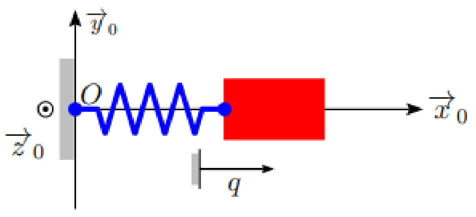
Séance 4 Dynamique des structures, Systems

> 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.

Oscillateur non linéaire à un degré de liberté



1 Newmark explicite

1.1) Les mouvements d'oscillations libres de la masse m sont des solutions de l'équation:

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

Et on a $\gamma = 0.5.\beta = 0$ pour le schéma Newmark explicite, donc

$$q_{j+1} = q_j + \Delta t \dot{q}_j + \frac{\Delta t^2}{2} \ddot{q}_j \tag{2}$$

$$\dot{q}_{j+1} = \dot{q}_j + \frac{\Delta t}{2}(\ddot{q}_j + \ddot{q}_{j+1}) \tag{3}$$

D'après l'équation (1), on a:

$$\ddot{q}_{j+1} = -w_0^2 q_{j+1}(1 + aq_{j+1}^2) \tag{4}$$

Donc on peut calculer état suivant en utilisant l'équation (2)(4) puis (3). Plus précisement, on a état entrée:q_j, $\dot{q}_j$, $\ddot{q}_j$, on utilise l'équation (2) pour obtenir q_{j+1}. D'après l'équation (4) , on peut avoir $\ddot{q}_{j+1}$. Finalement, on peut calculer $\dot{q}_{j+1}$ en utilisant l'équation (3).

1.2) Programmation en utilisant Python.

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


def compute_q_ddot(w0, a, q):
    """Equation de mouvement"""
    return -w0 * w0 * q * (1 + a * q * q)


def newmark(etat_init, w0, a, delta_t, T, mode="explicite"):
    t_list = []
    q_list = []

    i = 0
    etat_i = etat_init

    while (i * delta_t < T):
        t = i * delta_t
        t_list.append(t)
        q_list.append(etat_i[0])

        if mode == "explicite":
            q_iplus1 = etat_i[0] + delta_t * etat_i[1] + delta_t * delta_t / 2.0 * etat_i[2]
            q_ddot_iplus1 = compute_q_ddot(w0, a, q_iplus1)
            q_dot_iplus1 = etat_i[1] + delta_t / 2.0 * (etat_i[2] + q_ddot_iplus1)
        elif mode == "implicite":
            raise NotImplementedError
        else:
            raise NotImplementedError

        etat_i = [q_iplus1, q_dot_iplus1, q_ddot_iplus1]  # zip params in a list
        i = i + 1

    return t_list, q_list


if __name__ == '__main__':
    # initial params
    w0 = 2 * np.pi
    a = 0.1
    q0 = 2
    q0_dot = 0
    q0_ddot = compute_q_ddot(w0, a, q0)

    etat_init = [q0, q0_dot, q0_ddot]  # zip params in a list

    # default settings
    T0 = 6
    h = 0.02  # le pas de temps

    t_list, q_list = newmark(etat_init, w0, a, h, T0, mode="explicite")

    plt.plot(t_list, q_list, label="$\Delta t = %.2f$"%h)
    plt.ylabel("q")
    plt.xlabel("t")
    plt.title("Newmark explicite")
    plt.legend(loc="upper right")
    plt.show()
```

code matlab

```
compute_q_ddot.m
```

```
function q_ddot = compute_q_ddot(w0, a, q)

% Equation de mouvement
q_ddot = -w0 * w0 * q * (1 + a * q * q);

end
```

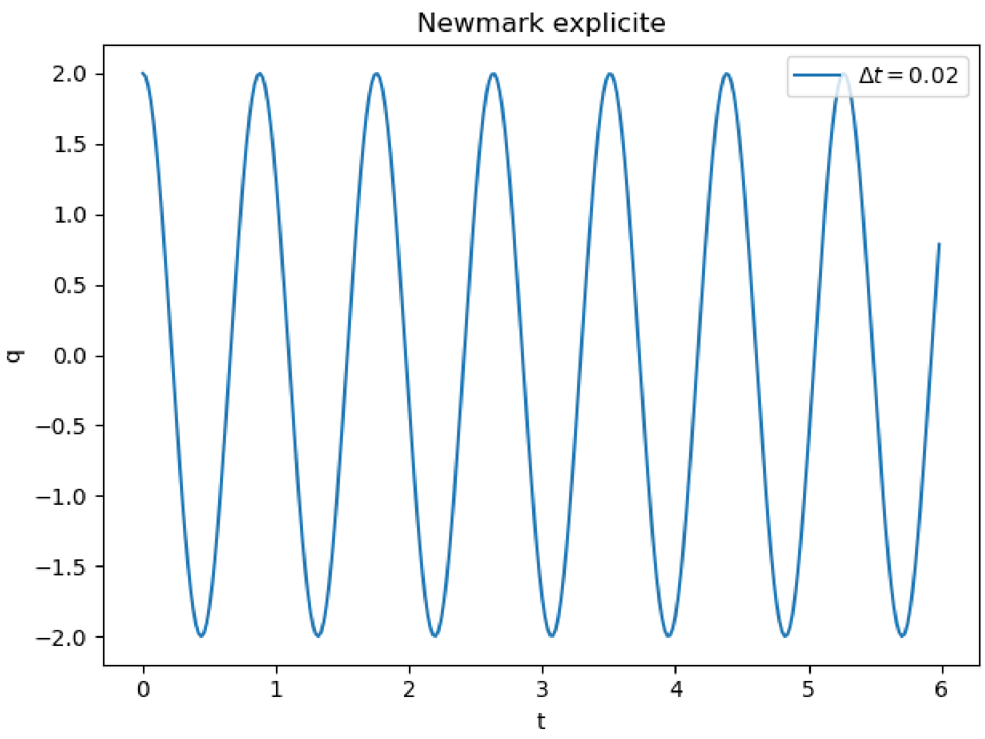
newmark_explicite.m

```
function [t_list, q_list] = newmark_explicite(etat_init, w0, a, delta_t, T)

t_list = [];
q_list = [];

i = 0;
etat_i = etat_init;

while (i * delta_t < T)
    t = i * delta_t;
    t_list = [t_list, t];
    q_list = [q_list, etat_i(1)];


    q_iplus1 = etat_i(1) + delta_t * etat_i(2) + delta_t * delta_t / 2.0 * etat_i(3);
    q_ddot_iplus1 = compute_q_ddot(w0, a, q_iplus1);
    q_dot_iplus1 = etat_i(2) + delta_t / 2.0 * (etat_i(3) + q_ddot_iplus1);


    etat_i = [q_iplus1, q_dot_iplus1, q_ddot_iplus1];  % zip params in a list
    i = i + 1;
end

end
```

main.m

```
% initial params
w0 = 2 * pi;
a = 0.1;
q0 = 2;
q0_dot = 0;
q0_ddot = compute_q_ddot(w0, a, q0);

etat_init = [q0, q0_dot, q0_ddot];  % zip params in a list

% default settings
T0 = 6;
h = 0.02 ; % le pas de temps

[t_list, q_list] = newmark_explicite(etat_init, w0, a, h, T0);

plot(t_list, q_list, 'DisplayName', sprintf('delta_t = %0.2f', h))
ylabel('q')
xlabel('t')
title('Newmark explicite')
legend
```

On peut déssiner la figure:



1.3)
Quand $t = 0$: $q(t) = 2$
Quand $t = \Delta t$: $q(t) = 1.9778920861415599$
Quand $t = 2\Delta t$: $q(t) = 1.912331781918763$
Quand $t = T_0$: $q(t) = 1.0329423474484474$

2. Newmark Implicite

2.1) Il est un peu compliqué pour utiliser Newmark Implicite. Mais on va le faire pas à pas. D'après Newmark, on a deux équation:

$$q_{j+1} = q_j + \Delta t \dot{q}_j + \Delta t^2 (0.5 - \beta) \ddot{q}_j + \beta \Delta t^2 \ddot{q}_{j+1} \tag{5}$$

$$\dot{q}_{j+1} = \dot{q}_j + \Delta t (1 - \gamma) \ddot{q}_j + \gamma \Delta t \ddot{q}_{j+1} \tag{6}$$

On a aussi l'équation de mouvement (1). Donc normalement on doit avoir:

$$\ddot{q}_{j+1} + w_0^2 q_{j+1} (1 + a q_{j+1}^2) = 0 \tag{7}$$

Parce que l'équation (5)(6) et (7) n'admet une solution analytique, on doit chcher une autre façon pour le calculer. En effet on propose des valeurs estimé et des corrections pour avoir les valeurs corrigé. Les valeurs estimé sont noté comme: $q_{j+1}^*, \dot{q}_{j+1}^*, \ddot{q}_{j+1}^*$, les corrections sont noté: $\Delta q_{j+1}, \Delta \dot{q}_{j+1}, \Delta \ddot{q}_{j+1}$. Donc les valeurs corrigé sont:

$$q_{j+1}^* + \Delta q_{j+1}, \dot{q}_{j+1}^* + \Delta \dot{q}_{j+1}, \ddot{q}_{j+1}^* + \Delta \ddot{q}_{j+1}$$

En fait, on veut les valeurs estimé proche de valeurs vrais à l'instant t_{j+1}, donc les état doit vérifier l'équation du mouvement. Donc il doit tends vers 0. Donc on veut minimiser la valeur absolue de la résidu:

$$|\ddot{q}_{j+1}^* + w_0^2 q_{j+1}^* (1 + a {q_{j+1}^*}^2)| \leq \epsilon$$

2.2) Pour les valeurs estimé, la relation (5) et (6) s'écrivent

$$q_{j+1}^* = q_j + \Delta t \dot{q}_j + \Delta t^2 (0.5 - \beta) \ddot{q}_j + \beta \Delta t^2 \ddot{q}_{j+1}^* \tag{8}$$

$$\dot{q}^*_{j+1} = \dot{q}_j + \Delta t(1-\gamma)\ddot{q}_j + \gamma\Delta t\ddot{q}^*_{j+1} \tag{9}$$

Pour les valeurs corrigé, la relation (5) et (6) s'écrivent

$$q^*_{j+1} + \Delta q_{j+1} = q_j + \Delta t\dot{q}_j + \Delta t^2(0.5-\beta)\ddot{q}_j + \beta\Delta t^2(\ddot{q}^*_{j+1} + \Delta\ddot{q}_{j+1}) \tag{10}$$

$$\dot{q}^*_{j+1} + \Delta\dot{q}_{j+1} = \dot{q}_j + \Delta t(1-\gamma)\ddot{q}_j + \gamma\Delta t(\ddot{q}^*_{j+1} + \Delta\ddot{q}_{j+1}) \tag{11}$$

On met (10) moins (8) on a:

$$\Delta q_{j+1} = \beta\Delta t^2\Delta\ddot{q}_{j+1} \tag{12}$$

On met (11) moins (9), on a :

$$\Delta\dot{q}_{j+1} = \gamma\Delta t\Delta\ddot{q}_{j+1} \tag{13}$$

Les valeurs corrigé vérifie l'équation du mouvement (7), donc on a

$$\ddot{q}^*_{j+1} + \Delta\ddot{q}_{j+1} + w_0^2(q^*_{j+1} + \Delta q_{j+1}) + w_0^2 a(q^*_{j+1} + \Delta q_{j+1})^3 = 0$$

En utilisant le développement limité au premier ordre on a :

$$\ddot{q}^*_{j+1} + \Delta\ddot{q}_{j+1} + w_0^2(q^*_{j+1} + \Delta q_{j+1}) + w_0^2 a(q^*_{j+1})^3 + 3w_0^2 a(q^*_{j+1})^2\Delta q_{j+1} = 0$$

Ce qui permet d'écrire:

$$\Delta\ddot{q}_{j+1} + w_0^2\Delta q_{j+1} + 3w_0^2 a(q^*_{j+1})^2\Delta q_{j+1} = -(\ddot{q}^*_{j+1} + w_0^2 q^*_{j+1} + w_0^2 a(q^*_{j+1})^3) \tag{14}$$

Donc on peut finalement trouver les trois corrections si la valeur absolue du résidu est supérieure à la précision souhaité.

$$\begin{cases} \Delta q_{j+1} = \beta\Delta t^2\Delta\ddot{q}_{j+1} \\ \Delta\dot{q}_{j+1} = \gamma\Delta t\Delta\ddot{q}_{j+1} \\ \Delta\ddot{q}_{j+1} + w_0^2\Delta q_{j+1} + 3w_0^2 a(q^*_{j+1})^2\Delta q_{j+1} = -(\ddot{q}^*_{j+1} + w_0^2 q^*_{j+1} + w_0^2 a(q^*_{j+1})^3) \end{cases} \tag{15}$$

2.3) En utilisant la définition de la première estimation des variable donnée, on peut le programmer.
code python:

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


def compute_q_ddot(w0, a, q):
    """Equation de mouvement"""
    return -w0 * w0 * q * (1 + a * q * q)


def compute_residu(w0, a, etat_estime):
    return np.abs(etat_estime[2] + w0 * w0 * etat_estime[0] * (1 + a * etat_estime[0] * etat_estime[0]))


def newmark(etat_init, w0, a, delta_t, T, gamma, beta, mode="explicite", epsilon=0.01):
    t_list = []
    q_list = []

    i = 0
    etat_i = etat_init

    while (i * delta_t <= T):
        t = i * delta_t
        t_list.append(t)
        q_list.append(etat_i[0])

        # if i in [0, 1, 2] or t == T:
        #     # fetch values
        #     print("Quand $t = {}\Delta t$: $q(t) = {}$".format(i, etat_i[0]))

        if mode == "explicite":
            q_ip1 = etat_i[0] + delta_t * etat_i[1] + delta_t * delta_t / 2.0 * etat_i[2]
            q_ddot_ip1 = compute_q_ddot(w0, a, q_ip1)
            q_dot_ip1 = etat_i[1] + delta_t / 2.0 * (etat_i[2] + q_ddot_ip1)
            etat_i = [q_ip1, q_dot_ip1, q_ddot_ip1]
        elif mode == "implicite":
            # initialize estimated values
            q_ddot_ip1_estime = 0
            q_dot_ip1_estime = etat_i[1] + delta_t * (1 - gamma) * etat_i[2]
            q_ip1_estime = etat_i[0] + delta_t * etat_i[1] + delta_t * delta_t * (0.5 - beta) * etat_i[2]
            etat_ip1_estime = [q_ip1_estime, q_dot_ip1_estime, q_ddot_ip1_estime]  # zip in a list

            # begin test
            while compute_residu(w0, a, etat_ip1_estime) > epsilon:
                # compute delta_q_ddot_ip1
                numerator = -(etat_ip1_estime[2] + (w0 ** 2) * etat_ip1_estime[0] + (w0 ** 2) * a * (
                        etat_ip1_estime[0] ** 3))
                denomiator = 1 + (w0 ** 2) * beta * (delta_t ** 2) + 3 * (w0 ** 2) * a * (
                        etat_ip1_estime[0] ** 2) * beta * (delta_t ** 2)
                delta_q_ddot_ip1 = numerator / denomiator

                # then we can compute other two deltas
                delta_q_ip1 = beta * (delta_t ** 2) * delta_q_ddot_ip1
                delta_q_dot_ip1 = gamma * delta_t * delta_q_ddot_ip1

                # update estimated values
                q_ip1_estime = etat_ip1_estime[0] + delta_q_ip1
                q_dot_ip1_estime = etat_ip1_estime[1] + delta_q_dot_ip1
                q_ddot_ip1_estime = etat_ip1_estime[2] + delta_q_ddot_ip1
                etat_ip1_estime = [q_ip1_estime, q_dot_ip1_estime, q_ddot_ip1_estime]  # zip in a list

            etat_i = etat_ip1_estime  # if satisfy precision, we go next step

        else:
            raise NotImplementedError

        i = i + 1

    return t_list, q_list
```

```python
if __name__ == '__main__':
    # initial params
    w0 = 2 * np.pi
    a = 0.1
    q0 = 2
    q0_dot = 0
    q0_ddot = compute_q_ddot(w0, a, q0)

    etat_init = [q0, q0_dot, q0_ddot]  # zip params in a list

    # default settings
    T0 = 6
    h = 0.02  # le pas de temps
    gamma = 0.5
    beta = 0.25
    mode = "implicite"

    t_list, q_list = newmark(etat_init, w0, a, h, T0, gamma, beta, mode=mode)

    plt.plot(t_list, q_list, label="$\Delta t = %.2f$" % h)
    plt.ylabel("q")
    plt.xlabel("t")
    plt.title("Newmark {}".format(mode))
    plt.legend(loc="upper right")
    plt.show()
```

code matlab

`compute_q_ddot.m`

```matlab
function q_ddot = compute_q_ddot(w0, a, q)

% Equation de mouvement
q_ddot = -w0 * w0 * q * (1 + a * q * q);

end
```

`compute_residu.m`

```matlab
function residu = compute_residu(w0, a, etat_estime)

residu = abs(etat_estime(3) + w0^2 * etat_estime(1) * (1 + a * etat_estime(1) * etat_estime(1)));

end
```

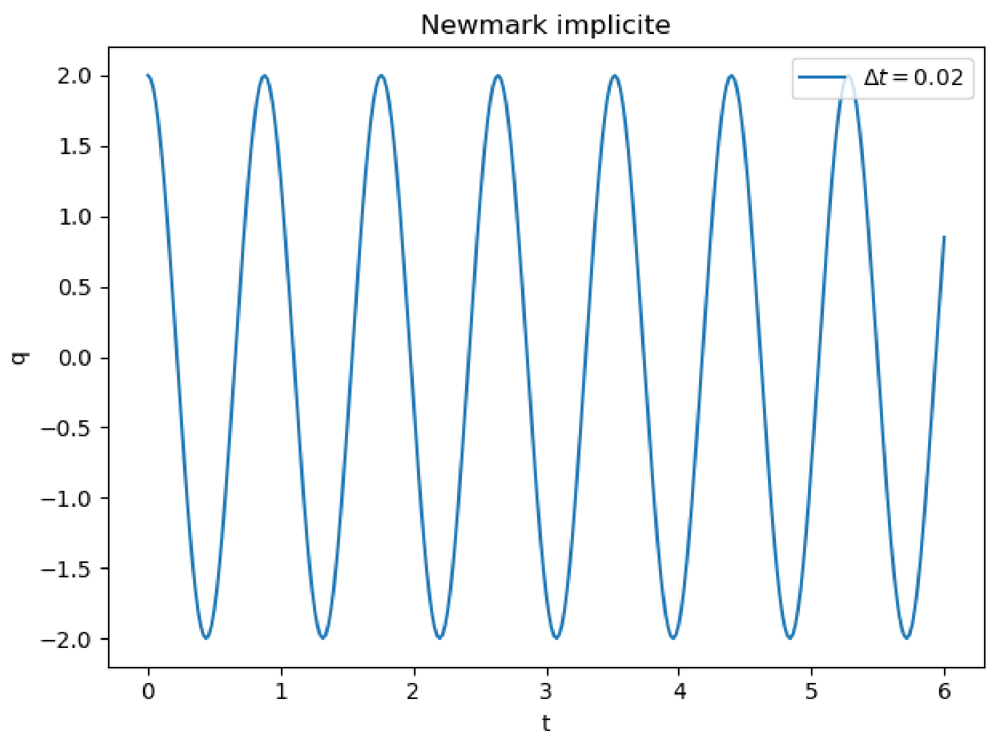
`newmark_implicite.m`

```
function [t_list, q_list] = newmark_implicite(etat_init, w0, a, delta_t, T, gamma, beta, precision)

t_list = [];
q_list = [];

i = 0;
etat_i = etat_init;

while (i * delta_t <= T)
    t = i * delta_t;
    t_list = [t_list, t];
    q_list = [q_list, etat_i(1)];

    % initialize estimated values
    q_ddot_ip1_estime = 0;
    q_dot_ip1_estime = etat_i(2) + delta_t * (1 - gamma) * etat_i(3);
    q_ip1_estime = etat_i(1) + delta_t * etat_i(2) + delta_t * delta_t * (0.5 - beta) * etat_i(3);
    etat_ip1_estime = [q_ip1_estime, q_dot_ip1_estime, q_ddot_ip1_estime;  % zip in a list

    % begin test
    while (compute_residu(w0, a, etat_ip1_estime) > precision)
        % compute delta_q_ddot_ip1
        numerator = -(etat_ip1_estime(3) + w0^2 * etat_ip1_estime(1) + w0^2 * a * etat_ip1_estime(1)^3);
        denominator = 1 + w0^2 * beta * delta_t^2 + 3 * w0^2 * a * etat_ip1_estime(1)^2 * beta * delta_t^2;
        delta_q_ddot_ip1 = numerator / denomiator;

        % then we can compute other two deltas
        delta_q_ip1 = beta * delta_t^2 * delta_q_ddot_ip1;
        delta_q_dot_ip1 = gamma * delta_t * delta_q_ddot_ip1;

        % update estimated values
        q_ip1_estime = etat_ip1_estime(1) + delta_q_ip1;
        q_dot_ip1_estime = etat_ip1_estime(2) + delta_q_dot_ip1;
        q_ddot_ip1_estime = etat_ip1_estime(3) + delta_q_ddot_ip1;
        etat_ip1_estime = [q_ip1_estime, q_dot_ip1_estime, q_ddot_ip1_estime];  % zip in a list
    end

    etat_i = etat_ip1_estime;  % if satisfy precision, we go next step
    i = i + 1;
end

end
```

main.m

```
% initial params
w0 = 2 * pi;
a = 0.1;
q0 = 2;
q0_dot = 0;
q0_ddot = compute_q_ddot(w0, a, q0);

etat_init = [q0, q0_dot, q0_ddot];  % zip params in a list

% default settings
T0 = 6;
h = 0.02;  % le pas de temps
gamma = 0.5;
beta = 0.25;

precision = 0.01; % residu precision

[t_list, q_list] = newmark_implicite(etat_init, w0, a, h, T0, gamma, beta, precision);

plot(t_list, q_list, 'DisplayName', sprintf('delta_t = %0.2f', h))
ylabel('q')
xlabel('t')
title('Newmark implicite')
legend
```

Ensuite on peut déssiner la figure:



2.4)
Quand $t = 0$: $q(t) = 2$
Quand $t = \Delta t$: $q(t) = 1.978081596728285$
Quand $t = 2\Delta t$: $q(t) = 1.9130640466175$
Quand $t = T_0$: $q(t) = 0.8477930530497348$

3 Energie mécanique

3.1) L'énergie mécanique pour cet oscillateur est composée de deux parties, une partie est l'énergie cinétique une autre énergie est l'énergie potentielle.
D'abord pour l'énergie potentielle, on peut le clculer:

$$\int_q^0 -kq - kaq^2 dq = \left[-\frac{k}{2}q^2 - \frac{ka}{3}q^3\right]_q^0 = \frac{k}{2}q^2 + \frac{ka}{3}q^3$$

Donc on a

$$E_p(m/R_0) = \frac{k}{2}q^2 + \frac{ka}{3}q^3 + Cte$$

Pour l'énergie cinétique on obtient:

$$E_c(m/R_0) = \frac{m\dot{q}^2}{2}$$

Donc son énergie mécanique est:

$$E = E_p(m/R_0) + E_c(m/R_0) = \frac{m\dot{q}^2}{2} + \frac{k}{2}q^2 + \frac{ka}{3}q^3 + Cte$$

3.2) Pour simplifier le calcule, on divise E par m, on obtient:

$$E^* = \frac{\dot{q}^2}{2} + \frac{w_0^2 q^2}{2} + \frac{w_0^2 a q^3}{3}$$

Ensuite on le programme et on a :

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


def compute_energie(w0, a, etat):
    return etat[1] * etat[1] / 2. + (w0 ** 2) * (etat[0] ** 2) / 2. + (w0 ** 2) * a * (etat[0] ** 3) / 3


def compute_q_ddot(w0, a, q):
    """Equation de mouvement"""
    return -w0 * w0 * q * (1 + a * q * q)


def compute_residu(w0, a, etat_estime):
    return np.abs(etat_estime[2] + w0 * w0 * etat_estime[0] * (1 + a * etat_estime[0] * etat_estime[0]))


def newmark(etat_init, w0, a, delta_t, T, gamma, beta, mode="explicite", epsilon=0.01):
    t_list = []
    q_list = []
    e_list = []

    i = 0
    etat_i = etat_init

    while (i * delta_t <= T):
        t = i * delta_t
        t_list.append(t)
        q_list.append(etat_i[0])
        e_list.append(compute_energie(w0, a, etat_i))

        if mode == "explicite":
            q_ip1 = etat_i[0] + delta_t * etat_i[1] + delta_t * delta_t / 2.0 * etat_i[2]
            q_ddot_ip1 = compute_q_ddot(w0, a, q_ip1)
            q_dot_ip1 = etat_i[1] + delta_t / 2.0 * (etat_i[2] + q_ddot_ip1)
            etat_i = [q_ip1, q_dot_ip1, q_ddot_ip1]
        elif mode == "implicite":
            # initialize estimated values
            q_ddot_ip1_estime = 0
            q_dot_ip1_estime = etat_i[1] + delta_t * (1 - gamma) * etat_i[2]
            q_ip1_estime = etat_i[0] + delta_t * etat_i[1] + delta_t * delta_t * (0.5 - beta) * etat_i[2]
            etat_ip1_estime = [q_ip1_estime, q_dot_ip1_estime, q_ddot_ip1_estime]  # zip in a list

            # begin test
            while compute_residu(w0, a, etat_ip1_estime) > epsilon:
                # compute delta_q_ddot_ip1
                numerator = -(etat_ip1_estime[2] + (w0 ** 2) * etat_ip1_estime[0] + (w0 ** 2) * a * (
                        etat_ip1_estime[0] ** 3))
                denomiator = 1 + (w0 ** 2) * beta * (delta_t ** 2) + 3 * (w0 ** 2) * a * (
                        etat_ip1_estime[0] ** 2) * beta * (delta_t ** 2)
                delta_q_ddot_ip1 = numerator / denomiator

                # then we can compute other two deltas
                delta_q_ip1 = beta * (delta_t ** 2) * delta_q_ddot_ip1
                delta_q_dot_ip1 = gamma * delta_t * delta_q_ddot_ip1

                # update estimated values
                q_ip1_estime = etat_ip1_estime[0] + delta_q_ip1
                q_dot_ip1_estime = etat_ip1_estime[1] + delta_q_dot_ip1
                q_ddot_ip1_estime = etat_ip1_estime[2] + delta_q_ddot_ip1
                etat_ip1_estime = [q_ip1_estime, q_dot_ip1_estime, q_ddot_ip1_estime]  # zip in a list

            etat_i = etat_ip1_estime  # if satisfy precision, we go next step

        else:
            raise NotImplementedError

        # if i in [0, 1, 2] or t == T:
```

```
        #     # fetch values
        #     print("Quand $t = {}\Delta t$: $q(t) = {}$".format(i, etat_i[0]))

        i = i + 1

    return t_list, q_list, e_list


if __name__ == '__main__':
    # initial params
    w0 = 2 * np.pi
    a = 0.1
    q0 = 2
    q0_dot = 0
    q0_ddot = compute_q_ddot(w0, a, q0)

    etat_init = [q0, q0_dot, q0_ddot]  # zip params in a list

    # default settings
    T0 = 6
    h = 0.02  # le pas de temps
    gamma = 0.5
    beta = 0.25

    t_list, q_list, e_list = newmark(etat_init, w0, a, h, T0, gamma, beta, mode="explicite")
    t_list2, q_list2, e_list2 = newmark(etat_init, w0, a, h, T0, gamma, beta, mode="implicite")


    plt.plot(t_list, e_list, label="Newmark explicite, $\Delta t = %.2f$" % h)
    plt.plot(t_list2, e_list2, label="Newmark implicite, $\Delta t = %.2f$" % h)
    # plt.ylabel("q")
    plt.ylabel("$E^{*}$")
    plt.xlabel("t")
    plt.title("Energie mecanique")
    plt.legend(loc="upper right")
    plt.show()
```

code matlab

`compute_energie.m`

```
function E = compute_energie(w0, a, etat)
E =  etat(2) * etat(2) / 2.0 + w0 ^ 2 * etat(1)^2 / 2.0 + w0^2 * a * etat(1)^ 3 / 3;
end
```

`compute_q_ddot.m`

```
function q_ddot = compute_q_ddot(w0, a, q)

% Equation de mouvement
q_ddot = -w0 * w0 * q * (1 + a * q * q);

end
```

`compute_residu.m`

```
function residu = compute_residu(w0, a, etat_estime):

residu = abs(etat_estime(3) + w0 * w0 * etat_estime(1) * (1 + a * etat_estime(1) * etat_estime(1)));

end
```

`newmark_explicite.m`

```matlab
function [t_list, q_liste, e_list] = newmark_explicite(etat_init, w0, a, delta_t, T)

t_list = [];
q_list = [];
e_list = [];

i = 0;
etat_i = etat_init;

while (i * delta_t < T)
    t = i * delta_t;
    t_list = [t_list, t];
    q_list = [q_list, etat_i(1)];
    e_list = [e_list, compute_energie(w0, a, etat_i)];


    q_iplus1 = etat_i(1) + delta_t * etat_i(2) + delta_t * delta_t / 2.0 * etat_i(3);
    q_ddot_iplus1 = compute_q_ddot(w0, a, q_iplus1);
    q_dot_iplus1 = etat_i(2) + delta_t / 2.0 * (etat_i(3) + q_ddot_iplus1);


    etat_i = [q_iplus1, q_dot_iplus1, q_ddot_iplus1];  % zip params in a list
    i = i + 1;
end

end
```

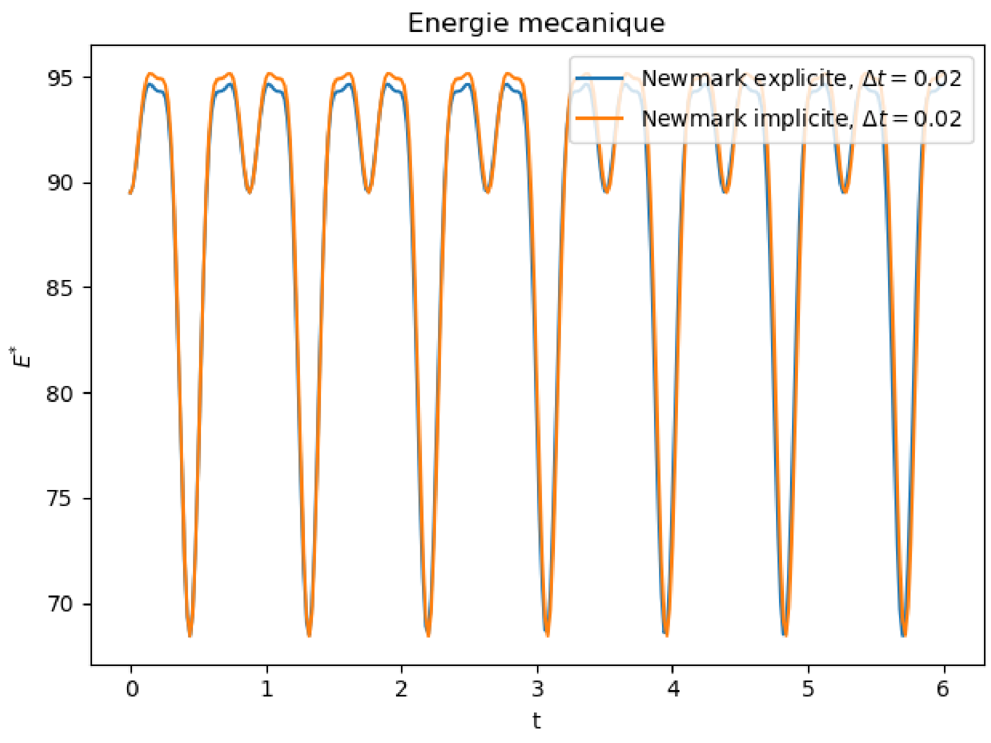
newmark_implicite.m

```
function [t_list, q_list, e_list] = newmark_implicite(etat_init, w0, a, delta_t, T, gamma, beta, precision)

t_list = [];
q_list = [];
e_list = [];

i = 0;
etat_i = etat_init;

while (i * delta_t <= T)
    t = i * delta_t;
    t_list = [t_list, t];
    q_list = [q_list, etat_i(1)];
    e_list = [e_list, compute_energie(w0, a, etat_i)]

    % initialize estimated values
    q_ddot_ip1_estime = 0;
    q_dot_ip1_estime = etat_i(2) + delta_t * (1 - gamma) * etat_i(3);
    q_ip1_estime = etat_i(1) + delta_t * etat_i(2) + delta_t * delta_t * (0.5 - beta) * etat_i(3);
    etat_ip1_estime = [q_ip1_estime, q_dot_ip1_estime, q_ddot_ip1_estime;  % zip in a list

    % begin test
    while (compute_residu(w0, a, etat_ip1_estime) > precision)
        % compute delta_q_ddot_ip1
        numerator = -(etat_ip1_estime(3) + w0^2 * etat_ip1_estime(1) + w0^2 * a * etat_ip1_estime(1)^3);
        denomiator = 1 + w0^2 * beta * delta_t^2 + 3 * w0^2 * a * etat_ip1_estime(1)^2 * beta * delta_t^2;
        delta_q_ddot_ip1 = numerator / denomiator;

        % then we can compute other two deltas
        delta_q_ip1 = beta * delta_t^2 * delta_q_ddot_ip1;
        delta_q_dot_ip1 = gamma * delta_t * delta_q_ddot_ip1;

        % update estimated values
        q_ip1_estime = etat_ip1_estime(1) + delta_q_ip1;
        q_dot_ip1_estime = etat_ip1_estime(2) + delta_q_dot_ip1;
        q_ddot_ip1_estime = etat_ip1_estime(3) + delta_q_ddot_ip1;
        etat_ip1_estime = [q_ip1_estime, q_dot_ip1_estime, q_ddot_ip1_estime];  % zip in a list
    end

    etat_i = etat_ip1_estime;  % if satisfy precision, we go next step
    i = i + 1;
end

end
```

main.m

```
% initial params
w0 = 2 * pi;
a = 0.1;
q0 = 2;
q0_dot = 0;
q0_ddot = compute_q_ddot(w0, a, q0);

etat_init = [q0, q0_dot, q0_ddot];  % zip params in a list

% default settings
T0 = 6;
h = 0.02;  % le pas de temps
gamma = 0.5;
beta = 0.25;

[t_list, q_list, e_list] = newmark_explicite(etat_init, w0, a, h, T0);
[t_list2, q_list2, e_list2] = newmark_implicite(etat_init, w0, a, h, T0, gamma, beta, precision);


plot(t_list, e_list, 'DisplayName', sprintf('Newmark explicite, delta_t = %0.2f', h;
hold on
plot(t_list2, e_list2, 'DisplayName', sprintf('Newmark implicite, delta_t = %0.2f', h));
hold on

ylabel('E*')
xlabel('t')
title('Energie mecanique')
hold off
legend
```

3.3) On a la figure d’après le code. On voit l’énergie mécanique ne reste pas identique. Elle varie périodiquement.


Energie mecanique
Newmark explicite, $\Delta t = 0.02$
Newmark implicite, $\Delta t = 0.02$
E^*
t