Physique : DS11

PARTIE A – PLATEFORME EN MER (CCP – PC – 2019)

Q1) En appliquant le PFD : $m\vec{a} = \vec{F}_{exc} + \vec{P} + \underbrace{\vec{F}_d + \vec{F}_N}_{\vec{F}_{TOT}} + \vec{F}_k$

le mouvement se fait suivant (Ox) d'où $m\ddot{y} = 0 \Leftrightarrow F_d + F_N = 0$

$$\Rightarrow \boxed{m\vec{a} = \vec{F}_{exc} + \vec{P} + \vec{F}_k}$$

Q2) On est dans le cas sans amortissement et sans excitation d'où $\vec{F}_k = -k(x-x_0)\vec{u}_x$

avec $x_0 = 0$ d'après l'énoncé : $\Rightarrow m\ddot{x} = -kx$

$$\Rightarrow \boxed{m\ddot{x} + kx = 0}$$

Q3) On reconnaît une E.D d'un oscillateur harmonique

$$\Rightarrow x = A_0 \sin(\omega_0 t) + B_0 \cos(\omega_0 t) \text{ où } \boxed{\omega_0 = \sqrt{k/m}}$$

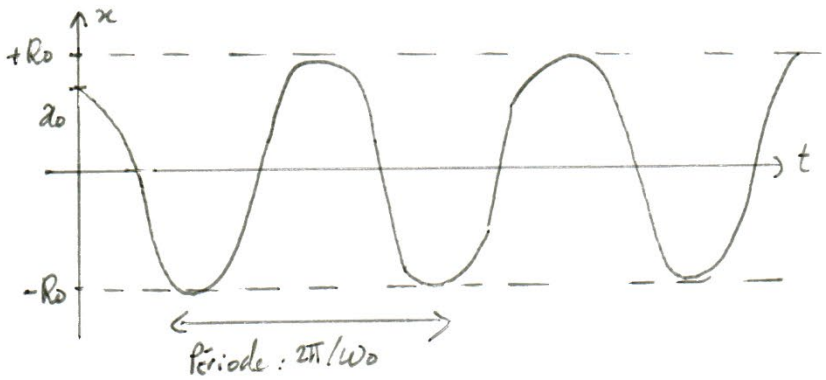
or $\begin{cases} x(0) = x_0 = B \\ \dot{x}(0) = \dot{x}_0 = A_0\omega_0 \end{cases}$ $\Rightarrow \boxed{x(t) = x_0 \cos(\omega_0 t) + \frac{\dot{x}_0}{\omega_0} \sin(\omega_0 t)}$

Q4) On pose $x(t) = R_0 \cos(\omega_0 t - \phi_0)$

$$= R_0 \left[\cos(\omega_0 t)\cos(\phi_0) + \sin(\omega_0 t)\sin(\phi_0)\right]$$

Par analogie : $\begin{cases} R_0 \cos\phi_0 = x_0 \\ R_0 \sin\phi_0 = \dot{x}_0/\omega_0 \end{cases} \Rightarrow \boxed{\begin{cases} R_0^2 = x_0^2 + \left(\frac{\dot{x}_0}{\omega_0}\right)^2 \\ \tan\phi_0 = \frac{\dot{x}_0}{x_0\omega_0} \end{cases}}$

Q5)



Q6) Calculons $E(t)$:

. Par définition : $E = K + U$

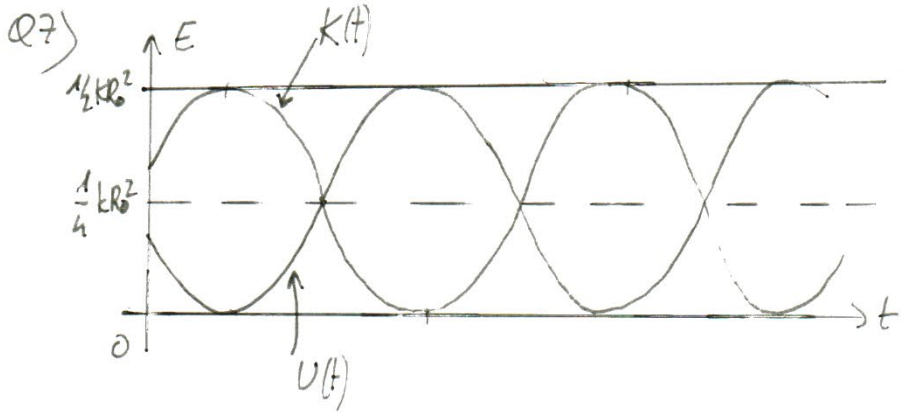
$$= \frac{1}{2} m v^2 + \frac{1}{2} k x^2$$

$$= \frac{1}{2} k R_0^2 \cos^2(\omega_0 t - \phi_0) + \frac{1}{2} \underbrace{m \omega_0^2}_{k} \sin^2(\omega_0 t - \phi_0) \cdot R_0^2$$

$$= \frac{1}{2} k R_0^2$$

. On remarque que l'énergie se conserve car il n'y a pas de forces non conservatives.

Q7)



Q8) On rajoute la force de frottement : $m\ddot{x} = -kx - \gamma \dot{x}$

$$\Leftrightarrow \ddot{x} + \frac{\gamma}{m} \dot{x} + \frac{k}{m} x = 0$$

$$\Leftrightarrow \ddot{x} + 2\xi\omega_0 \dot{x} + \omega_0^2 x = 0 \quad \text{où} \quad \begin{cases} \omega_0^2 = k/m \\ 2\xi\omega_0 = \gamma/m \end{cases}$$

$$\Rightarrow \xi = \frac{\gamma}{2m\omega_0} \Rightarrow \xi = \frac{\gamma}{2\sqrt{km}}$$

Q9) Soit la solution :

$$x(t) = e^{-\zeta \omega_0 t}\left[A_d \cos(\omega_d t) + B_d \sin(\omega_d t)\right]$$

avec $\begin{cases} x(0) = x_0 = A_d \\ \dot{x}(0) = \dot{x}_0 = \left[\omega_d B_d - \zeta \omega_0 A_d\right] \end{cases}$

D'où : $B_d = \dfrac{\dot{x}_0 + \zeta \omega_0 A_d}{\omega_d} = \dfrac{\dot{x}_0}{\omega_d} + \zeta x_0 \dfrac{\omega_0}{\omega_d}$

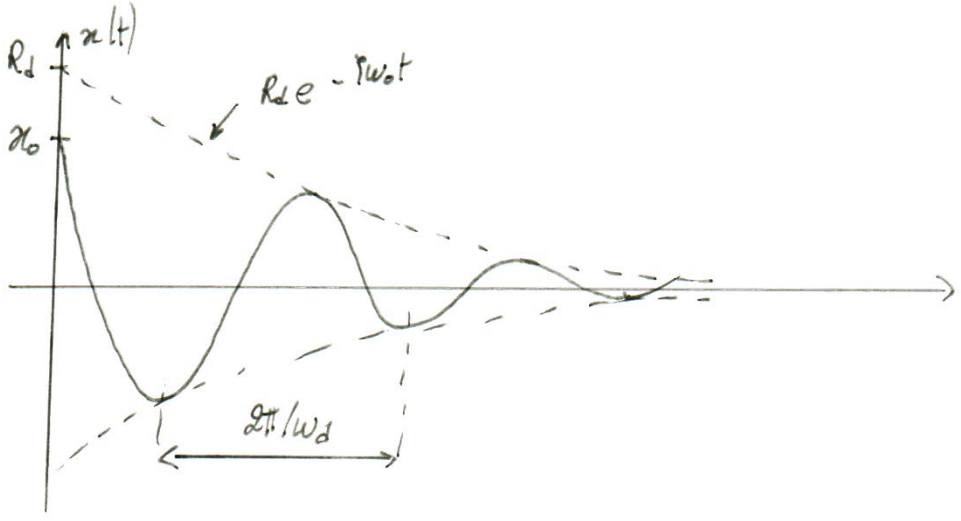
$$\Rightarrow \boxed{\begin{cases} A_d = x_0 \\ B_d = \dfrac{\dot{x}_0 + \zeta x_0 \omega_0}{\omega_d} \end{cases}}$$

Q10) Soit $x(t) = R_d e^{-\zeta \omega_0 t} \cos(\omega_d t - \phi_d)$

$= R_d e^{-\zeta \omega_0 t}\left[\cos(\omega_d t)\cos\phi_d + \sin(\omega_d t)\sin\phi_d\right]$

Par analogie : $\begin{cases} A_d = R_d \cos\phi_d \\ B_d = R_d \sin\phi_d \end{cases} \Rightarrow \boxed{\begin{cases} R_d^2 = A_d^2 + B_d^2 \\ \tan\phi_d = \dfrac{B_d}{A_d} \end{cases}}$

Q11)



Q12) Calculons $E(t)$

$$\text{Soit } E(t) = K + U$$

$$= \frac{1}{2}mv^2 + \frac{1}{2}kx^2$$

$$= \frac{1}{2}m\left[R_d^2 e^{-2\xi\omega_0 t}\left(-\xi\omega_0\cos(\omega_d t - \phi_d) - \omega_d \sin(\omega_d t - \phi_d)\right)\right]^2$$

$$+ \frac{1}{2}\underbrace{\frac{k}{m}}_{\omega_0^2} R_d^2 e^{-2\xi\omega_0 t}\cos^2(\omega_d t - \phi_d)$$

$$= \frac{1}{2}m R_d^2 e^{-2\xi\omega_0 t}\left[\left((\xi\omega_0)^2 + \omega_0^2\right)\cos^2(\omega_d t - \phi_d) + \underbrace{\omega_d^2}_{=\omega_0^2 - \xi^2\omega_0^2}\sin^2(\omega_d t - \phi_d)\right.$$

$$\left. + 2\xi\omega_0\omega_d \sin(\omega_d t - \phi_d)\cos(\omega_d t - \phi_d)\right]$$

$$= \frac{1}{2}m R_d^2 e^{-2\xi\omega_0 t}\left[\omega_0^2 + \xi^2\omega_0^2\left(\cos^2(\omega_d t - \phi_d) - \sin^2(\omega_d t - \phi_d)\right)\right.$$

$$\left. + 2\xi\omega_0^2\sqrt{1-\xi^2}\cdot\frac{1}{2}\cdot\sin(2\omega_d t - 2\phi_d)\right]$$

$$\Rightarrow E(t) = \frac{1}{2}k R_d^2 e^{-2\xi\omega_0 t}\left[1 + \xi^2\cos(2\omega_d t - 2\phi_d) + \xi\sqrt{1-\xi^2}\sin(2\omega_d t - 2\phi_d)\right]$$

- Si $\xi = 0$: $E(t) = \frac{1}{2}kR_d^2$ on retrouve le cas sans frottement.
- Si $\xi = 1$: $E(t) = \frac{1}{2}R_d^2 e^{-2\omega_0 t}\left[1 + \cos(\underbrace{2\omega_d}_{=0} - 2\phi_d)\right]$

$$= \frac{1}{2}R_d^2 e^{-2\omega_0 t}$$, l'énergie décroît de façon exponentielle.

Q13) D'après le théorème de l'E_m : $\frac{dE_m}{dt} = \mathcal{P}_{nc}$.

$$\Leftrightarrow \frac{dE_m}{dt} = -\gamma\vec{v}\cdot\vec{v}$$

$$\Leftrightarrow \frac{dE_m}{dt} = -\gamma v^2 < 0 \Rightarrow$$ l'énergie décroît car les forces de frottement s'opposent au mouvement

Q14) Calculons $\ln\left(\frac{x_1}{x_2}\right)$:

Or $\begin{cases} x_1 = R_d e^{-\gamma \omega_0 t_1} \cos(\omega_d t_1 - \phi_d) \\ x_2 = R_d e^{-\gamma \omega_0 t_2} \cos(\omega_d t_2 - \phi_d) \end{cases}$

D'où $\ln\left(\frac{x_1}{x_2}\right) = \ln\left[e^{-\gamma \omega_0 (t_1 - t_2)}\right]$ où $t_2 = t_1 + T_d$.

$\Rightarrow \ln\left(\frac{x_1}{x_2}\right) = \gamma \omega_0 T_d$. avec $T_d = \frac{2\pi}{\omega_d} = \frac{2\pi}{\omega_0 \sqrt{1-\gamma^2}}$

$\Rightarrow \ln\left(\frac{x_1}{x_2}\right) = \frac{2\gamma}{\sqrt{1-\gamma^2}}$

Si $\gamma \ll 1 \Rightarrow \underline{\ln\left(\frac{x_1}{x_2}\right) \simeq 2\pi\gamma}$

Q15) • D'après la figure proposée : $\begin{cases} x_2 = 0{,}010\,661\ m \\ x_1 = 0{,}014\,602\ m \end{cases} \Rightarrow \underline{\gamma \simeq 0{,}0501}$

or $\gamma \ll 1 \Rightarrow \omega_d \simeq \omega_0 = \frac{2\pi}{t_2 - t_1} = \sqrt{\frac{k}{m}}$.

D'où $k = \frac{4\pi^2}{(t_2 - t_1)^2} \cdot m = \underline{2{,}71 \cdot 10^5\ kg.s^{-2}}$

• Pour finir : $\gamma = 2\gamma\sqrt{km}$ d'après (Q8)

$\Rightarrow \underline{\gamma = 1{,}73 \cdot 10^4\ kg.s^{-1}}$

• Lorsque γ augmente, la décroissance sera plus rapide.

Q16) On rajoute l'excitation

Cette fois on peut écrire sur Ox : $m\ddot{x} = -kx - \gamma\dot{x} + F_0\cos(\omega t)$

$$\Rightarrow \boxed{\ddot{x} + 2\xi\omega_0\dot{x} + \omega_0^2 x = \frac{F_0}{m}\cos\omega t}$$

Q17) On utilise la notation complexe d'où :

$$-\omega^2\underline{X} + 2\xi\omega_0(j\omega)\underline{X} + \omega_0^2\underline{X} = \frac{F_0}{m}.$$

$$\Rightarrow \underline{X} = \frac{F_0/m}{(\omega_0^2-\omega^2) + j.2\xi\omega_0\omega}$$

D'où : $\begin{cases} X = |\underline{X}| = \dfrac{F_0/m}{\sqrt{(\omega_0^2-\omega^2)^2 + 4\xi^2\omega_0^2\omega^2}} \\ \tan\phi = -\dfrac{2\xi\omega_0\omega}{\omega_0^2-\omega^2} \quad \text{(erreur de signe dans l'énoncé)} \end{cases}$

Q18) Calculons M :

$$M = \frac{X}{F_0/k} = \frac{k/m}{\sqrt{\dots}} = \frac{\omega_0^2/\omega_0^2}{\sqrt{\left(1-\left(\frac{\omega}{\omega_0}\right)^2\right)^2 + 4\xi^2\left(\frac{\omega}{\omega_0}\right)^2}}$$

$$\Rightarrow \boxed{M = \frac{1}{\underbrace{\sqrt{(1-r^2)^2 + 4\xi^2 r^2}}_{A}}}$$

Q19) Cherchons la valeur de r qui rend M maximal en dérivant la qté A

$$\frac{dA}{dr} = 0 \Leftrightarrow -2r(2).(1-r^2) + 8\xi^2 r = 0 \Leftrightarrow (1-r^2) - 2\xi^2 = 0$$

$$\Leftrightarrow \boxed{r^2 = 1-2\xi^2} \quad \Rightarrow \boxed{\omega_r = \omega_0\sqrt{1-2\xi^2}}$$

Q20) Notre plateforme a une période de résonance de 4s ainsi $T \simeq 2T_0$. On est suffisamment loin pour ne pas avoir le développement d'oscillations de grandes amplitudes.

Partie II – Modélisation : codage

Q21.
Il ne faut pas oublier de prendre la partie entière car N est un entier. Le « +1 » est nécessaire car il y a un intervalle de moins que de valeurs de N dans une liste.

```
N = int(tmax/dt+1)
```

Q22.
À l'aide de l'instruction linspace de numpy :

```
t = np.linspace(0, tmax, N)
```

On peut aussi créer une liste à l'aide d'une boucle, puis la transformer (ou non) en tableau numpy :

```
t = []
for i in range(N):
    t.append(i*dt)
t = np.array(t)
```

En revanche ce code est plus long à l'écriture comme à l'exécution...

Q23.
On peut écrire directement $\boxed{x_{n+1} \approx x_n + v_n \cdot \Delta t}$. Pour l'expression de v_{n+1}, il suffit de remplacer a_n par la relation obtenue directement à partir de (11) : $a = -2\zeta\omega_0 v - \omega_0^2 x + F(t)/m$, soit :

$$\boxed{v_{n+1} \approx v_n + \left(-2\zeta\omega_0 v_n - \omega_0^2 x_n + \frac{F_n}{m}\right) \cdot \Delta t}$$

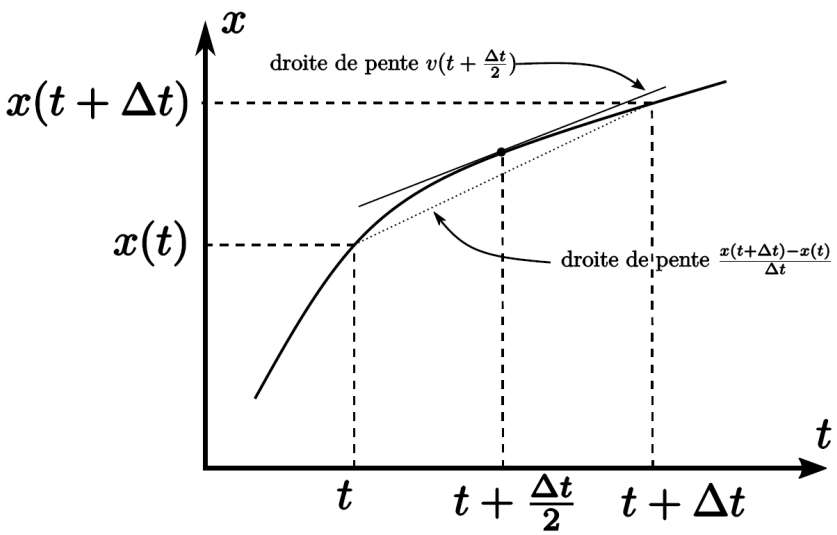
Q24.
On suppose que l'énoncé, lorsqu'il demande de « calculer toutes les valeurs de x[i+1] et v[i+1] » nous demande de remplir les deux tableaux x et v.
On suppose que le tableau F a déjà été défini et rempli. Les valeurs de N, k, dt, m, x0, v0, om0 et zeta sont aussi supposées connues.
L'énoncé utilisant le terme « tableau », nous allons utiliser des structures de type array de numpy (structures plus pratiques à manipuler par la suite). On pouvait cependant utiliser des listes et l'instruction « append » pour compléter ces listes au fur et à mesure.

```
#remplissage de la première case de chaque tableau
x[0] = x0
v[0] = v0
E[0] = 1/2*(k*x0**2+m*v0**2)

for i in range(1, N): #boucle pour calculer les valeurs suivantes
    x[i] = x[i-1]+v[i-1]*dt #utilisation des relations de Q23
    v[i] = v[i-1]+(-2*om0*zeta*v[i-1]-om0**2*x[i-1]*F[i-1]/m)*dt
    E[i] = 1/2*(k*x[i]**2+m*v[i]**2)
```

Q25.

En partant de $v(t+\Delta t/2) \approx \dfrac{x(t+\Delta t)-x(t)}{\Delta t}$ (calcul de la dérivée « au milieu » de l'intervalle comme explicité sur la figure ci-après), en se plaçant au temps $t = t_n = n\cdot\Delta t$, soit $t+\Delta t = t_{n+1} = (n+1)\cdot\Delta t$ ainsi que $t+\Delta t/2 = t_{n+1/2} = (n+1/2)\cdot\Delta t$, on peut écrire : $v_{n+1/2} \approx \dfrac{x_{n+1}-x_n}{\Delta t}$, soit en effet : $\boxed{x_{n+1} \approx x_n + v_{n+1/2}\cdot\Delta t}$



Pour l'autre relation demandée les calculs sont plus compliqués : il faut comprendre que comme les valeurs des x sont prises aux temps entiers et que les valeurs des v sont prises aux temps demi-entiers, les valeurs des a sont elles évaluées aux temps entiers, comme les valeurs des x (il y a une quinconce de répartition d'une valeur et de sa dérivée temporelle première).

On commence en écrivant $a(t) \approx \dfrac{v(t+\Delta t/2)-v(t-\Delta t/2)}{\Delta t}$. En explicitant cette relation pour $t = t_n = n\cdot\Delta t$ et donc $t+\Delta t/2 = t_{n+1/2} = (n+1/2)\cdot\Delta t$ et $t-\Delta t/2 = t_{n-1/2} = (n-1/2)\cdot\Delta t$:

$$v_{n+1/2} \approx v_{n-1/2} + a_n\cdot\Delta t$$

Or $a = -2\zeta\omega_0 v - \omega_0^2 x + F(t)/m$, soit $a_n \approx -2\zeta\omega_0\dfrac{v_{n+1/2}+v_{n-1/2}}{2} - \omega_0^2 x_n + F_n/m$ (on prend la valeur moyenne de v entre $t+\Delta t/2$ et $t-\Delta t/2$ pour l'évaluer au mieux en t).

Remplaçons a_n par cette expression dans la relation obtenue juste avant :

$v_{n+1/2} \approx v_{n-1/2} - \zeta\omega_0(v_{n+1/2}+v_{n-1/2})\Delta t - \omega_0^2 x_n\Delta t + F_n/m\Delta t$, soit :

$v_{n+1/2}(1+\zeta\omega_0\Delta t) \approx v_{n-1/2}(1-\zeta\omega_0\Delta t) - \omega_0^2 x_n\Delta t + F_n/m\Delta t$, soit finalement :

$$\boxed{v_{n+1/2} \approx \frac{1-\zeta\omega_0\Delta t}{1+\zeta\omega_0\Delta t}v_{n-1/2} - \frac{\omega_0^2\Delta t}{1+\zeta\omega_0\Delta t}x_n + \frac{F_n}{m(1+\zeta\omega_0\Delta t)}\Delta t}$$

Q26.

Solution : Pour évaluer `E[i+1]`, il faut la position et la vitesse à l'instant $(i+1)\Delta t$. La vitesse n'étant pas calculée sur les temps entiers, il faut l'évaluer par sa moyenne entre les valeurs décalées de $\Delta t/2$ avant et après, ce qui demande d'avoir accès à la valeur qui sera stockée dans `v[i+2]` et qui ne sera évaluée qu'à l'itération suivante de la boucle. Calculer `E[i]` par contre est réalisable dans la boucle qui puisqu'on aura accès à toutes les valeurs de x et de v utiles.

Q27.

Solution : $E_n = \frac{1}{2}kx_n^2 + \frac{1}{2}mv_n^2 = \frac{1}{2}kx_n^2 + \frac{1}{2}m\left(\frac{v_{n-1/2}+v_{n+1/2}}{2}\right)^2$ ce qui ce code en :

```
E[i]=0.5*k*x[i]**2+0.125*m*(v[i]+v[i+1])**2
```

Q28

Solution :

```
>>> ### les facteurs de proportionnalité
>>> fac1=1-zeta*om0*dt
>>> fac2=1/(1+zeta*om0*dt)

>>> ### initialisation: les tableaux existent déjà, je réinitialise juste les
 ↪  valeurs initiales

>>> x[0]=x0
>>> v[0]=v0

>>> ### la boucle demandée:

>>> for i in range(N-1):
...     v[i+1]=-fac2*om0**2*dt*x[i]+fac1*fac2*v[i]+F[i]*fac2*dt/m
...     x[i+1]=x[i]+v[i+1]*dt
```

Q29.

Solution :

```
>>> for i in range(N-1):
...     v[i+1]=-fac2*om0**2*dt*x[i]+fac1*fac2*v[i]+F[i]*fac2*dt/m
...     x[i+1]=x[i]+v[i+1]*dt
...     E[i]=0.5*k*x[i]**2+0.125*m*(v[i]+v[i+1])**2
...
>>> ### Il reste le dernier élément du tableau E à remplir, pour lequel il faut un
 ↪  calcul de vitesse supplémentaire:

>>> vsup=-fac2*om0**2*dt*x[N-1]+fac1*fac2*v[N-1]+F[N-1]*fac2*dt/m
>>> E[N-1]=0.5*k*x[N-1]**2+0.125*m*(v[N-1]+vsup)**2
```

Q30.

Le code à écrire est très court car il reprend en très grande partie les lignes de code écrites dans les questions **Q24**, **Q28** et **Q29**

Solution :

```
>>> def integration(F):
...     ''' À partir d'une excitation donnée par un tableau numpy, retourne les
 ↪  tableaux x, v et E, initialisés à 0.'''
...     global algo # 0 pour Euler, 1 pour Leapfrop
...     ### Initialisation. Le calcul de E[0] est omis car tout est au repos
...     x=np.zeros(len(F))
...     v=np.zeros(len(F))
...     E=np.zeros(len(F))
...     ## Valeurs initiales
...     x[0],v[0]=x0,v0
...     ### Boucles
...     if algo==0:
...         # Euler
...         for i in range(N-1):
...             x[i+1]=x[i]+v[i]*dt
...             v[i+1]=v[i]+(F[i]/m-2*zeta*om0*v[i]-x[i]*om0**2)*dt
...             E[i+1]=0.5*k*x[i+1]**2+0.5*m*v[i+1]**2
...     elif algo==1:
...         # Leapfrog
...         for i in range(N-1):
...             v[i+1]=-fac2*om0**2*dt*x[i]+fac1*fac2*v[i]+F[i]*fac2*dt/m
...             x[i+1]=x[i]+v[i+1]*dt
...             E[i]=0.5*k*x[i]**2+0.125*m*(v[i]+v[i+1])**2
...         # dernier élément du tableau E
...         vsup=-fac2*om0**2*dt*x[N-1]+fac1*fac2*v[N-1]+F[N-1]*fac2*dt/m
...         E[N-1]=0.5*k*x[N-1]**2+0.125*m*(v[N-1]+vsup)**2
...     else:
...         print("Erreur: pas d'algo reconnu!")

...     return x,v,E
```

Q31.
L'écriture est immédiate :

```
def force(f, t, w):
  return f*np.cos(w*t)
```

Q32 —Écrire une fonction `force_exc()` qui complète et retourne le tableau `F[]` des forces d'excitation en fonction du booléen `exc` défini globalement valant `True` si une excitation est appliquée au système, `False` sinon.

Dans le cas où `exc==True`, on appellera la fonction force définie précédemment en question 31. Dans le cas où `exc==False`, on prendra alors : `F[i]=0`, ∀`i`.

Solution : C'est étrange de ne pas passer les caractéristiques de la force en argument, mais les arguments sont toujours spécifiés dans ce sujet.Je considère que l'amplitude de la force `F0` et sa pulsation `om` sont définis par ailleurs avant l'appel et que Python ira les chercher en les considérant comme des variables globales.

```
>>> def force_exc():
...     ''' Retourne le tableau décrivant la force d'excitation'''
...     global exc
...     F=np.zeros(N)
...     if exc:
...         for i in range(len(F)):
...             F[i]=force(F0,i*dt,om)
...     return F
...
```

Q33 —Donner alors les lignes de code permettant de réaliser la simulation numérique à partir des fonctions précédentes.

```
>>> ### on reprend la définition des paramètres du problème:
>>> zeta = 5e-2
>>> k = 2.7e5
>>> om0 = 2*np.pi/4 #T0=4s
>>> om = 2*np.pi/8 #Tv=8s pour la période des vagues
>>> F0 = 5e3 #N, au pif, mais tel que x soit proche des 2cm en statique d'après la
 ↪  valeur de k et celle de x0...
>>> m = 110e3 #110 tonnes
>>> x0 = 0.02 #m
>>> v0 = 0 #m.s^-1

>>> tmax=10 #s d'après partie C
>>> dt=0.05 #s d'après le tableau fourni

>>> N=int(tmax/dt + 1)

>>> algo=1
>>> exc=0

>>> x,v,E=integration(force_exc())
```

Q34.

On suppose que d est un tableau numpy des données numériques et que dref est un tableau numpy des données analytiques.

On utilise la fonction np.max fournie dans l'énoncé :

```
def ema(d, dref):
  return np.max(np.abs(d-dref))
```

Q35.
L'indice maximal à considérer est N-1 puisque dans un tableau de N éléments, les indices varient de 0 à N-1.
Dans la question **Q29** nous avons fait en sorte que E contienne N éléments : il n'y a donc pas lieu de modifier la fonction ema car dans tous les cas on travaille avec des tableaux de même longueur égale à N.
En l'absence de cette précaution, le tableau E ne comporterait que N-1 éléments et il faudrait se restreindre à l'indice maximal N-2 pour le cas de E pour l'algorithme Leapfrog.

On peut proposer la fonction ema modifiée suivante qui permet de travailler avec des tableaux de longueurs différentes :

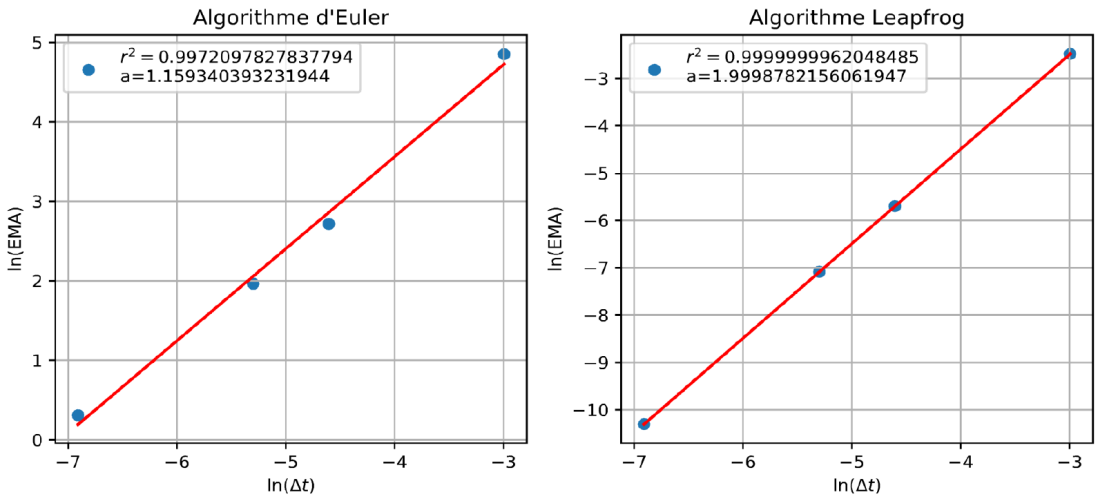
```
def ema_mod(d, dref):
    n = min(len(d), len(dref)) #on récupère la longueur du tableau le plus petit
    return ema(d[:n], dref[:n]) #on utilise la fonction ema sur les n premiers éléments des tableaux
```

Partie III – Modélisation : analyse des résultats d'un cas simple

Q36.
Il faut que $\Delta t \ll T_0$, avec $T_0 = 4\,\text{s}$.

Q37.
Dans les deux cas on cherche à déterminer a tel que $\text{EMA} = k(\Delta t)^a$, soit encore $\ln(\text{EMA}) = \ln k + a\ln(\Delta t)$. Il suffit donc de tracer (ou de faire une régression linéaire sur) $\ln(\text{EMA}) = f(\ln(\Delta t))$, la pente de la droite ainsi obtenue sera égale à l'ordre a cherché.

Dans le cas de l'algorithme Leapfrog, l'erreur est clairement d'ordre 2 par rapport à Δt alors que dans le cas de l'algorithme d'Euler celle-ci est plutôt voisine de l'ordre 1.



Q38.
$E_{n+1} = \frac{1}{2}kx_{n+1}^2 + \frac{1}{2}mv_{n+1}^2$, avec $x_{n+1} = x_n + v_n\Delta t$ et $v_{n+1} = v_n - \omega_0^2 x_n \Delta t$ dans le cas où il n'y a ni excitation ni amortissement.
De là, en développant tout :

$$E_{n+1} = \underbrace{\frac{1}{2}mv_n^2 + \frac{1}{2}kx_n^2}_{E_n} + \cancel{kx_n v_n \Delta t} - \underbrace{m\omega_0^2}_{k}\cancel{v_n x_n \Delta t} + \frac{1}{2}m\omega_0^4 x_n^2(\Delta t)^2 + \frac{1}{2}kv_n^2(\Delta t)^2$$

Soit finalement : $\boxed{E_{n+1} = E_n + \frac{1}{2}\left(m\omega_0^4 x_n^2 + k v_n^2\right)(\Delta t)^2}$. On en déduit que $E_{n+1} > E_n$: l'énergie mécanique calculée ne fait qu'augmenter proportionnellement à $(\Delta t)^2$ à chaque étape alors que dans ce cas l'énergie mécanique est censée se conserver.
Lors de l'utilisation de l'algorithme d'Euler entre $t = 0$ et $t = t_{\max}$, cette erreur va se cumuler $N - 1$ fois (il y a N points au total dont l'instant initial déjà connu). Or $t_{\max} = (N-1) \cdot \Delta t$, soit $N - 1 = \frac{t_{\max}}{\Delta t}$.
L'erreur cumulée sera donc proportionnelle à $(\Delta t)^2/\Delta t$ pour une valeur de $t_{\max}$ donnée : On retrouve bien une erreur globale sur E d'ordre 1 par rapport à Δt comme déduit des données du **tableau 2**.

Q39.
Dans le cas de l'oscillateur harmonique non amorti, l'équation différentielle est réversible : le fait que la seule dérivée par rapport au temps soit une dérivée seconde permet de remplacer t par $-t$ sans modifier l'équation (comme dans le cas de l'équation de d'Alembert par exemple, et contrairement au cas d'une équation de diffusion dont la présence d'une dérivée première par rapport au temps traduit un phénomène non réversible).

Q40.
Dans le cas de l'algorithme d'Euler : $x_{n+1} = x_n + v_n \cdot \Delta t$ et $\overline{x}_n = x_{n+1} + v_{n+1} \cdot (-\Delta t)$, soit :

$$\boxed{\overline{x}_n = x_n + (v_n - v_{n+1})\Delta t}$$

On en conclut directement que l'algorithme n'est pas réversible dans le temps puisque $v_n \neq v_{n+1} \Rightarrow \overline{x}_n \neq x_n$.

Dans le cas de l'algorithme Leapfrog, $x_{n+1} = x_n + v_{n+1/2}\Delta t$ et $\overline{x}_n = x_{n+1} + v_{n+1-1/2}(-\Delta t)$: la vitesse utilisée pour les deux calculs est la vitesse évaluée au point milieu entre t et $t + \Delta t$ (entre les points n et $n + 1$) : c'est donc la même dans les deux relations. On en déduit que $\overline{x}_n = x_n$.
Pour conclure quant à la réversibilité dans le temps ou non de cet algorithme il faut maintenant vérifier si $\overline{v}_{n-1/2} = v_{n-1/2}$ ou non. C'est l'objet de la question suivante.

Q41.
Ce n'est pas la peine de faire l'étude pour l'algorithme d'Euler puisque nous avons déjà conclu quant à sa non réversibilité dans le temps.
Dans le cas de l'algorithme Leapfrog, il faut vérifier si $\overline{v}_{n-1/2} = v_{n-1/2}$ ou non (et non pas si $\overline{v}_n = v_n$ comme demandé dans l'énoncé !). D'après la question **Q25**, Lorsqu'il n'y a ni excitation externe, ni amortissement ($\zeta = 0$ et $F_n = 0$), on a $v_{n+1/2} = v_{n-1/2} - x_n\omega_0^2\Delta t$ ainsi que $\overline{v}_{n-1/2} = v_{n+1/2} - x_n\omega_0^2 \cdot (-\Delta t)$, les vitesses étant évaluées au instants $t - \Delta t/2$ et $t + \Delta t/2$ (c'est à dire pour les points $n - 1/2$ et $n + 1/2$) et x étant pris au milieu, c'est à dire à l'instant t (donc au point n).
On en déduit que $\overline{v}_{n-1/2} = v_{n-1/2}$ et que l'algorithme Leapfrog est réversible dans le temps.

Q42.
Dans le cas de l'algorithme d'Euler, l'erreur augmente en moyenne avec le temps même si elle s'annule périodiquement. Il faudrait vérifier comment évolue l'erreur sur la vitesse mais il est très probable que celle-ci soit maximale localement quand celle sur x est nulle et inversement.

Dans le cas de l'algorithme Leapfrog, l'erreur n'augmente pas en moyenne et reste inférieure à 1×10^{-4}. Il n'y a pas de dérive contrairement au cas de l'algorithme d'Euler.

Q43.
L'algorithme d'Euler est plus facile à implémenter, comporte moins de calculs machine, et reste adaptable facilement à d'autres équations différentielles. En revanche il induit une erreur plus importante que pour Leapfrog et qui, de plus, augmente régulièrement avec le temps (dérive). Il est aussi plus sensible à Δt, dont la valeur sera choisie en général plus grande pour des simulations longues afin de ne pas multiplier exagérément le nombre de points à calculer et à stocker. Pour toutes ces raisons il n'est pas souhaitable pour des simulations de plusieurs heures.

L'algorithme Leapfrog est plus compliqué à implémenter et demande plus de calculs par la machine. En revanche il est souhaitable pour des simulations de grande durée car il n'induit pas de dérive.

--- FIN ---