Méthode d'Euler - correction

1 Equation linéarisée

1. La relation $x_{i+1} = x_i + v_i\,\varepsilon$ vient de $v = \frac{dx}{dt}$. Physiquement, il s'agit de la relation entre vitesse et position. L'autre relation, $v_{i+1} = v_i - \omega_0^2 x_i\,\varepsilon$, vient de $\frac{dv}{dt} = -\omega_0^2 x$, ce qui correspond à l'équation différentielle puisque $\frac{dv}{dt} = \frac{d^2x}{dt^2}$. Physiquement, cela combine la relation entre la vitesse et l'accélération $a = \frac{dv}{dt}$, la deuxième loi de Newton $ma = f$ et l'expression d'une force de rappel élastique $f = -kx$ (tout cela avec $\omega_0^2 = \frac{k}{m}$). Enfin, ϵ représente le pas.
2. Par rapport à l'énoncé, on ajoute dans les arguments les conditions initiales (position et vitesse).

```
def solve(w0,h,a,b,x0,v0) :

    x,v = x0,v0
    t = a
    positions = [x]
    vitesses = [v]
    temps = [t]

    while t < b-h :
        tmp = x
        x = x+v*h
        v = v-(w0**2)*tmp*h
        t = t+h
        positions.append(x)
        vitesses.append(v)
        temps.append(t)

    return temps, positions, vitesses
```

3. 4. et 5. Les résultats étaient déjà donnés dans l'énonce. Sans surprise, la solution numérique est d'autant plus proche de la solution analytique que le pas est petit.

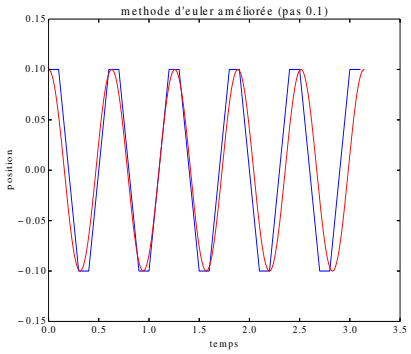
```
import matplotlib.pyplot as plt
import math as m

s = solve(10,0.001,0,5*2*m.pi/10,0.1,0)
plt.plot(s[0],s[1],color='b')
liste_t = [ i*(5*2*m.pi/10)/1000 for i in range(1001) ]
liste_x = [ 0.1*m.cos(10*t) for t in liste_t ]
plt.plot(liste_t,liste_x,color='r')
plt.xlabel("temps")
plt.ylabel("position")
plt.title("methode d'euler")
plt.show()
```

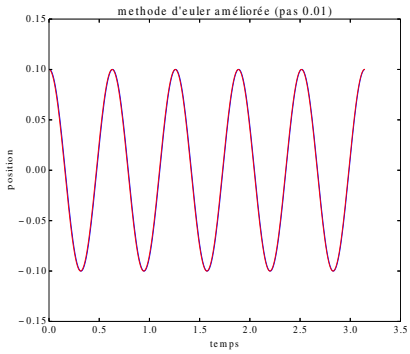
2 Méthode d'Euler améliorée

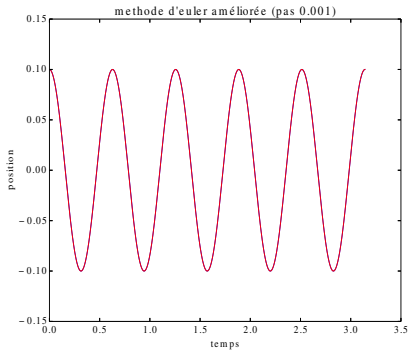
1. Il y a peu de choses à modifier :

```
def solve2(w0,h,a,b,x0,v0) :

    x,v = x0,v0
    t = a
    positions = [x]
    vitesses = [v]
    temps = [t]

    while t < b-h :
        x = x+v*h
        v = v-(w0**2)*x*h
        t = t+h
        positions.append(x)
        vitesses.append(v)
        temps.append(t)

    return temps, positions, vitesses
```

2. Mais les résultats sont très différents !







3. Cela revient à prendre la dérivée au milieu de l'intervalle et non à gauche de l'intervalle.

3 Equation non linéaire

On doit modifier *solve* pour tenir compte de la nouvelle équation :

```
def solve3(w0,h,a,b,x0,v0) :

    from math import sin
    x,v = x0,v0
    t = a
    positions = [x]
    vitesses = [v]
    temps = [t]

    while t < b-h :
        x = x+v*h
        v = v-(w0**2)*sin(x)*h
        t = t+h
        positions.append(x)
        vitesses.append(v)
        temps.append(t)

    return temps, positions, vitesses
```

4 Non isochronisme des oscillations

1. Attention à ne pas tester l'égalité d'une valeur de position avec zéro, mais à définir un seuil en dessous duquel une valeur est considérée comme proche de zero.

```
    def periode(temps,positions) :

        # on recupere la valeur le max de la liste des positions
        max = positions[0]
        for elt in positions :
            if elt > max :
                max = elt
        # epsilon est l'ecart a la moyenne en dessous duquel
        # on considerera une valeur comme proche de zero
        epsilon = max / 100

        liste = []
        n = len(temps)
        i = 0
        # on parcourt la liste des positions et on recherche
        # les bornes a et b d'intervalles a l'interieur desquels
        # les valeurs sont inferieures a epsilon (en valeur absolue)
        # puis on ajoute le milieu de l'intervalle a la liste
        while i < n :
            if abs(positions[i]) < epsilon :
                a = temps[i]
                while abs(positions[i]) < epsilon and i < n-1 :
                    i += 1
                b = temps[i-1]
                liste.append((a+b)/2)
            i += 1
```

```
        # on calcul la moyenne des ecarts entre deux valeurs
        # consecutives de la liste, que l'on renvoie
        s = 0
        n = len(liste)
        if n > 1 :
            for i in range(1,n) :
                s += liste[i]-liste[i-1]
            T = 2 * s / (n-1)
        else :
            T = None
        return T
```

2. On utilise les fonctions *solve3* et *période* :

```
import matplotlib.pyplot as plt

amplitudes = []
periodes = []

for i in range(100) :
    x0 = 0.03*(i+1)
    amplitudes.append(x0)
    s = solve3(10,0.001,0,5*2*m.pi/10,x0,0)
    T = periode(s[0],s[1])
    periodes.append(T)

plt.plot(amplitudes,periodes,'b')
plt.xlabel("amplitude")
plt.ylabel("periode")
plt.axis([0,4,0,2.5])
plt.title("evolution de la periode avec l'amplitude")
plt.show()
```

3. Plus l'amplitude est faible et plus l'approximation $\sin(\theta) \simeq \theta$ est pertinente, la période tend donc vers $T_0 = \frac{2\pi}{\omega_0}$ lorsque l'amplitude tend vers 0, et numériquement $T_0 = 0,628s$ pour $\omega_0 = 10s^{-1}$.
4. Cette expression signifie que la période des oscillations dépend de l'amplitude, on constate bien ici que la période augmente avec l'amplitude.