Travaux dirigés sur la résolution numérique des équations du mouvement.

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

On fournie deux version de code, les codes matlab sont juste après les codes python.

Osilateur conservatif linéaire à un degré de liberté

1 Solution Analytique

1.1) Pour l'équation

$$\ddot{q} + w_0^2 q = 0 \tag{1}$$

On associe son équation caractéristique $r^2 + w_0^2 = 0$. On a deux résolution pour cette équation: $r_1 = iw_0$ et $r_2 = -iw_0$.
Donc on propose la solution générale de l'équation (1) est:

$$q = C_1 cos(w_0 t) + C_2 sin(w_0 t)$$

D'après les conditions inititiales, $w_0 = 2\pi \text{ rad}.s^{-1}$, $q(t=0) = 1$ et $\dot{q}(t=0) = 0$, on a $C_1 = 1, C_2 = 0$.
Donc:

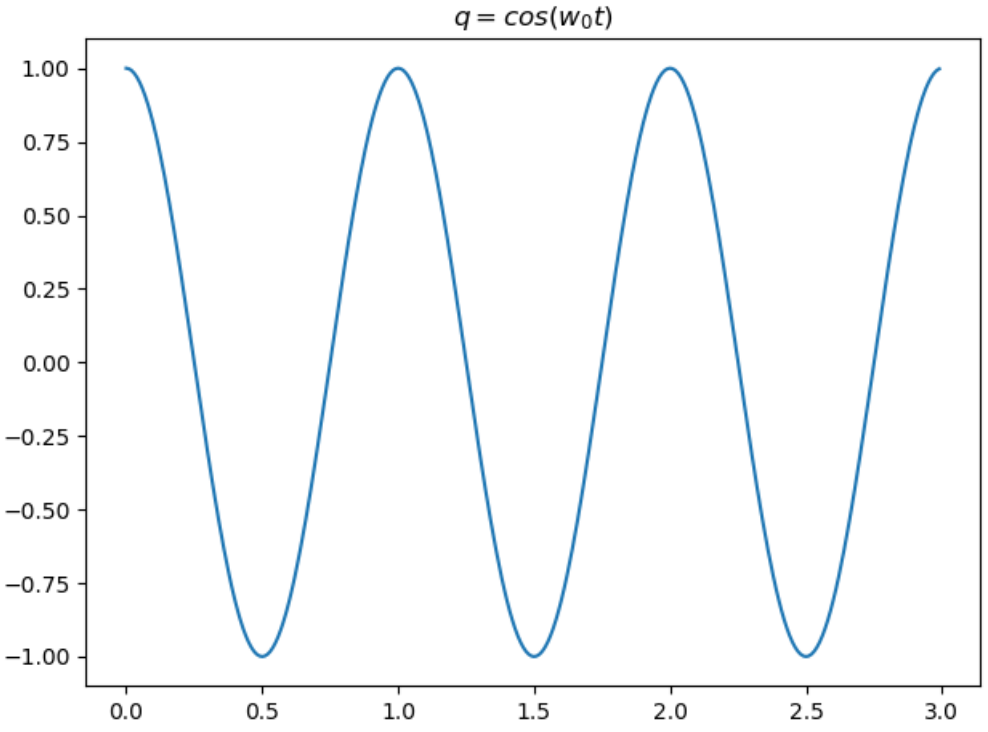
$$q = cos(w_0 t)$$

```python
import matplotlib.pyplot as plt
import numpy as np
if __name__ == '__main__':
    t = np.arange(0, 3, 0.01)
    w_0 = 2*np.pi
    q = np.cos(w_0*t)
    plt.plot(t, q)
    plt.title(r"$q = cos(w_0t)$")
    plt.show()
```

code matlab

```matlab
t = 0:0.01:3;
w0 = 2*pi;
q = cos(t);
plot(t, q)
```

Et on a la figure ci-dessous:



1.2) E^* est définie par la relation $E^* = \frac{1}{2}(\dot{q}^2 + w_0^2 q^2)$, donc on a

$$E^* = \frac{w_0^2}{2}$$

Donc l'énergie méganique de l'oscillateur reste la même.

2. Euler Explicite

2.1) comme on a éauqtion (1), on le multiplie par Δt donc on a

$$\Delta t\ddot{q} + \Delta t w_0^2 q = 0$$

Donc on a $\Delta t\ddot{q} = -\Delta t w_0^2 q$. D'après l'équation (5), on a $\dot{q}_{j+1} = \dot{q}_j + \Delta t\ddot{q}_j$. on a:

$$\dot{q}_{j+1} = -\Delta t w_0^2 q_j + \dot{q}_j$$

Et comme $q_{j+1} = q_j + \Delta t\dot{q}_j$. On a bien l'équation (5).

$$\begin{bmatrix} q_{j+1} \\ \dot{q}_{j+1} \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ -w_0^2 \Delta t & 1 \end{bmatrix} \begin{bmatrix} q_j \\ \dot{q}_j \end{bmatrix} \quad (5)$$

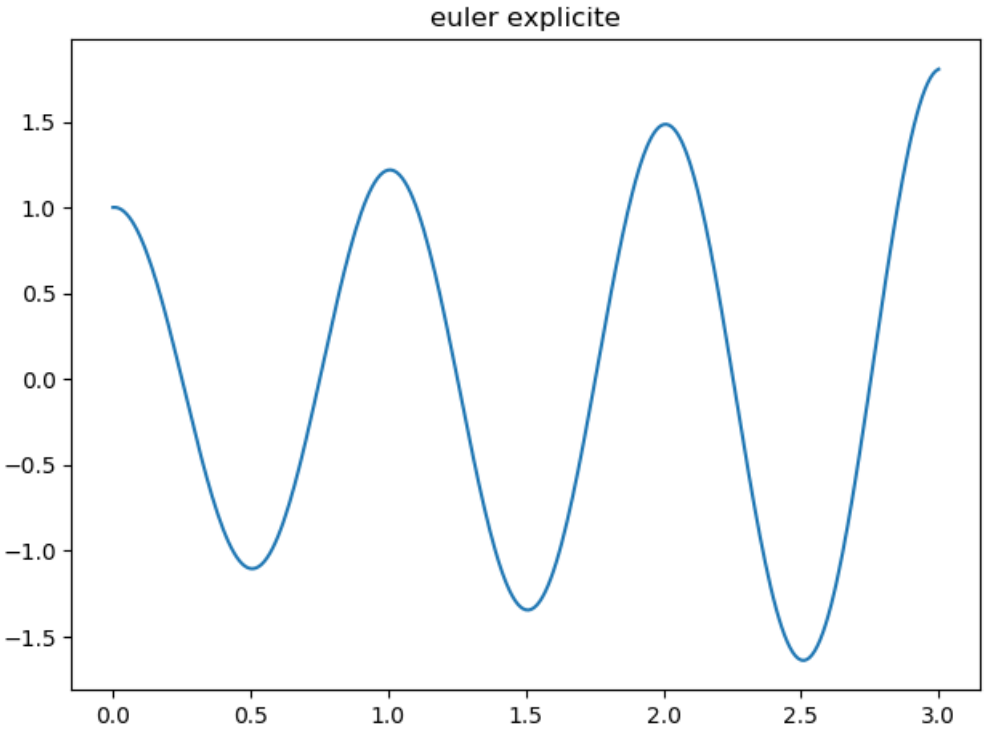
2.2) code pour la programmation en utilisant la méthode 2:

```python
import matplotlib.pyplot as plt
import numpy as np
if __name__ == '__main__':
    delta_t = 0.01
    w_0 = 2*np.pi
    T_0 = 3
    A = np.array([[1, delta_t],[-w_0*w_0*delta_t, 1]])
    etat_init = np.array([1, 0])
    # transpose
    etat_init = etat_init.reshape(-1, 1)

    i = 0
    q_list = []
    t_list = []
    etat_i = etat_init
    t_list.append(i*delta_t)
    q_list.append(etat_i[0][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])

    plt.plot(t_list, q_list)
    plt.title("euler explicite")
    plt.show()
```

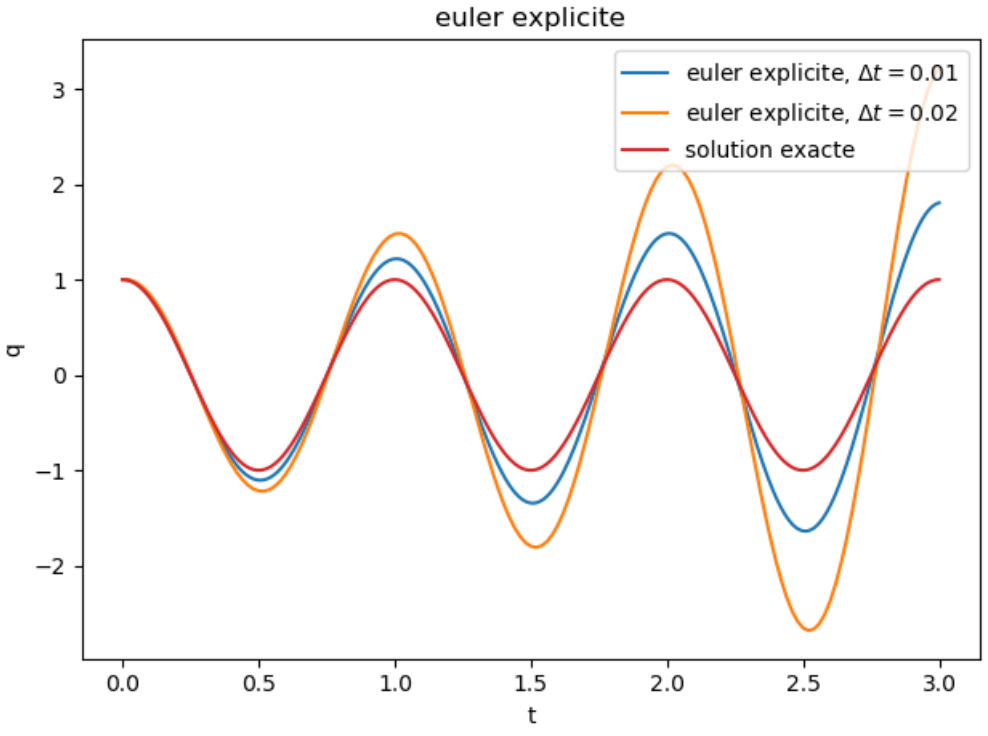
code matlab

```matlab
delta_t = 0.01;
w_0 = 2*pi;
T_0 = 3;
A = [1, delta_t; -w_0*w_0*delta_t, 1];
etat_init = [1; 0];

i = 0;
q_list = [];
t_list = [];
etat_i = etat_init;
t_list = [t_list, i*delta_t];
q_list = [q_list, etat_i(1, 1)];

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(1, 1)];
end

plot(t_list, q_list)
title('euler explicite')
```

on a la figure ci-dessous:

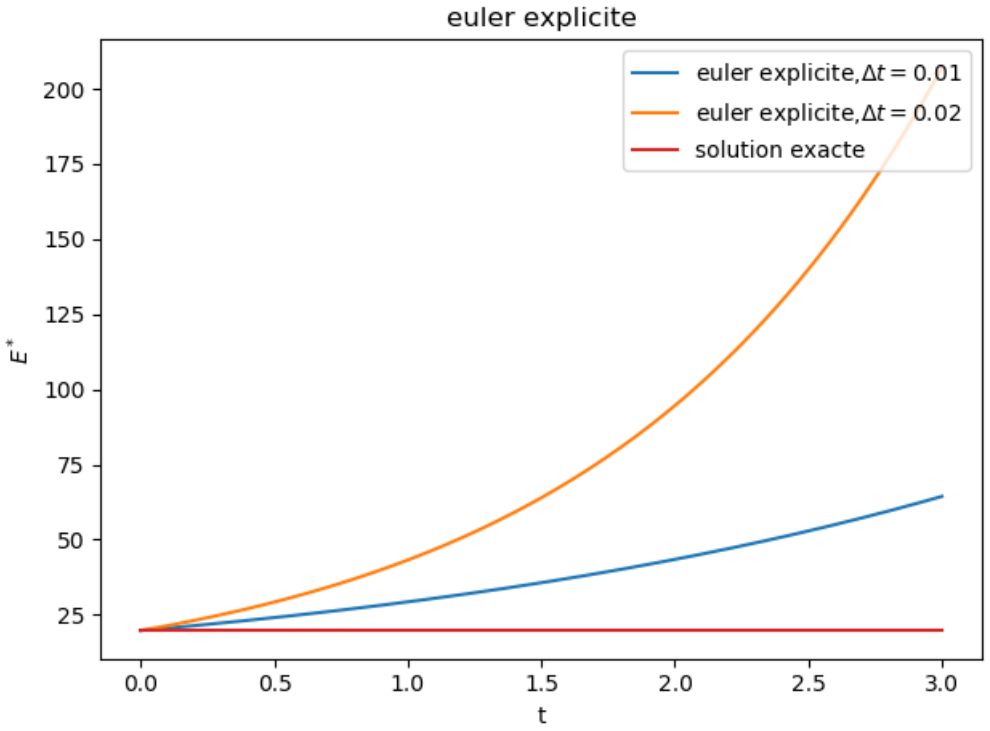


2.3) on a trois courbes ci-dessous



on peut voir très clairement, quant le temps Δt est petit, plus la divergence est lente.

2.4) on a la figure ci-dessous:



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

2.5) On suppose:

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

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

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

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

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

Donc:

$$\lambda = 1 \pm i w_0 \Delta t$$

On a vu $|\lambda| > 1$, Donc il y a forcément une divergence, le courbe n'est pas stable.

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
```

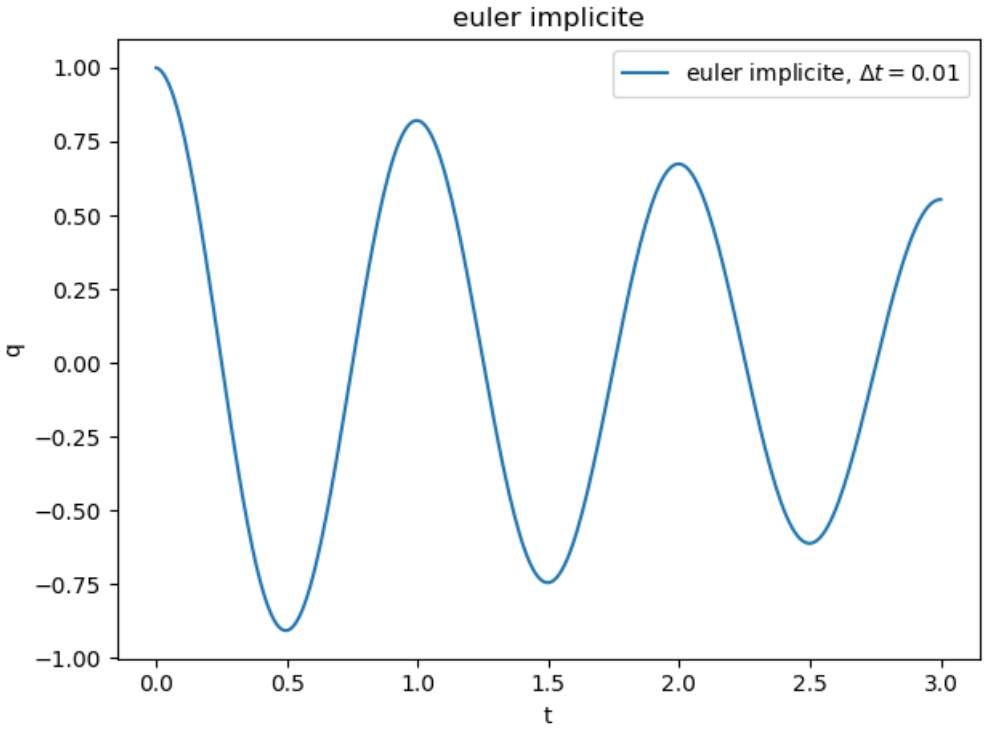
obtenir_lists.m

```
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(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];
end

end
```

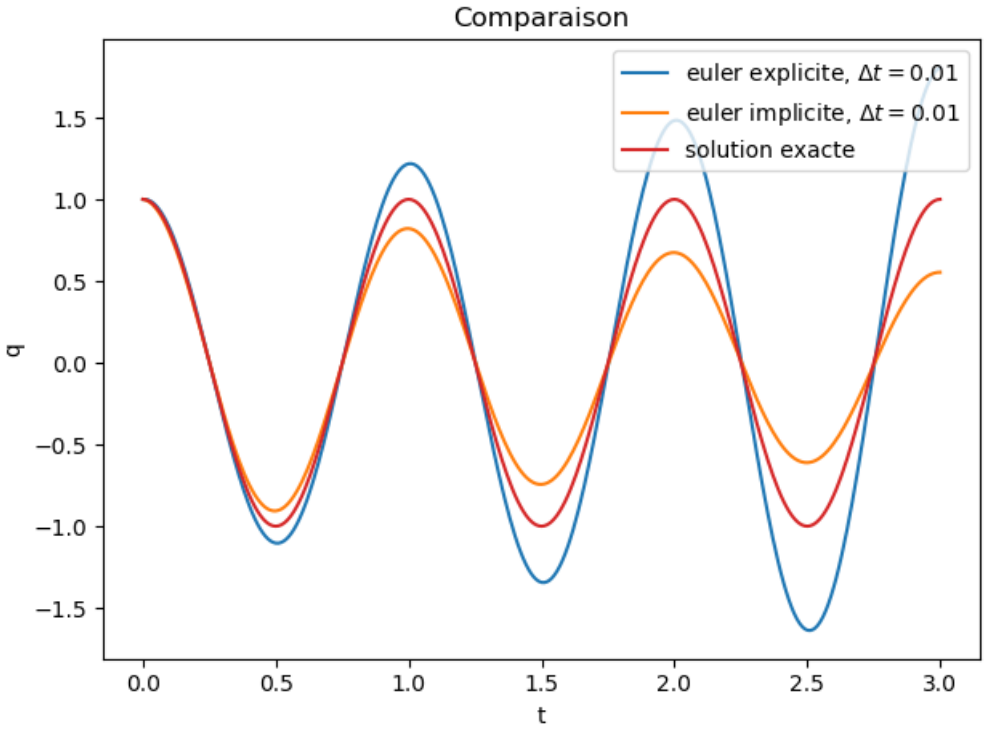
main.m

```
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')
```

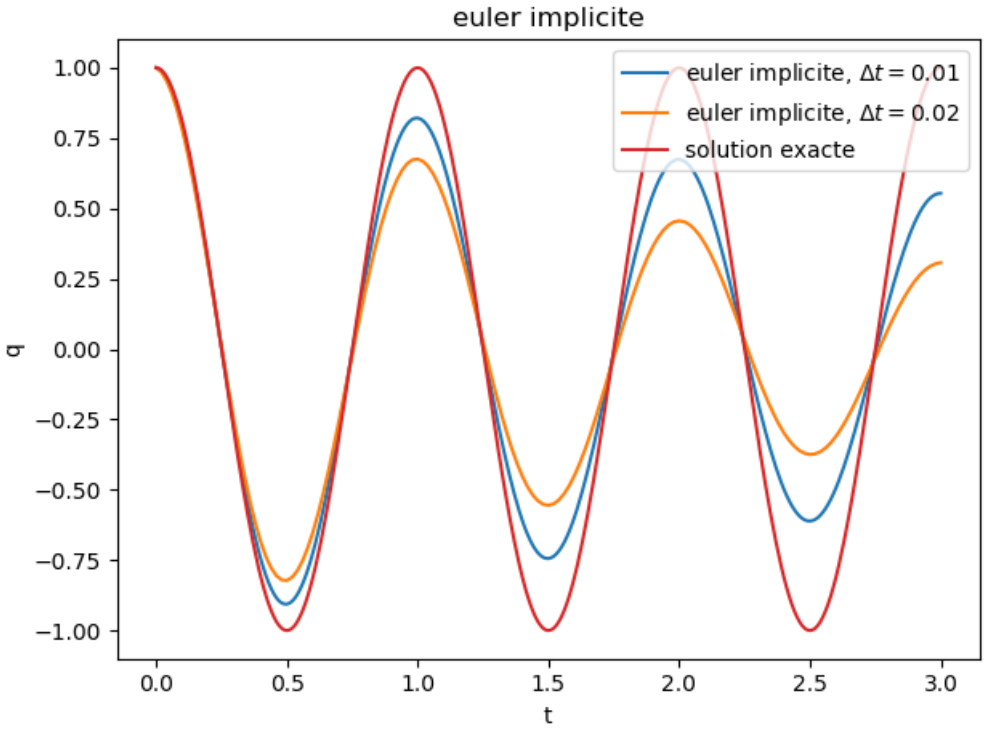
On a la figure:



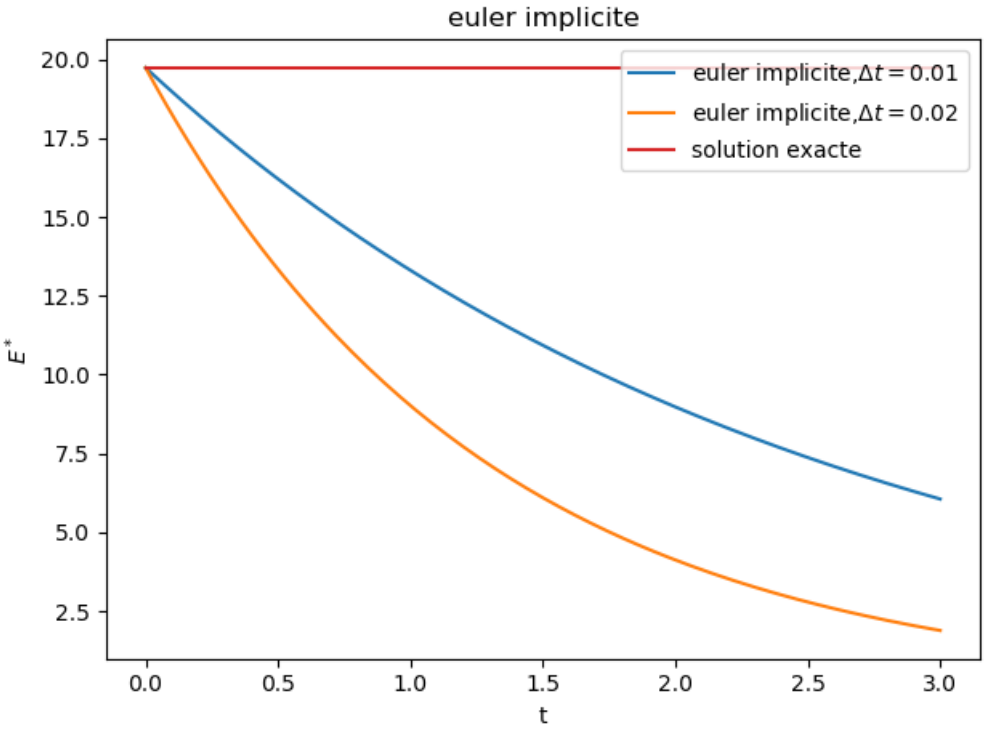
3.2) On a la figure ci-dessous pour 3 solutions différents:



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
```

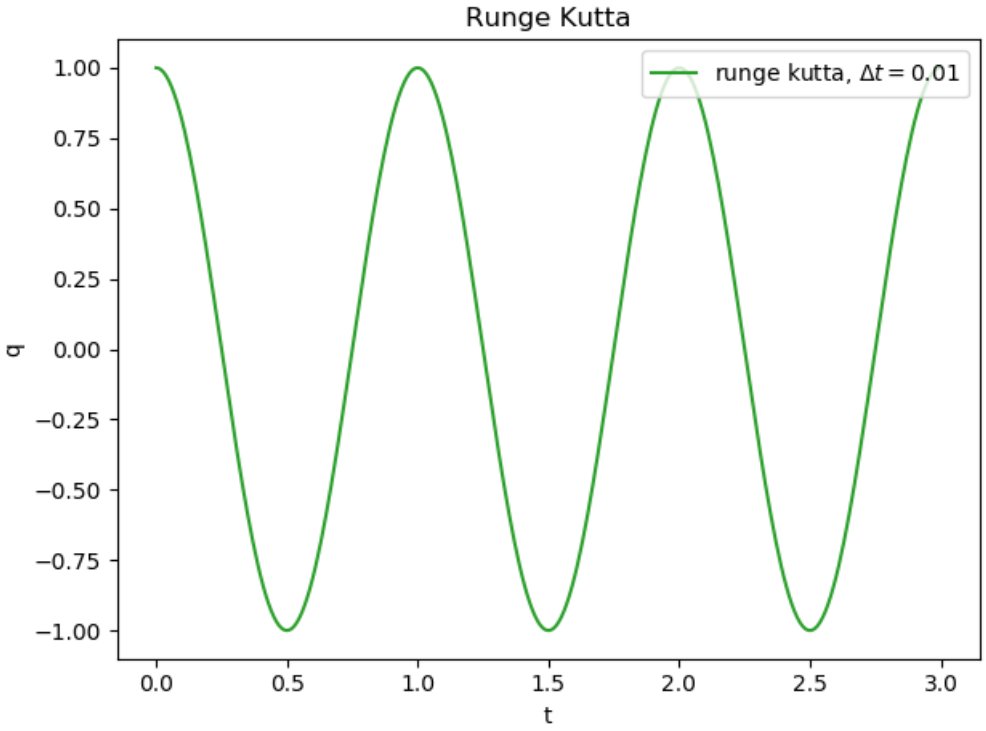
runge_kutta.m

```
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
```

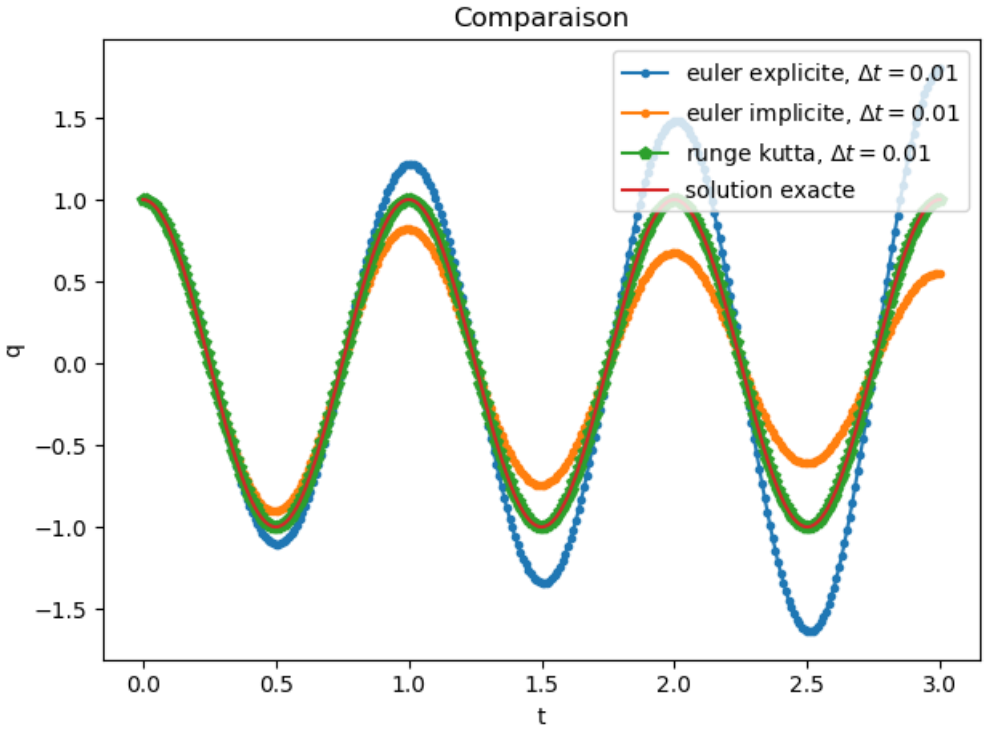
main.m

```
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:

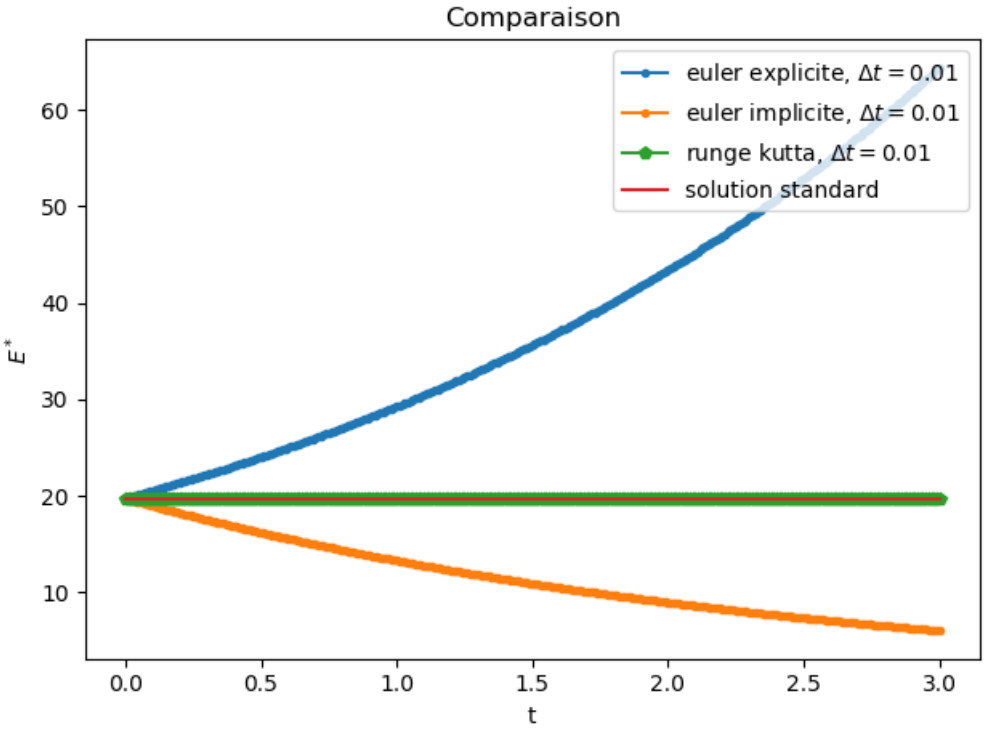


4.3) On a 3 résolutions maintenant.



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.

5 Newmark

5.1) **Résolution avec un schéma de Newmark** $\gamma = 0.5, \beta = 0.25$

code python:

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


def calculate_E(q, omega, w0):
    E = 0.5 * (omega * omega + w0 * w0 * q * q)
    return E


def newmark(deltaT, T0, q0, omega0, w0, gamma=0.25, beta=0.5):
    etat_init = np.array([q0, omega0])
    etat_init = etat_init.reshape(-1, 1)
    B = np.array([[1 + beta * deltaT * deltaT * w0 * w0, 0],
                  [gamma * deltaT * w0 * w0, 1]])
    C = np.array([[1 - (0.5 - beta) * deltaT * deltaT * w0 * w0, deltaT],
                  [-(1 - gamma) * deltaT * w0 * w0, 1]])
    B_inv = np.linalg.inv(B)
    A = B_inv.dot(C)

    t_list = []
    q_list = []
    E_list = []

    i = 0
    etat_i = etat_init
    t_list.append(i * deltaT)
    q_list.append(etat_i[0][0])
    E_list.append(calculate_E(etat_i[0][0], etat_i[1][0], w0))

    while (i * deltaT < T0):
        etat_i = A.dot(etat_i)
        i = i + 1
        t_list.append(i * deltaT)
        q_list.append(etat_i[0][0])
        E_list.append(calculate_E(etat_i[0][0], etat_i[1][0], w0))

    return t_list, q_list, E_list


if __name__ == '__main__':
    dt1 = 0.01
    dt2 = 0.02
    T0 = 3
    q0 = 1
    omega0 = 0
    w0 = 2 * np.pi

    t_list4, q_list4, e_list4 = newmark(dt1, T0, q0, omega0, w0, gamma=0.5, beta=0.25)

    plt.plot(t_list4, q_list4, color="C4", marker="x", label="Newmark, $\Delta t = {}$".format(dt1))
    plt.plot(t_list1, np.cos(2 * np.pi * np.array(t_list1)), color="C3",  label="solution exacte")

    plt.ylabel("q")
    plt.xlabel("t")
    plt.legend(loc="upper right")
    plt.show()
```

code matlab

calculate_E.m

```matlab
function E = calculate_E(q, omega, w0)

E = 0.5 * (omega * omega + w0 * w0 * q * q);

end
```

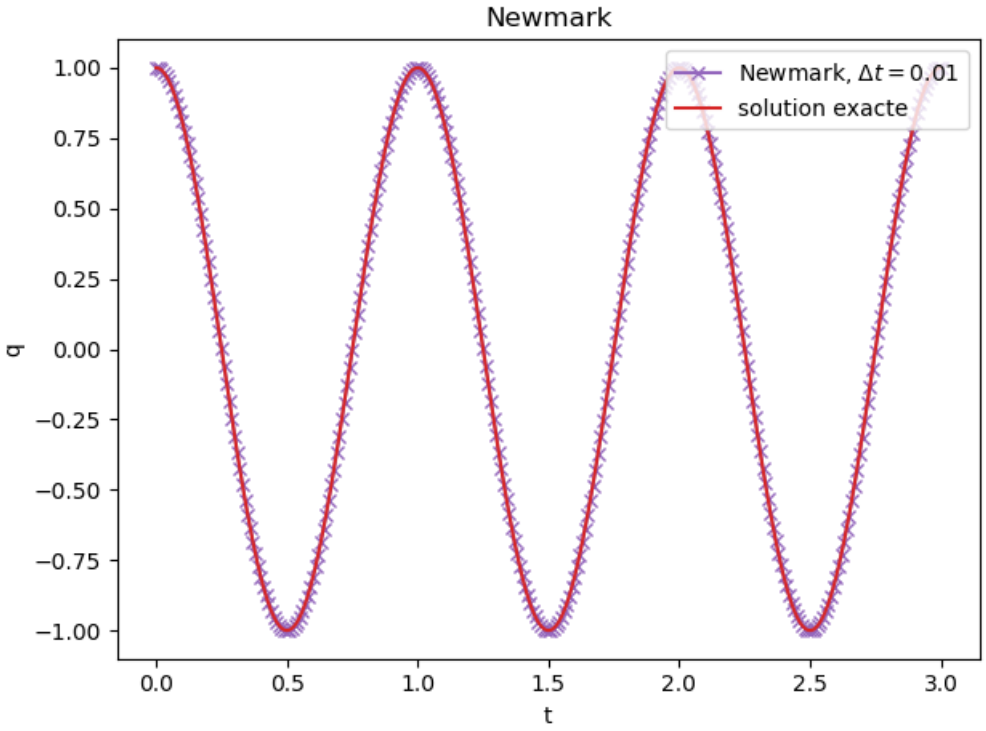
newmark.m

```
function [t_list, q_list, E_list] =  newmark(deltaT, T0, q0, omega0, w0, gamma, beta)

etat_init = [q0, omega0];
etat_init = etat_init';
B = [1 + beta * deltaT * deltaT * w0 * w0, 0;
              gamma * deltaT * w0 * w0, 1];
C = [1 - (0.5 - beta) * deltaT * deltaT * w0 * w0, deltaT;
              -(1 - gamma) * deltaT * w0 * w0, 1];
B_inv = inv(B);
A = B_inv*C;

t_list = [];
q_list = [];
E_list = [];

i = 0;
etat_i = etat_init;
t_list = [t_list, i*deltaT];
q_list = [q_list, etat_i(1,1)];
E_list = [E_list, calculate_E(etat_i(1, 1), etat_i(2, 1), w0)];

while (i * deltaT < T0)
    etat_i = A*etat_i;
    i = i + 1;
    t_list = [t_list, i*deltaT];
    q_list = [q_list, etat_i(1, 1)];
    E_list = [E_list, calculate_E(etat_i(1, 1), etat_i(2, 1), w0)];

end

end
```

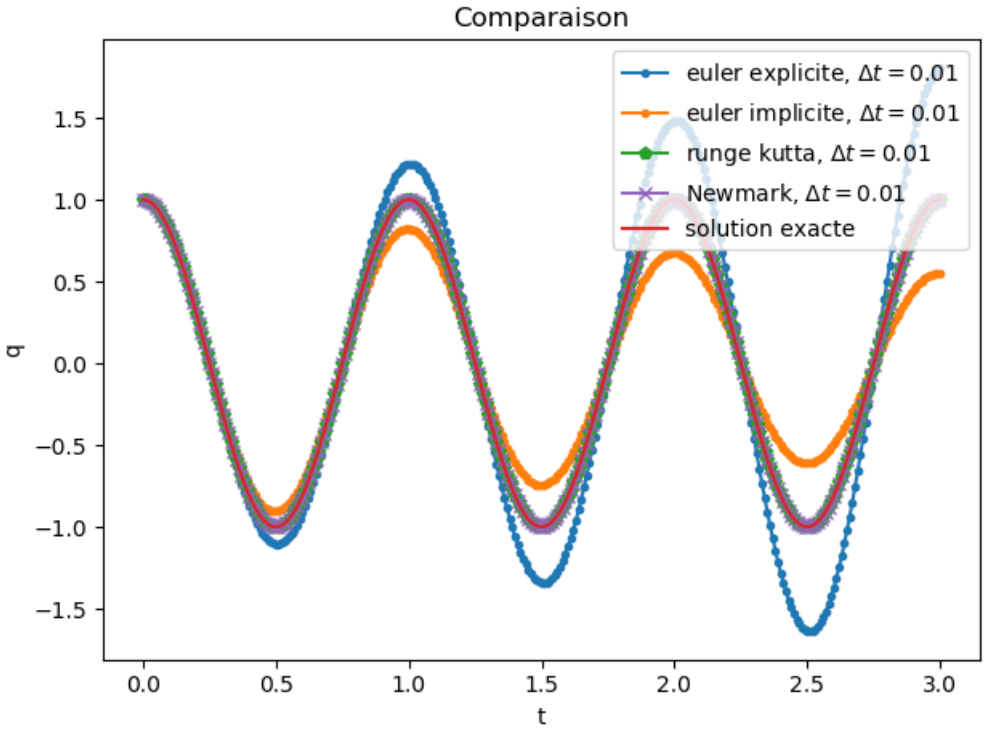
main.m

```
dt1 = 0.01;
dt2 = 0.02;
T0 = 3;
q0 = 1;
omega0 = 0;
w0 = 2 * pi;

gamma = 0.5;
beta = 0.25;


[t_list4, q_list4, e_list4] = newmark(dt1, T0, q0, omega0, w0, gamma, beta);

plot(t_list4, q_list4, 'DisplayName', sprintf('Newmark, dt=%0.2f', dt1))
hold on
plot(t_list4, cos(2 * pi * t_list4), 'DisplayNmae', 'solution exacte')

ylabel('q')
xlabel('t')
```

On a la figure pour la méthode Newmark:

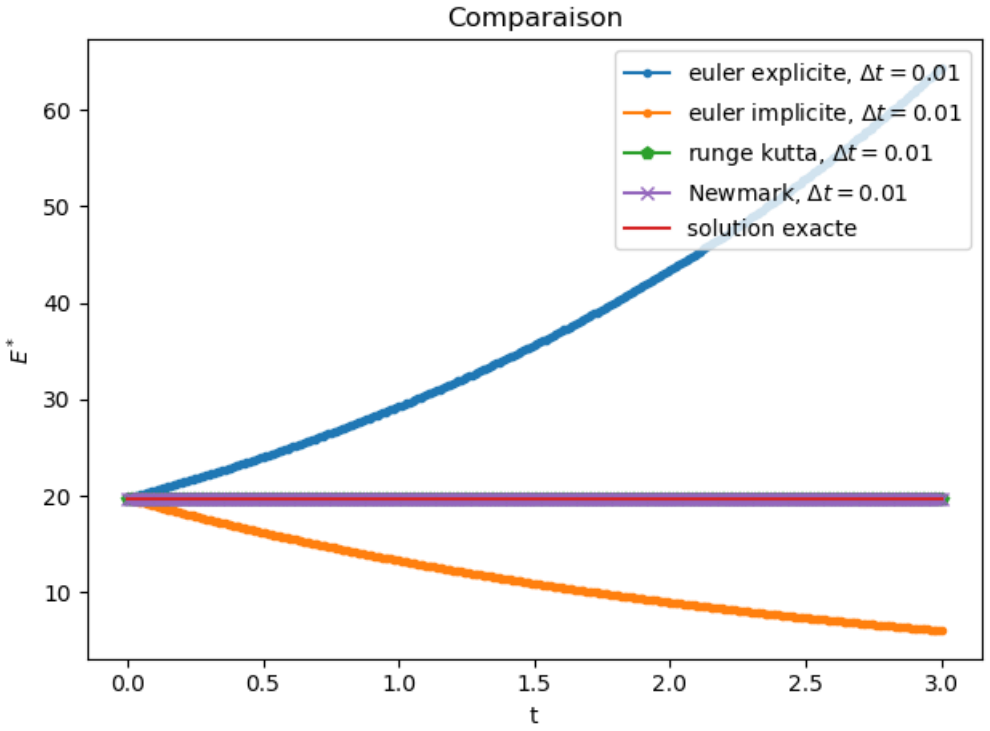


5.1.2) Comparaison avec autre methodes:



Il y a une boone superposition entre la Méthode Runge Kutta et Newmark et la solution standard.

5.1.3) Comparaison avec autre méthodes pour E^*



On peut voir très clairement, il y a une trèse bonne superposition entre la methode Runge Kutta, Newmark et la méthode analytique. Mais pour Euler explicite, il y a une augmentaion de l'énergie mécanique. Pour Euler implicite, il y une déclination de l'énergie mécanique.

5.1.4) Pour calculer les valeurs propres. Il faut d'abord savoir B^{-1}

En fait,

$$B^{-1} = \frac{adj(B)}{det(B)}$$

comme on a

$$B = \begin{bmatrix} 1 + \beta\Delta^2 t w_0^2 & 0 \\ \gamma\Delta t w_0^2 & 1 \end{bmatrix}$$

On a donc

$$adj(B) = \begin{bmatrix} 1 & 0 \\ -\gamma\Delta t w_0^2 & 1 + \beta\Delta^2 t w_0^2 \end{bmatrix}$$

Puis on calcule

$$det(B) = 1 + \beta\Delta^2 t w_0^2$$

Donc on a B^{-1}

$$B^{-1} = \begin{bmatrix} \frac{1}{1+\beta\Delta^2 t w_0^2} & 0 \\ \frac{-\gamma\Delta t w_0^2}{1+\beta\Delta^2 t w_0^2} & 1 \end{bmatrix}$$

On sait que C

$$C = \begin{bmatrix} 1 - (0.5 - \beta)\Delta^2 t w_0^2 & \Delta t \\ -(1 - \gamma)\Delta t w_0^2 & 1 \end{bmatrix}$$

Donc on peut calculer A maintenant:

$$A = B^{-1} \times C$$

$$A = \begin{bmatrix} 1 - \frac{0.5\Delta^2 t w_0^2}{1+\beta\Delta^2 t w_0^2} & \frac{\Delta t}{1+\beta\Delta^2 t w_0^2} \\ (-\gamma\Delta t w_0^2)(1 - \frac{0.5\Delta^2 t w_0^2}{1+\beta\Delta^2 t w_0^2}) - (1-\gamma)\Delta t w_0^2 & 1 - \frac{\gamma\Delta^2 t w_0^2}{1+\beta\Delta^2 t w_0^2} \end{bmatrix}$$

Donc on peut calculer $det(\lambda I - A)$

$$det(\lambda I - A) = \begin{vmatrix} \lambda - (1 - \frac{0.5\Delta^2 t w_0^2}{1+\beta\Delta^2 t w_0^2}) & \frac{-\Delta t}{1+\beta\Delta^2 t w_0^2} \\ \gamma\Delta t w_0^2(1 - \frac{0.5\Delta^2 t w_0^2}{1+\beta\Delta^2 t w_0^2}) + (1-\gamma)\Delta t w_0^2 & \lambda - (1 - \frac{\gamma\Delta^2 t w_0^2}{1+\beta\Delta^2 t w_0^2}) \end{vmatrix}$$

Quand $\gamma = 0.5, \beta = 0.25$, on a

$$det(\lambda I - A) = \begin{vmatrix} \lambda - \frac{1-0.25\Delta^2 t w_0^2}{1+0.25\Delta^2 t w_0^2} & \frac{-\Delta t}{1+0.25\Delta^2 t w_0^2} \\ \frac{\Delta t w_0^2}{1+0.25\Delta^2 t w_0^2} & \lambda - \frac{1-0.25\Delta^2 t w_0^2}{1+0.25\Delta^2 t w_0^2} \end{vmatrix}$$

Puis on peut obtenir les valeur propre:

$$\lambda_1 = 1 - \frac{0.5\Delta^2 t w_0^2}{1+0.25\Delta^2 t w_0^2} + i\frac{\Delta t w_0}{1+0.25\Delta^2 t w_0^2}$$

$$\lambda_2 = 1 - \frac{0.5\Delta^2 t w_0^2}{1+0.25\Delta^2 t w_0^2} - i\frac{\Delta t w_0}{1+0.25\Delta^2 t w_0^2}$$

On a vu $|\lambda| = 1$, Donc il est bien stable.

5.2) **Résolution avec un schéma de Newmark** $\gamma = 0.5, \beta = 0$

code python:

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

def calculate_E(q, omega, w0):
    E = 0.5 * (omega * omega + w0 * w0 * q * q)

def newmark(deltaT, T0, q0, omega0, w0, gamma=0.25, beta=0.5):
    etat_init = np.array([q0, omega0])
    etat_init = etat_init.reshape(-1, 1)
    B = np.array([[1 + beta * deltaT * deltaT * w0 * w0, 0],
                  [gamma * deltaT * w0 * w0, 1]])
    C = np.array([[1 - (0.5 - beta) * deltaT * deltaT * w0 * w0, deltaT],
                  [-(1 - gamma) * deltaT * w0 * w0, 1]])
    B_inv = np.linalg.inv(B)
    A = B_inv.dot(C)

    t_list = []
    q_list = []
    E_list = []

    i = 0
    etat_i = etat_init
    t_list.append(i * deltaT)
    q_list.append(etat_i[0][0])
    E_list.append(calculate_E(etat_i[0][0], etat_i[1][0], w0))

    while (i * deltaT < T0):
        etat_i = A.dot(etat_i)
        i = i + 1
        t_list.append(i * deltaT)
        q_list.append(etat_i[0][0])
        E_list.append(calculate_E(etat_i[0][0], etat_i[1][0], w0))

    return t_list, q_list, E_list

if __name__ == '__main__':
    dt1 = 0.01
    dt2 = 0.05
    dt3 = 1
    T0 = 3
    q0 = 1
    omega0 = 0
    w0 = 2 * np.pi

    gamma = 0.5
    beta = 0

    t_list4, q_list4, e_list4 = newmark(dt1, T0, q0, omega0, w0, gamma=gamma, beta=beta)

    plt.plot(t_list4, q_list4, color="C4", marker="x", label="Newmark, $\Delta t = {}$".format(dt1))
    plt.plot(t_list1, np.cos(2 * np.pi * np.array(t_list1)), color="C3",  label="solution exacte")

    plt.ylabel("q")
    plt.xlabel("t")
    plt.legend(loc="upper right")
    plt.show()
```

code matlab

calculate_E.m

```matlab
function E = calculate_E(q, omega, w0)
    E = 0.5 * (omega * omega + w0 * w0 * q * q);
end
```

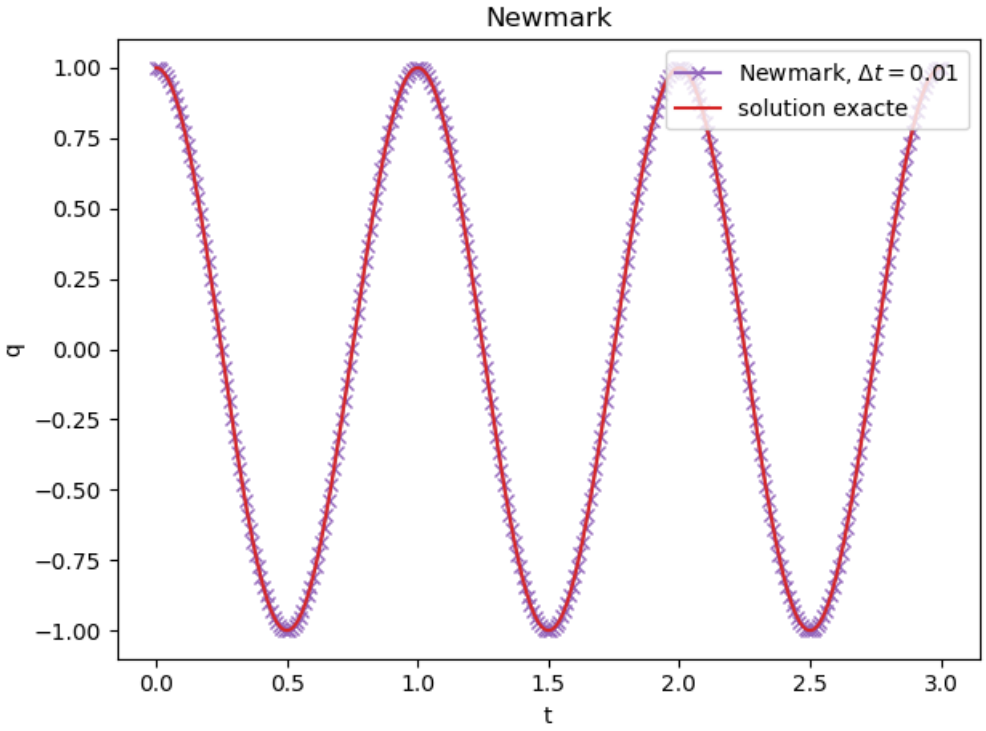
newmark.m

```
function [t_list, q_list, E_list] = newmark(deltaT, T0, q0, omega0, w0, gamma, beta)

etat_init = [q0, omega0];
etat_init = etat_init';
B = [1 + beta * deltaT * deltaT * w0 * w0, 0;
             gamma * deltaT * w0 * w0, 1];
C = [1 - (0.5 - beta) * deltaT * deltaT * w0 * w0, deltaT;
             -(1 - gamma) * deltaT * w0 * w0, 1];
B_inv = inv(B);
A = B_inv*C;

t_list = [];
q_list = [];
E_list = [];

i = 0;
etat_i = etat_init;
t_list = [t_list, i*deltaT];
q_list = [q_list, etat_i(1, 1)];
E_list = [E_list, calculate_E(etat_i(1, 1), etat_i(2, 1), w0)];


while (i * deltaT < T0)
    etat_i = A*etat_i;
    i = i + 1;
    t_list = [t_list, i*deltaT];
    q_list = [q_list, etat_i(1, 1)];
    E_list = [E_list, calculate_E(etat_i(1, 1), etat_i(2, 1), w0)];
end

end
```

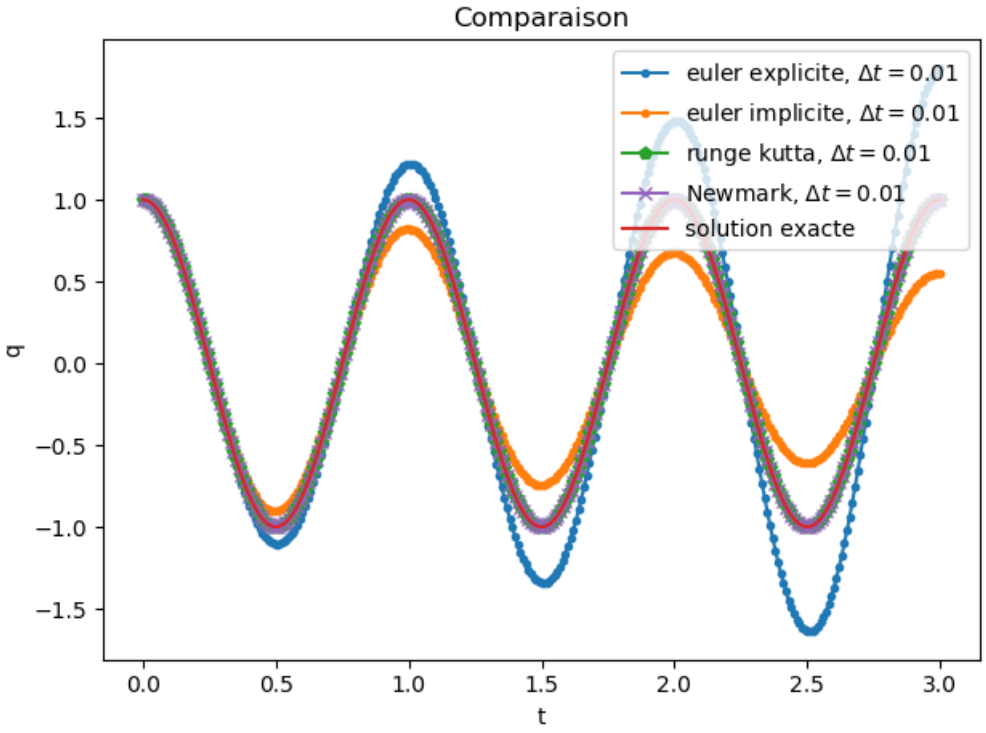
main.m

```
dt1 = 0.01;
dt2 = 0.05;
dt3 = 1;
T0 = 3;
q0 = 1;
omega0 = 0;
w0 = 2 * pi;

gamma = 0.5;
beta = 0;

[t_list4, q_list4, e_list4] = newmark(dt1, T0, q0, omega0, w0, gamma, beta);

plot(t_list4, q_list4, 'DisplayName', sprintf('Newmark, dt=%0.2f', dt1))
hold on
plot(t_list4, cos(2 * pi * t_list1), 'DisplayName', 'solution exacte')

ylabel('q')
xlabel('t')
title('Newmark')
```

On a la figure pour la méthode Newmark:

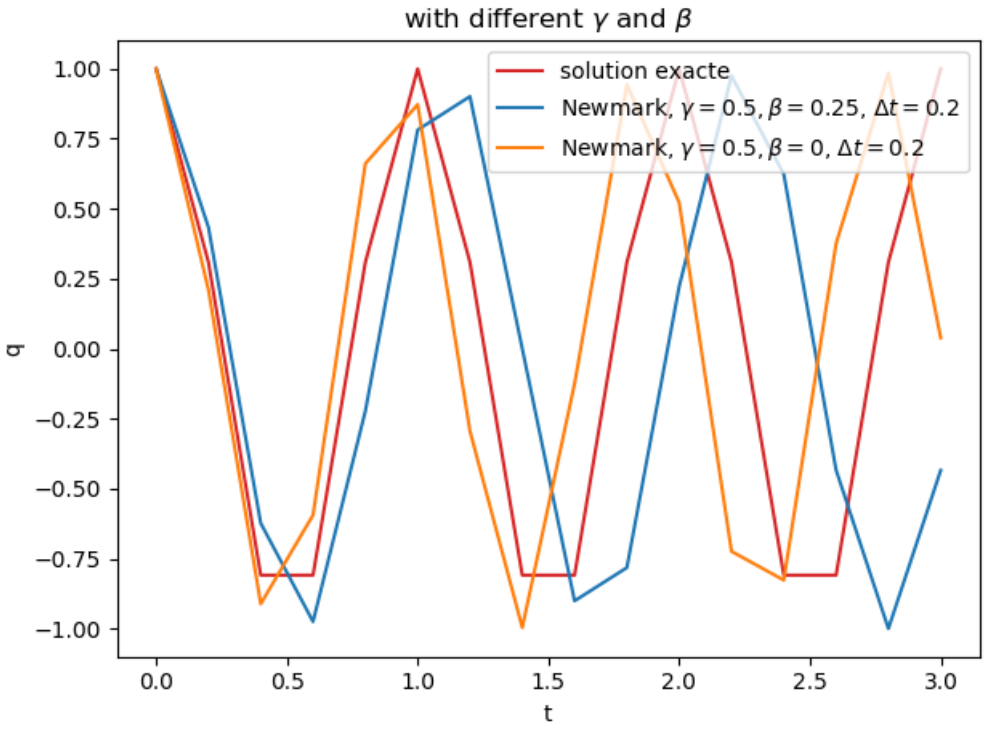


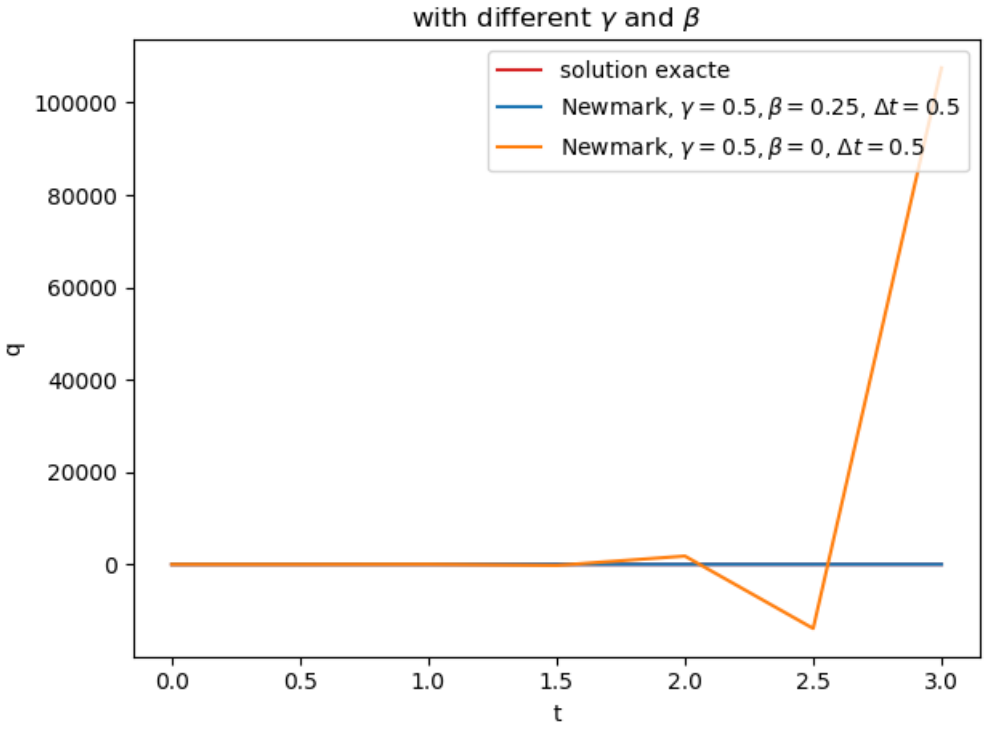
5.2.2) Comparaison avec autre methodes:



Il y a une boone superposition entre la Méthode Runge Kutta et Newmark et la solution standard.

5.2.3) Comparaison avec trois solutions: solution exacte, Newmark ($\gamma = 0.5, \beta = 0.25$) et Newmark(Newmark ($\gamma = 0.5, \beta = 0$))





On peut voir il y a des différences entre les trois solutions. Qunad $\Delta t = 0.2$, On a vu deux courbes oscillent autour de la courbe de la solution exacte. Mais quand $\Delta t = 0.5$, il y a une divergence pour la methode Newmark qui a $\beta = 0$.

5.2.4) D'après la question précédente, on a

$$A = B^{-1} \times C$$

$$A = \begin{bmatrix} 1 - \frac{0.5\Delta^2 t w_0^2}{1+\beta\Delta^2 t w_0^2} & \frac{\Delta t}{1+\beta\Delta^2 t w_0^2} \\ (-\gamma\Delta t w_0^2)(1 - \frac{0.5\Delta^2 t w_0^2}{1+\beta\Delta^2 t w_0^2}) - (1-\gamma)\Delta t w_0^2 & 1 - \frac{\gamma\Delta^2 t w_0^2}{1+\beta\Delta^2 t w_0^2} \end{bmatrix}$$

Donc on peut calculer $det(\lambda I - A)$

$$det(\lambda I - A) = \begin{vmatrix} \lambda - (1 - \frac{0.5\Delta^2 t w_0^2}{1+\beta\Delta^2 t w_0^2}) & \frac{-\Delta t}{1+\beta\Delta^2 t w_0^2} \\ \gamma\Delta t w_0^2(1 - \frac{0.5\Delta^2 t w_0^2}{1+\beta\Delta^2 t w_0^2}) + (1-\gamma)\Delta t w_0^2 & \lambda - (1 - \frac{\gamma\Delta^2 t w_0^2}{1+\beta\Delta^2 t w_0^2}) \end{vmatrix}$$

Quand $\gamma = 0.5, \beta = 0$, on a

$$det(\lambda I - A) = \begin{vmatrix} \lambda - (1 - 0.5\Delta^2 t w_0^2) & -\Delta t \\ 0.5\Delta t w_0^2(2 - 0.5\Delta^2 t w_0^2) & \lambda - (1 - 0.5\Delta^2 t w_0^2) \end{vmatrix}$$

Puis on peut obtenir les valeur propre:

$$\lambda = 1 - 0.5\Delta^2 t w_0^2 \pm i w_0 \Delta t \sqrt{1 - 0.25\Delta^2 t w_0^2}$$

Proposons que $\Delta t w_0 = r$

Donc on a

$$|\lambda| = (1 - 0.5r^2)^2 + r^2(1 - 0.25r^2) = 1$$

Donc il est suffit pour mettre r quelconque pour mettre le courbe stable. Donc le pas de temps critique est

$$\Delta t = \alpha \times \frac{2}{w_0}$$