Séance 3: Euler Implicite et Runge Kutta

> Nom Chinois: CAI Pengfei
> Pénom françaus: Vincent
> Numéro d'étudiant: SY1724107

On fournie deux versions de code, le code matlab sont juste après le code python.

3. Euler Implicite

3.1) Pour euler implicite, on a matrice A:

$$A = \begin{bmatrix} \frac{1}{1+w_0^2\Delta^2 t} & \frac{\Delta t}{1+w_0^2\Delta^2 t} \\ \frac{-w_0^2\Delta t}{1+w_0^2\Delta^2 t} & \frac{1}{1+w_0^2\Delta^2 t} \end{bmatrix}$$

code python:

```python
import matplotlib.pyplot as plt
import numpy as np


def calculate_E(q, q_dot, w_0):
    E = 0.5*(q_dot*q_dot + w_0*w_0*q*q)
    return E

def obtenir_lists(delta_t):
    w_0 = 2*np.pi
    T_0 = 3
    A = np.array([[1./(1.+w_0*w_0*delta_t*delta_t), delta_t/(1.+w_0*w_0*delta_t*delta_t)],
                  [-w_0*w_0*delta_t/(1. + w_0*w_0*delta_t*delta_t), 1./(1.+w_0*w_0*delta_t*delta_t)]])
    etat_init = np.array([1, 0])
    # transpose
    etat_init = etat_init.reshape(-1, 1)

    i = 0
    q_list = []
    t_list = []
    E_list = []
    E_standard = []
    etat_i = etat_init
    t_list.append(i*delta_t)
    q_list.append(etat_i[0][0])
    E_list.append(calculate_E(etat_i[0][0], etat_i[1][0], w_0))
    E_standard.append(w_0*w_0/2.0)

    while(i*delta_t < T_0):
        etat_i = A.dot(etat_i)
        i = i+1
        t_list.append(i * delta_t)
        q_list.append(etat_i[0][0])
        E_list.append(calculate_E(etat_i[0][0], etat_i[1][0], w_0))
        E_standard.append(w_0 * w_0 / 2.0)

    return t_list, q_list, E_list, E_standard

if __name__ == '__main__':

    dt1 = 0.01
    dt2 = 0.02

    t_list1, q_list1, e_list1, e_standard1 = obtenir_lists(dt1)
    plt.plot(t_list1, q_list1, color="C0", label=r"euler implicite, $\Delta t = {}$".format(dt1))

    plt.ylabel("q")
    plt.xlabel("t")
    plt.legend()
    plt.titile("euler implicite")
    plt.show()
```

code matlab

calculate_E.m

```
function E = calculate_E(q, q_dot, w_0)

E = 0.5*(q_dot*q_dot + w_0*w_0*q*q);

end
```

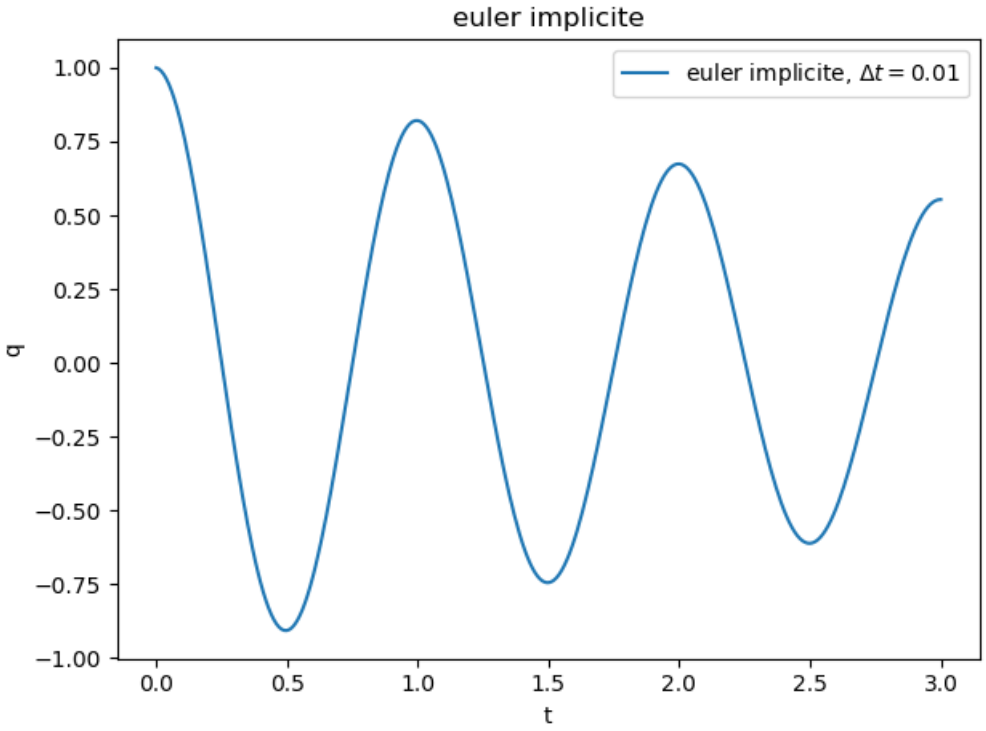
obtenir_lists.m

```
function [t_list, q_list, E_list, E_standard] =  obtenir_lists(delta_t)

w_0 = 2*pi;
T_0 = 3;
A = [1./(1.+w_0*w_0*delta_t*delta_t), delta_t/(1.+w_0*w_0*delta_t*delta_t);
              -w_0*w_0*delta_t/(1. + w_0*w_0*delta_t*delta_t), 1.0/(1.+w_0*w_0*delta_t*delta_t)];
etat_init = [1, 0];

etat_init = etat_init';

i = 0;
q_list = [];
t_list = [];
E_list = [];
E_standard = [];
etat_i = etat_init;
t_list = [t_list, i*delta_t];
q_list = [q_list, etat(1, 1)];
E_list = [E_list, calculate_E(etat_i(1, 1), etat_i(2, 1), w_0)];
E_standard = [E_standard, w_0*w_0/2.0];

while(i*delta_t < T_0)
    etat_i = A*etat_i;
    i = i+1;
    t_list = [t_list, i*delta_t];
    q_list = [q_list, etat_i(0, 0)];
    E_list = [E_list, calculate_E(etat_i(1, 1), etat_i(2, 1), w_0)];
    E_standard = [E_standard, w_0*w_0/2.0];
end

end
```

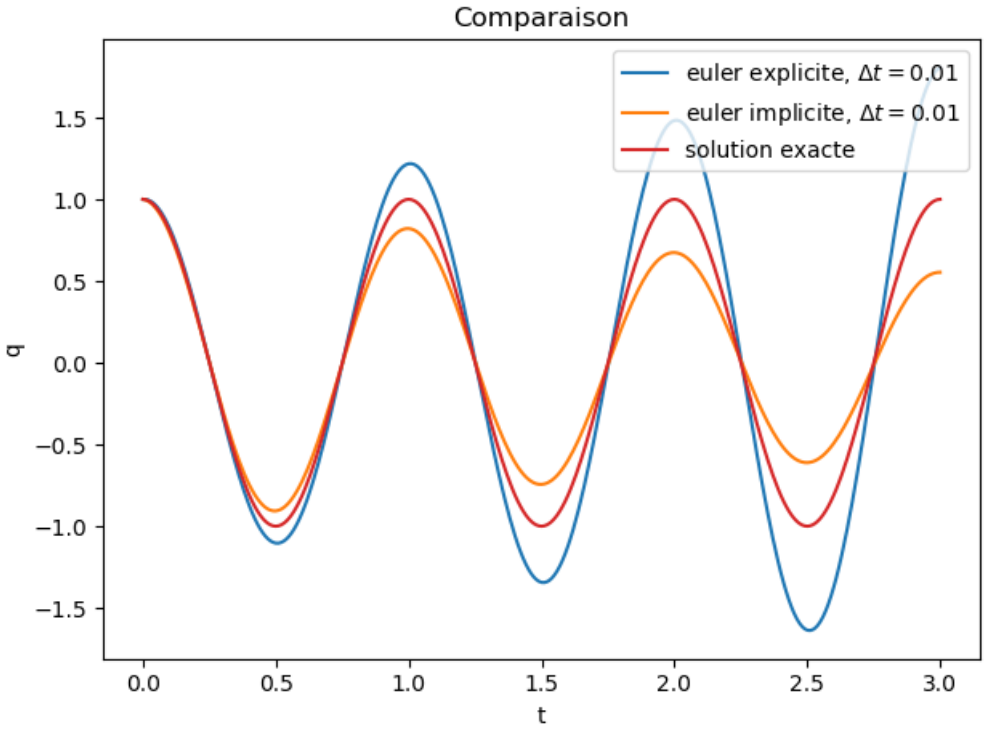
main.m

```
dt1 = 0.01;
dt2 = 0.02;

[t_list1, q_list1, e_list1, e_standard1] = obtenir_lists(dt1);
plot(t_list1, q_list1, 'DisplayName', sprintf('euler implicite, dt=%0.2f', dt1))

ylabel('q')
xlabel('t')
title('euler implicite')
```

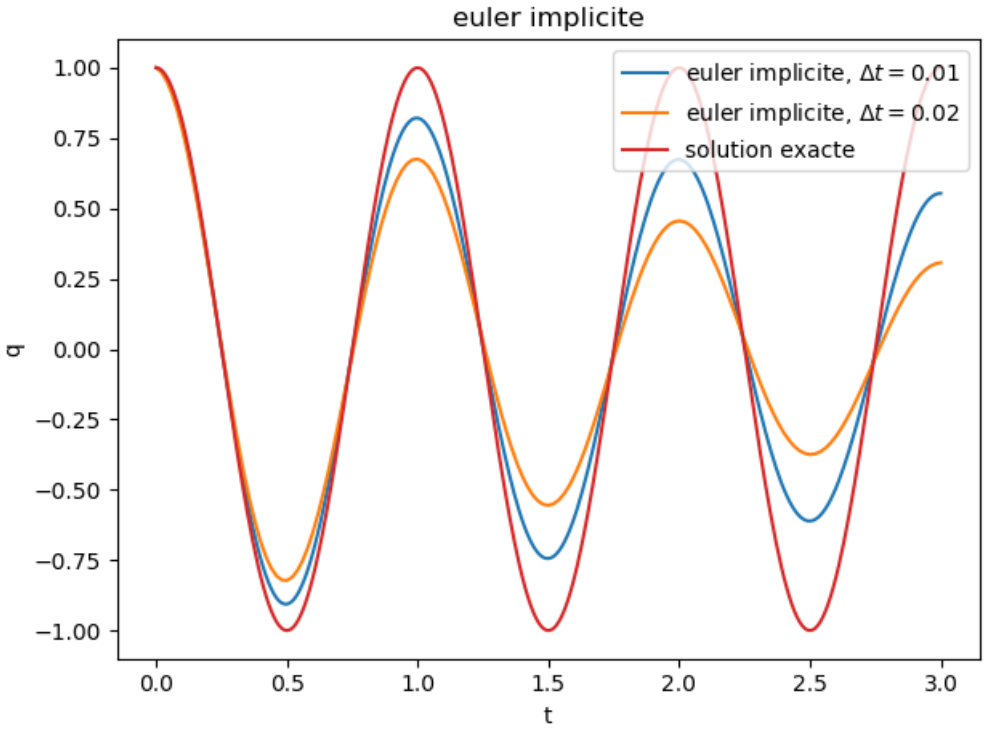
On a la figure:



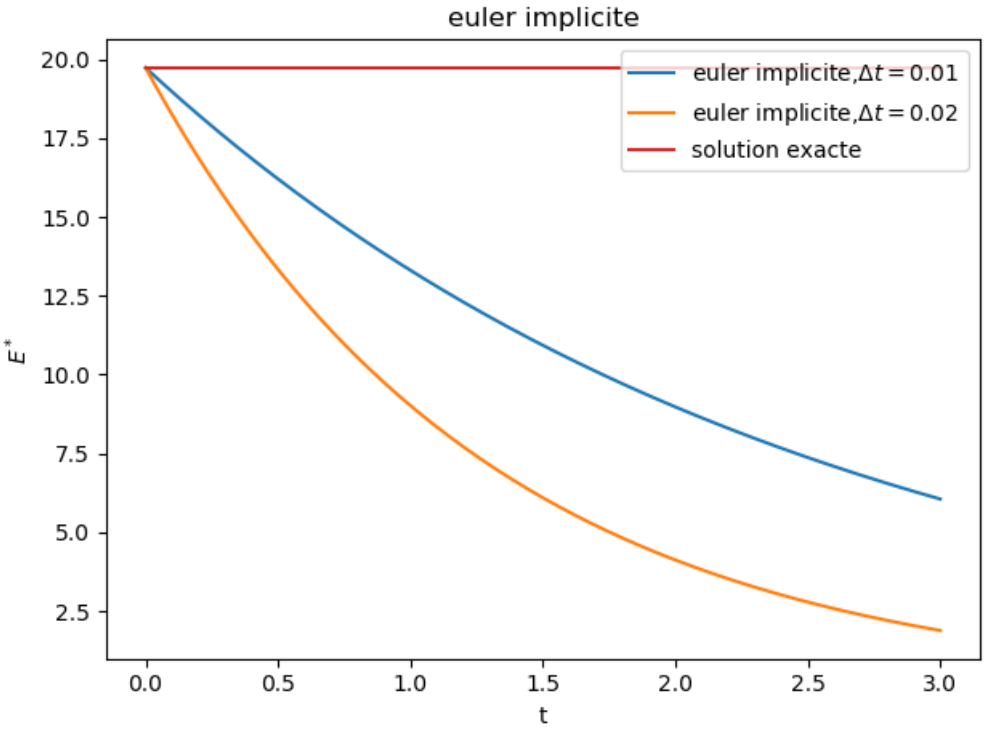
3.2) On a la figure ci-dessous pour 3 solutions différents:



3.3) Pour la différents pas, on peut voir très clairement, plus le pas Δt est petit, plus l'atténuation des oscillations est faible.



3.4) on a la figure ci-dessous:



Donc pour la méthode euler implicite, l'énergie mécanique de l'oscillateur devient de plus en plus petit. Plus la valeur Δt grand, l'énergie descend plus vite.

3.5) On suppose:

$$A = \begin{bmatrix} \frac{1}{1+w_0^2\Delta^2 t} & \frac{\Delta t}{1+w_0^2\Delta^2 t} \\ \frac{-w_0^2\Delta t}{1+w_0^2\Delta^2 t} & \frac{1}{1+w_0^2\Delta^2 t} \end{bmatrix}$$

D'après la définition de valeur prope, on a $Ax = \lambda x$. x est vecteur prope. on a $(\lambda I - A)x = 0$.

$$\lambda I - A = \begin{bmatrix} \lambda - \frac{1}{1+w_0^2\Delta^2 t} & \frac{-\Delta t}{1+w_0^2\Delta^2 t} \\ \frac{w_0^2\Delta t}{1+w_0^2\Delta^2 t} & \lambda - \frac{1}{1+w_0^2\Delta^2 t} \end{bmatrix}$$

Il faut calcule $det(\lambda I - A)$

$$det(\lambda I - A) = (\lambda - \frac{1}{1 + w_0^2\Delta^2 t})^2 + \frac{w_0^2\Delta^2 t}{(1 + w_0^2\Delta^2 t)^2} = 0$$

Donc:

$$\lambda = \frac{1 \pm i w_0 \Delta t}{1 + w_0^2\Delta^2 t}$$

On a vu $|\lambda| < 1$, Donc il est bien stable. Et il y une atténuation aussi.

4 Runge Kutta

4.1) Supposons que $\Omega = \dot{q}$. Donc on deux équations adapté aux schéma du premier ordre.

$$\dot{q} = \Omega$$

$$\dot{\Omega} = -w_0^2 q$$

Donc on a

$$\begin{cases} k_1 = \Omega_0 \\ j_1 = -w_0^2 q_0 \\ k_2 = \Omega_0 + \frac{j_1 \Delta t}{2} \\ j_2 = -w_0^2 (q_0 + \frac{k_1 \Delta t}{2}) \\ k_3 = \Omega_0 + \frac{j_2 \Delta t}{2} \\ j_3 = -w_0^2 (q_0 + \frac{k_2 \Delta t}{2}) \\ k_4 = \Omega_0 + j_3 \Delta t \\ j_4 = -w_0^2 (q_0 + k_3 \Delta t) \end{cases}$$

Donc on a bien

$$q_1 = q_0 + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)\Delta t$$

et

$$\Omega_1 = \Omega_0 + \frac{1}{6}(j_1 + 2j_2 + 2j_3 + j_4)\Delta t$$

4.2) code python:

```python
import matplotlib.pyplot as plt
import numpy as np


def one_step(q0, omega0, w0, deltaT):
    k1 = omega0
    j1 = -w0 * w0 * q0
    k2 = omega0 + j1 * deltaT / 2.
    j2 = -w0 * w0 * (q0 + k1 * deltaT / 2)
    k3 = omega0 + j2 * deltaT / 2
    j3 = -w0 * w0 * (q0 + k2 * deltaT / 2)
    k4 = omega0 + j3 * deltaT
    j4 = -w0 * w0 * (q0 + k3 * deltaT)

    q1 = q0 + (k1 + 2 * k2 + 2 * k3 + k4) * deltaT / 6.
    omega1 = omega0 + (j1 + 2 * j2 + 2 * j3 + j4) * deltaT / 6.
    return q1, omega1


def runge_kutta(deltaT, T0, q0, omega0, w0):
    t_list = []
    q_list = []

    i = 0
    t_list.append(i * deltaT)
    q_list.append(q0)
    q = q0
    omega = omega0
    while (i * deltaT < T0):
        q, omega = one_step(q, omega, w0, deltaT)
        i = i + 1
        t_list.append(i * deltaT)
        q_list.append(q)

    return t_list, q_list


if __name__ == '__main__':
    delta_t = 0.01
    T0 = 3

    # etat initial
    q0 = 1
    omega0 = 0
    w0 = 2*np.pi

    t_list, q_list = runge_kutta(delta_t, T0, q0, omega0, w0)
    plt.plot(t_list3, q_list3, color="C2", label="runge kutta, $\Delta t = {}$".format(dt1))
    plt.ylabel("q")
    plt.xlabel("x")
    plt.title("Runge Kutta")
    plt.legend()
    plt.show()
```

code matlab:

one_step.m

```
function [q1, omega1] = one_step(q0, omega0, w0, deltaT)

k1 = omega0;
j1 = -w0 * w0 * q0;
k2 = omega0 + j1 * deltaT / 2.0;
j2 = -w0 * w0 * (q0 + k1 * deltaT / 2);
k3 = omega0 + j2 * deltaT / 2;
j3 = -w0 * w0 * (q0 + k2 * deltaT / 2);
k4 = omega0 + j3 * deltaT;
j4 = -w0 * w0 * (q0 + k3 * deltaT);

q1 = q0 + (k1 + 2 * k2 + 2 * k3 + k4) * deltaT / 6.0;
omega1 = omega0 + (j1 + 2 * j2 + 2 * j3 + j4) * deltaT / 6.0;

end
```

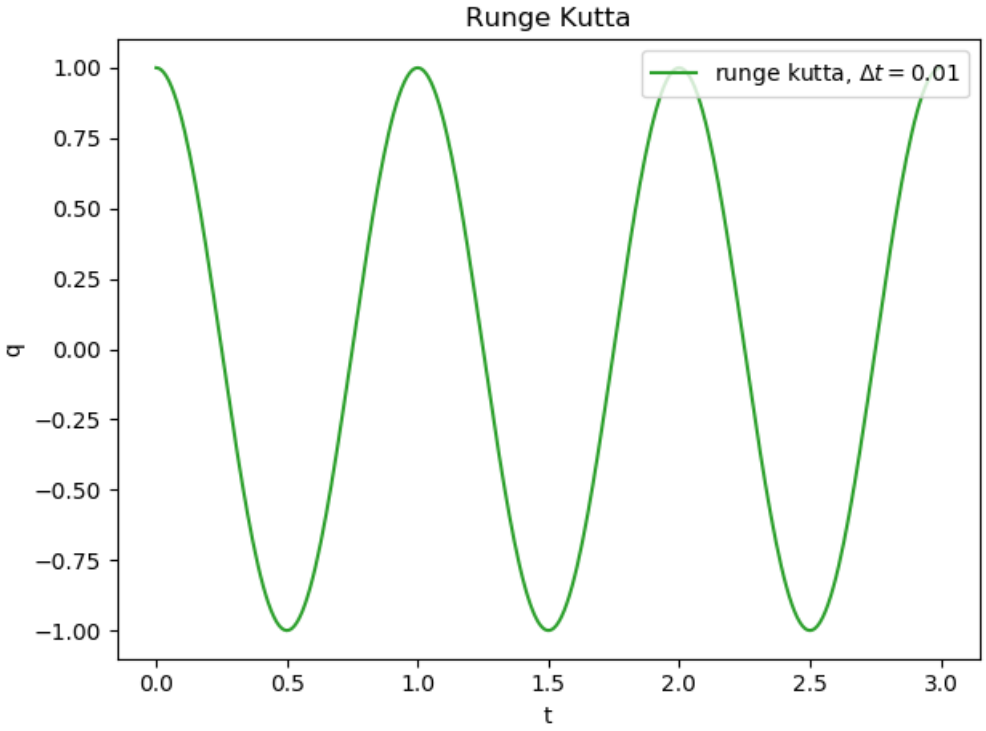
runge_kutta.m

```
function [t_list, q_list] = runge_kutta(deltaT, T0, q0, omega0, w0)

t_list = [];
q_list = [];

i = 0;
t_list = [t_list, i*deltaT];
q_list = [q_list, q0];
q = q0;
omega = omega0;
while (i * deltaT < T0)
    [q, omega] = one_step(q, omega, w0, deltaT);
    i = i + 1;
    t_list = [t_list, i*deltaT];
    q_list = [q_list, q];

end

end
```

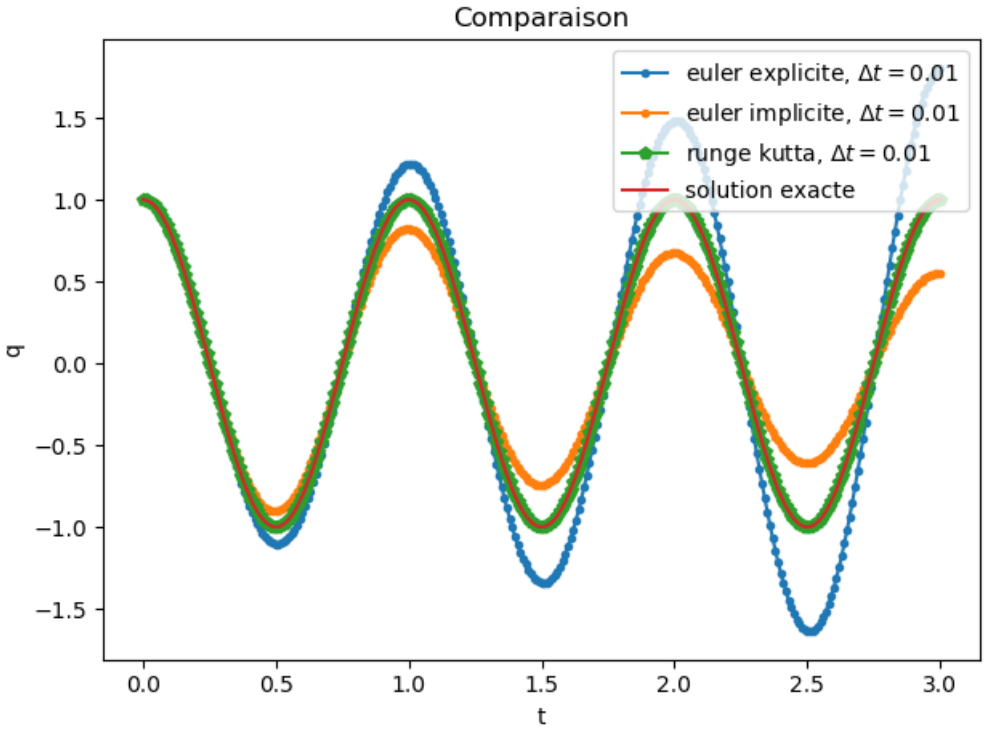
main.m

```
delta_t = 0.01;
T0 = 3;

% etat initial
q0 = 1;
omega0 = 0;
w0 = 2*pi;

[t_list, q_list] = runge_kutta(delta_t, T0, q0, omega0, w0);
plot(t_list3, q_list3, 'DisplayName', sprintf('runge kutta, dt=%0.2f', delta_t))
ylabel('q')
xlabel('x')
title('Runge Kutta')
```

On a la figure:

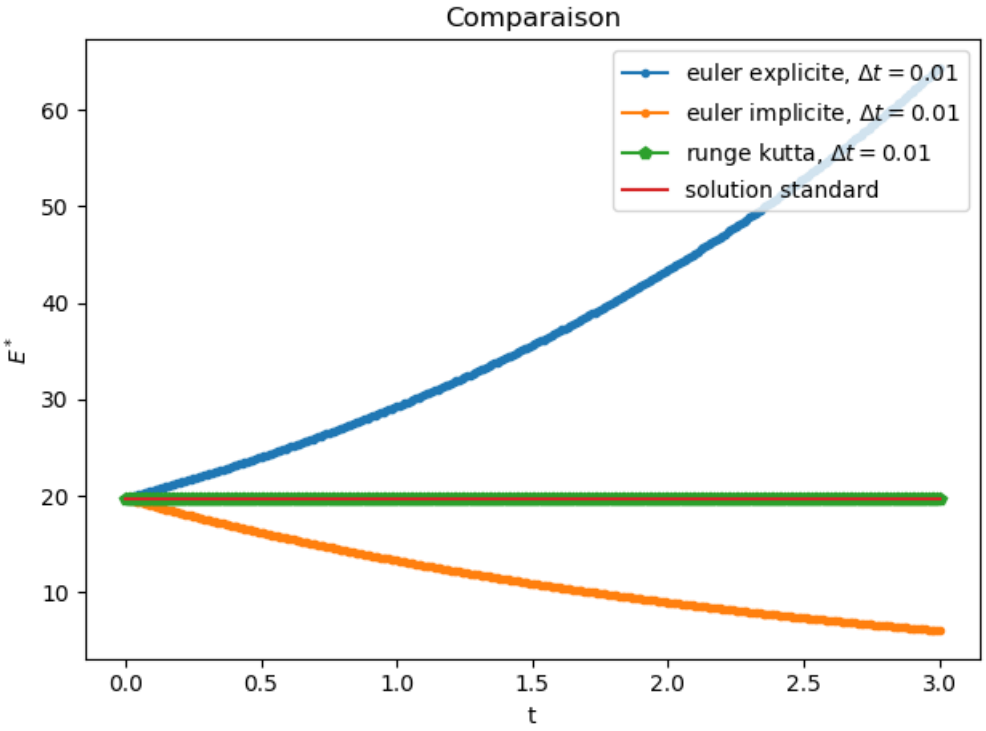


4.3) On a 3 résolutions maintenant.



D'après la figure, on peut voir il y a divergence pour la méthode Euler explicite, il y atténuation pour la méthode Euler implicite. Mais pour la méthode Runge Kutta, on peut voir une très bonne superposition avec la solution standard.

4.4)



L'énergie mécanique reste la même pour la méthode Runge Kutta. On peut voir très clairement une bonne performance pour la méthode Runge Kutta.