \frac{\lambda}{M} \text{ soit } \boxed{\xi = \frac{\lambda}{2M\omega_0} \Leftrightarrow \xi = \frac{\lambda}{2\sqrt{Mk_2}}}$$

Résolution numérique de l'équation différentielle avec Python

```python
## Cellule 1 : Importation des bibliothèques utiles
import numpy as np
import matplotlib.pyplot as plt
import scipy.integrate as sci    # Pour utiliser la fonction odeint
from math import *
```

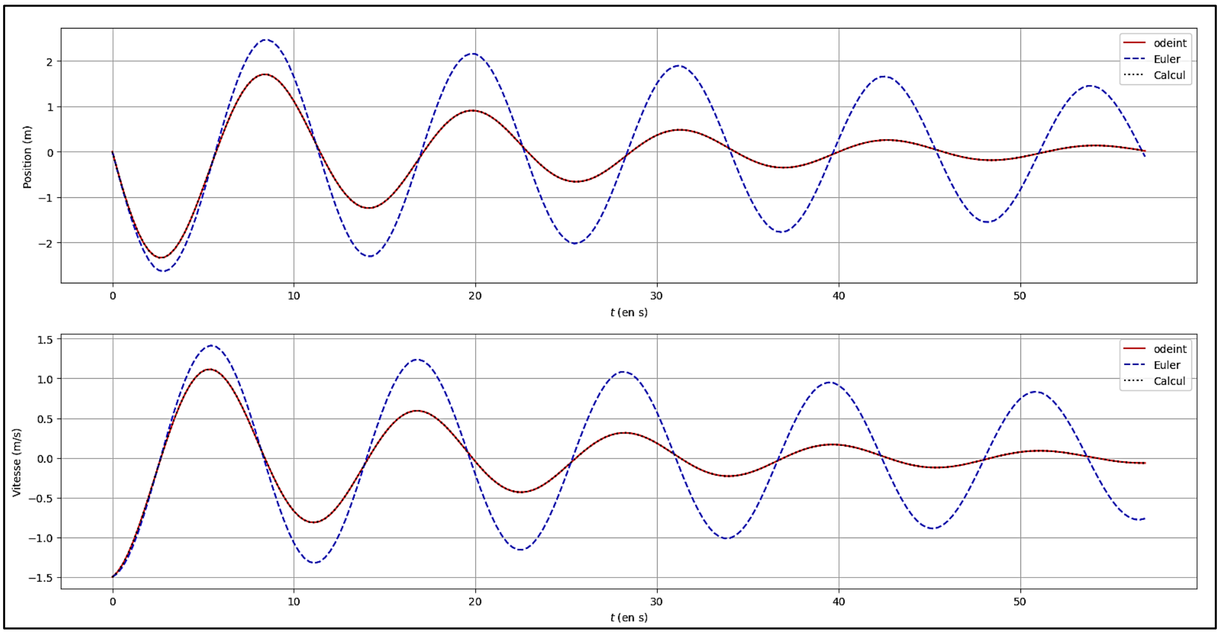
```python
## Cellule 2 :  Données du problème physique -> A COMPLETER

# Constantes physiques
ksi = 0.1                    # Coefficient d'amortissement
M = 800.                     # Masse du canon (kg)
m = 2.                       # Masse de l'obus (kg)
k2 = 244.                    # Constante de raideur du ressort (N/m)
v0 = 600.                    # Vitesse initiale de l'obus (m/s)
omega = sqrt(k2 / M)         # Expression de la pulsation propre (en rad.s-1)
T = 2*np.pi/omega            # Expression de la période propre (en s)

# Paramètres de la résolution numérique
t0, tf = 0, 5*T                  # bornes de l'intervalle de résolution
n = 200                          # nombre de points
dt = (tf-t0) / (n-1)             # expression du pas temporel pour la résolution numérique

# Conditions initiales
x0 = 0                    # Abscisse initiale du canon (m)
xprim0 = - m / M *v0      # Expression de la vitesse initiale du canon (m/s)
y0 = np.array([x0, xprim0]) # Initialisation du vecteur y=[x,x']

# Création de la variable temporelle
"""
numpy.linspace (tmin,tmax,N)
Renvoie un tableau de N points régulièrement espacés
entre tmin (inclus) et tmax (inclus)
"""
t = np.linspace(t0,tf,n)  # n points régulièrement espacés entre t0 (inclus) et tf (inclus)

# Définition de la fonction dérivée de y(t)
def derivee_y(y,t):
    return np.array([y[1], -2*ksi*omega*y[1] - omega**2*y[0]]) # expression à compléter
```

```python
## Cellule 3 : Résolution avec odeint et représentation graphique -> A COMPLETER
"""
scipy.integrate.odeint (f,y0,t)

Résout un système d'équations différentielles d'ordre 1
Paramètres :
    f : f(y,t) : fonction qui calcule la dérivée de y à l'instant t
    y0 : tableau ou vecteur : condition initiale
    t : tableau d'instants pour lesquels la résolution est réalisée
Renvoie :
    y : tableau ou vecteur avec les valeurs de y calculées pour chaque instant t
    (les valeurs initiales sont sur la 1ère ligne)
"""
y = sci.odeint(derivee_y,y0,t)  # Vecteur y obtenu avec odeint : à compléter

# Représentation graphique
plt.subplot(2,1,1)  # Subdivision de la fenêtre graphique en 2 fenêtres, l'une au-dessus de l'autre et tracé sur la fenêtre du haut
plt.plot(t,y[:,0],'r-',label='odeint')  # Graphe de l'abscisse x en fonction du temps en trait rouge : à compléter
plt.xlabel(r"$t$ (en s)")           # Nom de l'axe des abscisses
plt.ylabel("Position (m)")          # Nom de l'axe des ordonnées
plt.legend(loc = 'upper right')   # Affichage de la légende en haut à droite
plt.grid()  # Affichage de la grille
plt.show()  # Affichage de la fenêtre

plt.subplot(2,1,2) # Tracé sur la fenêtre du bas
plt.plot(t,y[:,1],'r-',label='odeint') # Graphe de la vitesse xprim en fonction du temps en trait rouge : à compléter
plt.xlabel(r"$t$ (en s)")               # Nom de l'axe des abscisses
plt.ylabel("Vitesse (m/s)")             # Nom de l'axe des ordonnées
plt.legend(loc = 'upper right')     # Affichage de la légende en haut à droite
plt.grid()  # Affichage de la grille
plt.show()  # Affichage de la fenêtre
```

```python
## Cellule 4 : Résolution avec Euler et représentation graphique -> A COMPLETER
def euler(F, y0, t, dt, n):
    """
Paramètres :
    F : fonction donnant y'
    y0 : condition initiale sur y
    t : tableaux des instants pour lesquels les calculs sont réalisés
    dt : pas de discrétisation utilisé pour la résolution
    n : nombre de points pour lesquels les calculs sont réalisés
Renvoie :
    y : vecteur contenant l'ensemble des valeurs approchées yk
    """
    y = np.zeros([n,2])            # initialisation du vecteur y : n lignes, 2 colonnes
    y[0] = y0                      # prise en compte des conditions initiales
```

```
    # Boucle permettant le calcul des yk par récurrence
    for k in range(0,n-1):          # k prend les valeurs de 0 à n-2
        y[k+1] = y[k] + F(y[k],t[k])*dt # relation de récurrence : à compléter
    return y

y_euler = euler(derivee_y, y0, t, dt, n) # Vecteur y_euler obtenu avec la méthode d'Euler : à compléter

# Représentation graphique
plt.subplot(2,1,1)
plt.plot(t,y_euler[:,0],'b--',label='Euler') # Graphe de l'abscisse x en fonction du temps en tirets bleus
plt.legend(loc = 'upper right')
plt.subplot(2,1,2)
plt.plot(t,y_euler[:,1],'b--',label='Euler') # Graphe de la vitesse xprim en fonction du temps en tirets bleus
plt.legend(loc = 'upper right')
```

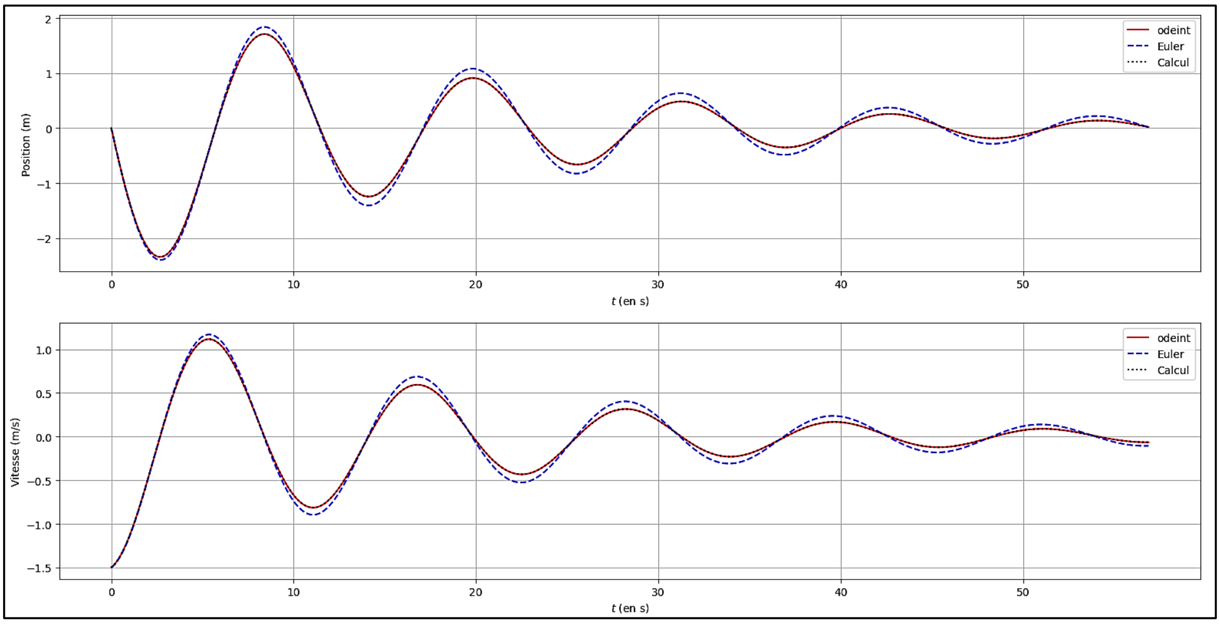
```
## Cellule 5 : Calcul analytique et représentation graphique -> A COMPLETER
wp = omega*sqrt(1-ksi**2) # Expression de la pseudo-pulsation (rad/s)
x_calc = xprim0 / wp * np.sin(wp*t) * np.exp(-ksi*omega*t) # Expression analytique de l'abscisse
xprim_calc = xprim0 *(np.cos(wp*t) - ksi*omega / wp * np.sin(wp*t)) * np.exp(-ksi*omega*t) # Expression analytique de la vitesse

# Représentation graphique
plt.subplot(2,1,1)
plt.plot(t,x_calc,'k:',label='Calcul') # Graphe de l'abscisse x en fonction du temps en pointillés noirs : à compléter
plt.legend(loc = 'upper right')
plt.subplot(2,1,2)
plt.plot(t,xprim_calc,'k:',label='Calcul') # Graphe de la vitesse xprim en fonction du temps en pointillés noirs : à compléter
plt.legend(loc = 'upper right')
```



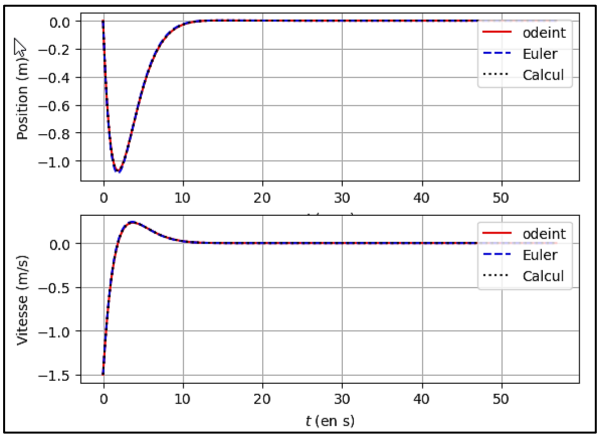
- <u>Commentaires</u> : Les graphes obtenus avec la méthode d'Euler diffèrent de ceux obtenus avec la fonction odeint, qui sont similaires à ceux obtenus avec le calcul analytique, ce qui montre la performance de l'algorithme de Runge-Kutta.
- Distance de recul $\boxed{d_m = 2{,}3\ \text{m}}$ à l'instant $\boxed{t_m = 2{,}6\ \text{s}}$

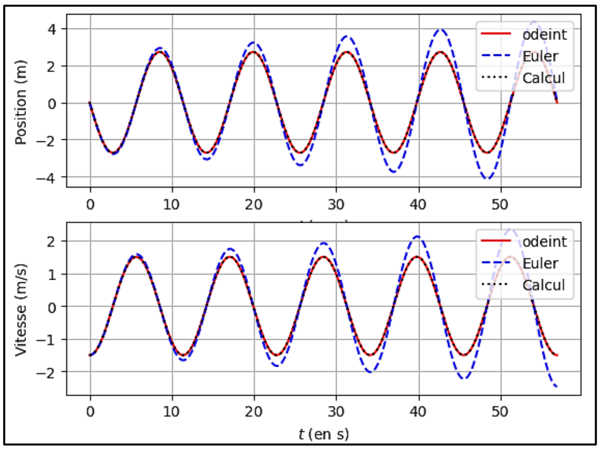
- **<u>Influence du nombre de points</u>**

- <u>Commentaires</u> : Pour $n = 1000$ points, les performances de la méthode d'Euler explicite s'améliorent : les courbes se rapprochent de celles obtenues analytiquement. <u>L'augmentation du nombre de points et donc la diminution du pas de résolution permettent de réduire l'erreur commise</u>.



❖ Influence du coefficient d'amortissement





- Commentaires : Pour $n = 1000$ points et $\xi = 0{,}9$ (figure de gauche) : la méthode d'Euler donne des résultats convenables. Pour $\xi = 0$ (figure de droite), la solution obtenue avec la méthode d'Euler diverge ! Dans tous les cas, l'algorithme de Runge-Kutta à pas variable de la fonction odeint fournit une solution numérique identique à la solution analytique.
- Dans le cas du régime critique, i.e. pour $\xi = 1$, la méthode d'Euler fournit une solution très proche de celle fournie par l'algorithme de Runge-Kutta.
  La distance de recul est $\boxed{d'_m = 1{,}0\ \text{m}}$ à l'instant $\boxed{t'_m = 1{,}8\ \text{s}}$: on retrouve les valeurs calculées en TD.