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.

Etude d'un oscillateur linéaire amorti à un degré de liberté

On a la formule de x, d'après les paramètres qu'on a pris, on peut déssiner la figure de la solutions analytique.

Code python:

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


def solution_analytique(x0, x0_dot, epsilon, w0, T, delta_t):
    t_list = np.arange(0, T, delta_t)

    # exp(-epsilon*w0*t)
    outside = np.exp(-epsilon * w0 * t_list)
    omega = w0 * np.sqrt(1 - epsilon * epsilon)

    # inside parentheses
    term1 = x0 * np.cos(omega * t_list)
    term2 = ((epsilon * w0 * x0 + x0_dot) / omega) * np.sin(omega * t_list)

    x_list = outside * (term1 + term2)
    return t_list, x_list


if __name__ == '__main__':
    x0 = 0.01
    x0_dot = 0
    t0 = 1
    delta_t = 0.01
    epsilon = 0.02
    w0 = 2 * np.pi / t0

    T = 10 * t0     # we consider only (0, T)

    t_list, x_list = solution_analytique(x0, x0_dot, epsilon, w0, T, delta_t)

    plt.plot(t_list, x_list,
             color = "C3",
             label="solution analytique, $\Delta t$ = {}".format(delta_t))
    plt.title("Solution analytique")
    plt.xlabel("t")
    plt.ylabel("x")
    plt.legend(loc="upper right")
    plt.show()
```

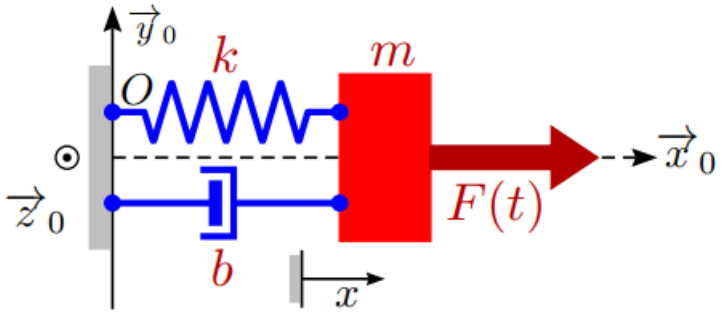
code matlab

```
solution_analytique.m
```

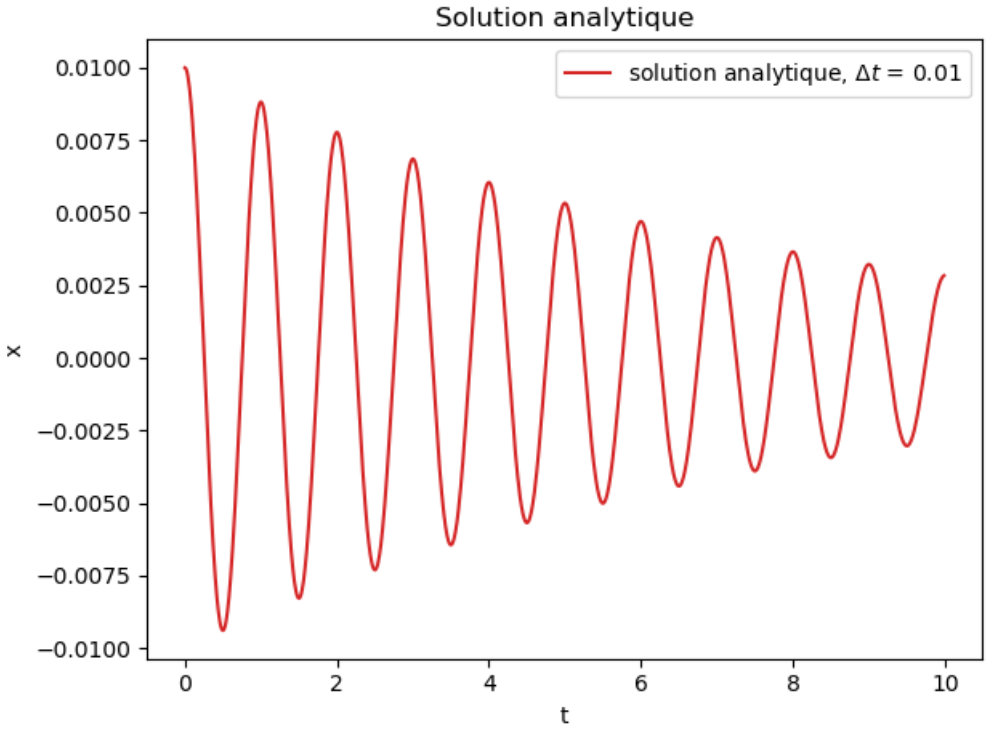
```
function [t_list, x_list] =  solution_analytique(x0, x0_dot, epsilon, w0, T, delta_t)

t_list = 0:delta_t:T;


% exp(-epsilon*w0*t)
outside = exp(-epsilon * w0 * t_list);
omega = w0 * sqrt(1 - epsilon * epsilon);

% inside parentheses
term1 = x0 * cos(omega * t_list);
term2 = ((epsilon * w0 * x0 + x0_dot) / omega) * sin(omega * t_list);

x_list = outside * (term1 + term2);

end
```

main.m

```
x0 = 0.01;
x0_dot = 0;
t0 = 1;
delta_t = 0.01;
epsilon = 0.02;
w0 = 2 * pi / t0;

T = 10 * t0 ;    % we consider only (0, T)

[t_list, x_list] = solution_analytique(x0, x0_dot, epsilon, w0, T, delta_t);

plot(t_list, x_list, 'DisplayName', sprintf('solution exacte, dt=%0.2f', delta_t))
title('Solution analytique')
xlabel('t')
ylabel('x')
hold off
legend
```



La figure du solution analytique est:



Euler Explicite

Dans le cas où $F(t) = 0$, on a l'equation (1)

$$\ddot{x} + 2\epsilon w_0 \dot{x} + w_0^2 x = 0 \tag{1}$$

Il faut d'abord cherchedr la matrice d'amplification.

D'après la relation:

$$\begin{bmatrix} x_{j+1} \\ \dot{x}_{j+1} \end{bmatrix} = \begin{bmatrix} x_j \\ \dot{x}_j \end{bmatrix} + \Delta t \times \begin{bmatrix} \dot{x}_j \\ \ddot{x}_j \end{bmatrix}$$

. On a deux equations:

$$x_{j+1} = x_j + \Delta t \times \dot{x}_j \tag{2}$$

$$\dot{x}_{j+1} = \dot{x}_j + \Delta t \times \ddot{x}_j \tag{3}$$

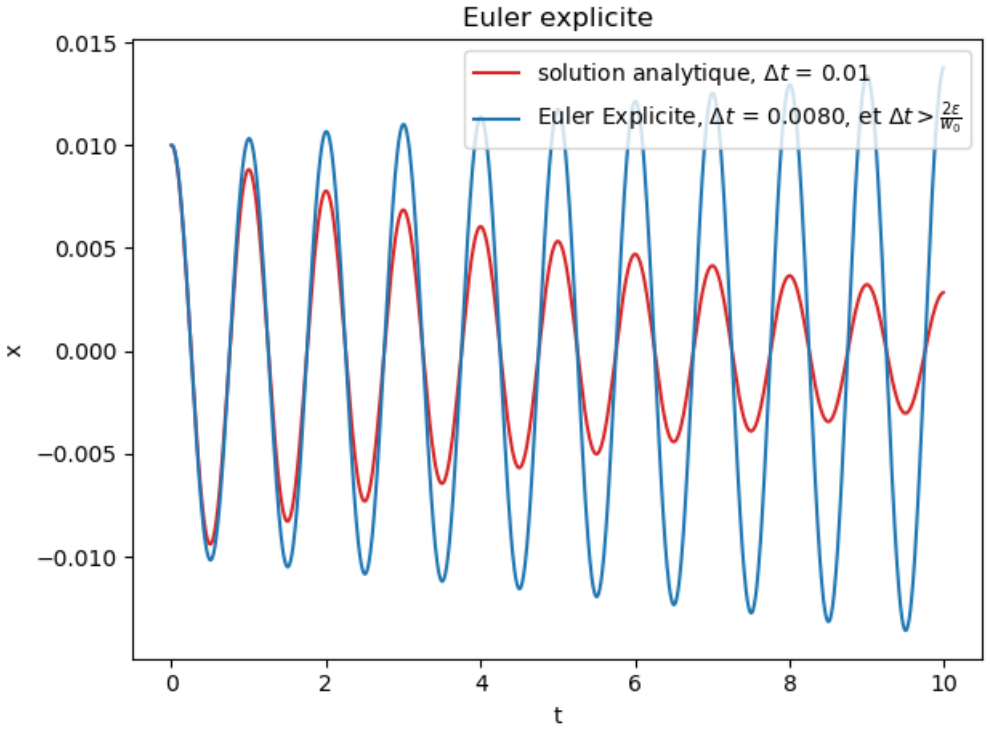
D'après l'équation (1) on a l'équation (4):

$$\ddot{x}_{j+1} = -w_0^2 \Delta t x_j + (1 - 2\epsilon w_0 \Delta t)\dot{x}_j \tag{4}$$

Donc on a la matrice d'amplification:

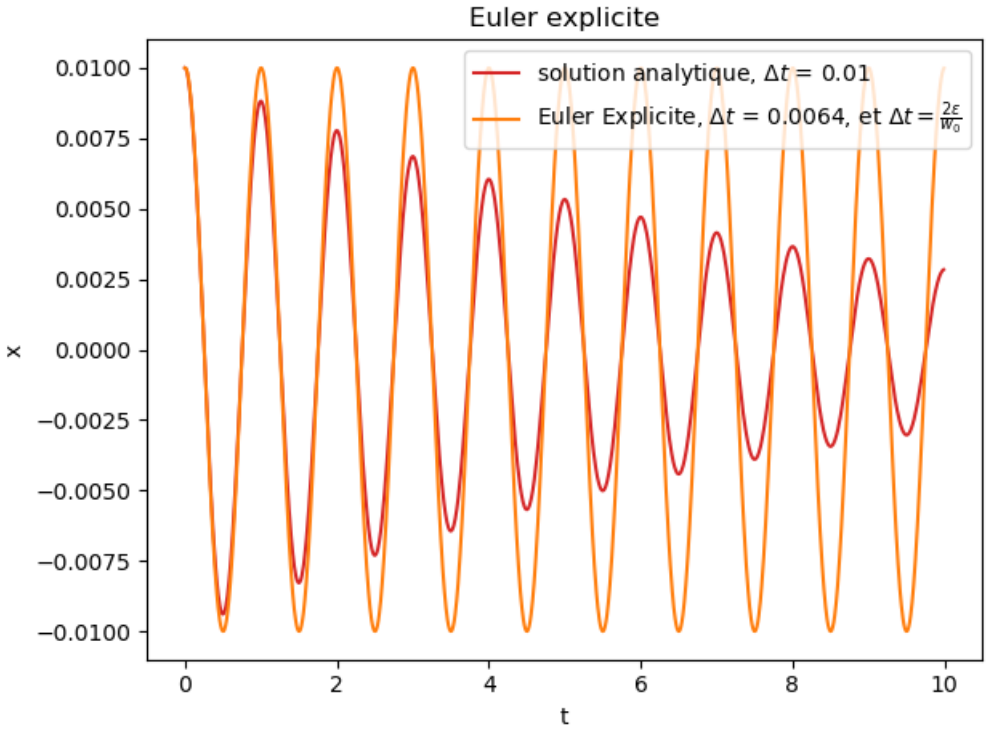
$$A = \begin{bmatrix} 1 & \Delta t \\ -w_0^2 \Delta t & 1 - 2\epsilon w_0 \Delta t \end{bmatrix}$$

1.1.a) Quand on'a choisit $\Delta t > \frac{2\epsilon}{w_0}$, on a la figure ci-dessous.



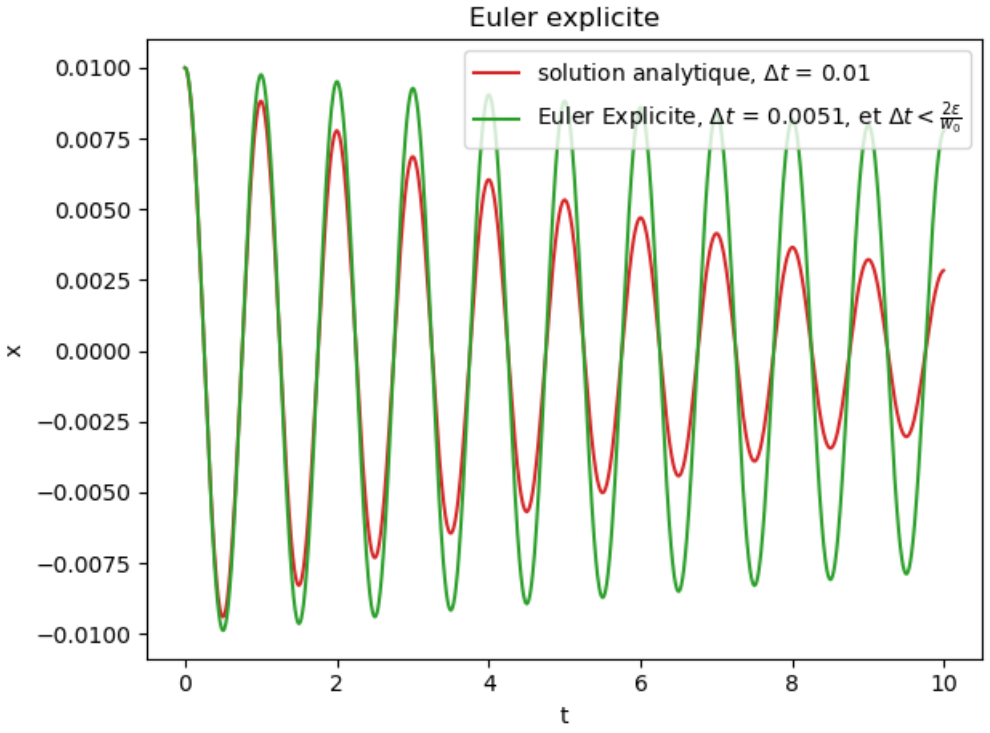
On a vu une grande divergence comparé avec la solution analytique. Donc il n'est pas stable.

1.1.b) Quand on'a choisit $\Delta t = \frac{2\epsilon}{w_0}$, on a la figure ci-dessous.



On a vu la courbe est bien stable.

1.1.c) Quand on'a choisit $\Delta t = 0.8 \times \frac{2\epsilon}{w_0}$, on a la figure ci-dessous.



On a vu le courbe est atténué mais existe encore une divergence comparé avec la solution analytique.

1.1.d) Il faut d'abord chercher les valeurs propres de la matrice d'amplification.

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

On sait que $Ax = \lambda x$, λ est valeur propre et x est vecteur propre. on a $(\lambda I - A)x = 0$.
Ensuite il faut calculer le determinant.

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

Donc

$$\begin{aligned} det(\lambda I - A) &= (\lambda - 1)(\lambda - 1 + 2\epsilon w_0 \Delta t) + w_0^2 \Delta^2 t \\ &= r^2 + 2\epsilon w_0 \Delta t r + w_0^2 \Delta^2 t \end{aligned}$$

où $r = \lambda - 1$

on a $b^2 - 4ac = 4w_0^2 \Delta^2 t(\epsilon^2 - 1) < 0$. Donc les solution est

$$r = -\epsilon w_0 \Delta t \pm i w_0 \Delta t \sqrt{1 - \epsilon^2}$$

Donc

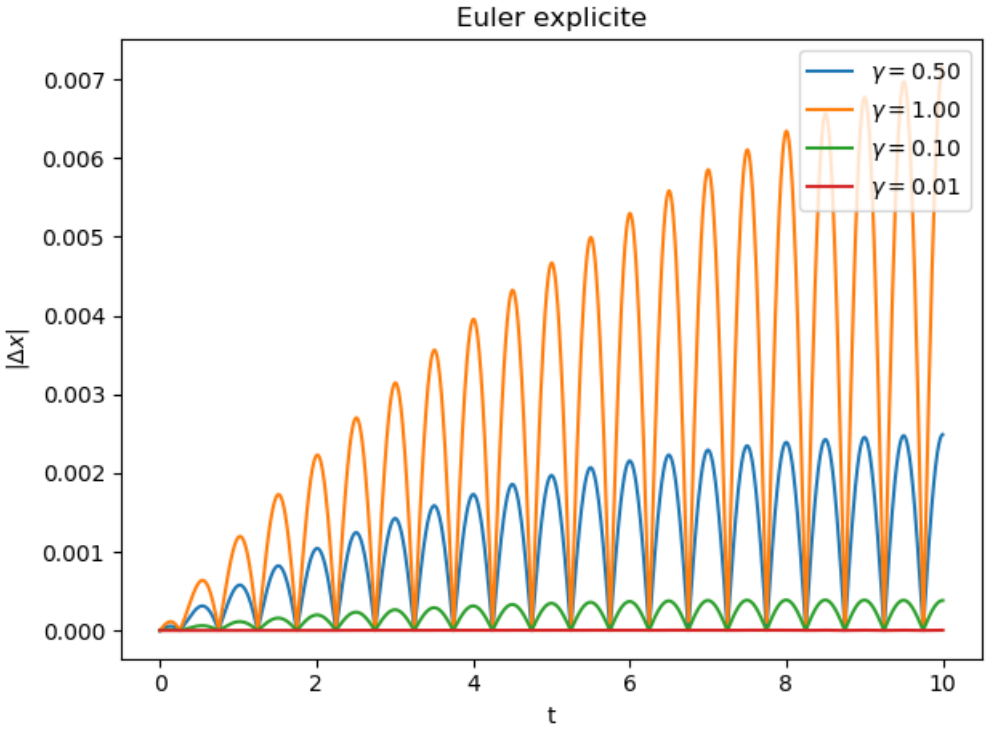
$$\lambda = 1 - \epsilon w_0 \Delta t \pm i w_0 \Delta t \sqrt{1 - \epsilon^2}$$

Ensuite la module de λ vaut:

$$|\lambda| = 1 + w_0^2 \Delta^2 t - 2\epsilon w_0 \Delta t$$

Donc si $|\lambda| > 1$ on a $\Delta t > \frac{2\epsilon}{w_0}$, si $|\lambda| = 1$ on a $\Delta t = \frac{2\epsilon}{w_0}$ et $|\lambda| < 1$ on a $\Delta t < \frac{2\epsilon}{w_0}$. Donc Δt est bien le critère qui permetre d'étudier la stabilité. Donc il est aussi important pour étudier la précision de la solution.

Quand $|\lambda| < 1$, la solution est toujours stable, donc plus le pas de temps est petit, la précision est plus haute.On étude certains valeur pour déterminer la bonne précision. On propose $\Delta t = \gamma \frac{2\epsilon}{w_0}$



On peut voir très clairement quand plus γ est petit, la précision est plus haute.

code python:

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


def solution_analytique(x0, x0_dot, epsilon, w0, T, delta_t):
    t_list = np.arange(0, T, delta_t)

    # exp(-epsilon*w0*t)
    outside = np.exp(-epsilon * w0 * t_list)
    omega = w0 * np.sqrt(1 - epsilon * epsilon)

    # inside parentheses
    term1 = x0 * np.cos(omega * t_list)
    term2 = ((epsilon * w0 * x0 + x0_dot) / omega) * np.sin(omega * t_list)

    x_list = outside * (term1 + term2)
    return t_list, x_list


def euler(x0, x0_dot, epsilon, w0, T, delta_t, mode="explicite"):
    if mode == "explicite":
        A = np.array([[1, delta_t],
                      [-w0 * w0 * delta_t, 1 - 2 * epsilon * w0 * delta_t]])
    elif mode == "implicite":
        raise NotImplementedError
    else:
        raise NotImplementedError("Unknown mode: {}".format(mode))

    etat_init = np.array([x0, x0_dot])
    # transpose
    etat_init = etat_init.reshape(-1, 1)

    i = 0
    x_list = []
    t_list = []
    etat_i = etat_init

    while (i * delta_t < T):
        t_list.append(i * delta_t)
        x_list.append(etat_i[0][0])
        i = i + 1
        etat_i = A.dot(etat_i)

    return np.array(t_list), np.array(x_list)


if __name__ == '__main__':
    x0 = 0.01
    x0_dot = 0
    t0 = 1
    epsilon = 0.02
    w0 = 2 * np.pi / t0

    # delta_t = 0.01

    dt2 = 2 * epsilon / w0
    dt1 = 0.5 * dt2
    dt3 = 0.1 * dt2
    dt4 = 0.01 * dt3
    T = 10 * t0  # we consider only (0, T)

    t_list, x_list = solution_analytique(x0, x0_dot, epsilon, w0, T, dt1)
    t_list1, x_list1 = euler(x0, x0_dot, epsilon, w0, T, dt1, mode="explicite")
    erreur_list1 = np.abs(x_list1 - x_list)

    t_list, x_list = solution_analytique(x0, x0_dot, epsilon, w0, T, dt2)
    t_list2, x_list2 = euler(x0, x0_dot, epsilon, w0, T, dt2, mode="explicite")
    erreur_list2 = np.abs(x_list2 - x_list)
```

```
    t_list, x_list = solution_analytique(x0, x0_dot, epsilon, w0, T, dt3)
    t_list3, x_list3 = euler(x0, x0_dot, epsilon, w0, T, dt3, mode="explicite")
    erreur_list3 = np.abs(x_list3 - x_list)

    t_list, x_list = solution_analytique(x0, x0_dot, epsilon, w0, T, dt4)
    t_list4, x_list4 = euler(x0, x0_dot, epsilon, w0, T, dt4, mode="explicite")
    erreur_list4 = np.abs(x_list4 - x_list)

    plt.plot(t_list1, erreur_list1, label="$\gamma = %.2f$" % (0.5))
    plt.plot(t_list2, erreur_list2, label="$\gamma = %.2f$" % (1.0))
    plt.plot(t_list3, erreur_list3, label="$\gamma = %.2f$" % (0.1))
    plt.plot(t_list4, erreur_list4, label="$\gamma = %.2f$" % (0.01))

    plt.title("Euler explicite")
    plt.xlabel("t")
    plt.ylabel("$|\Delta x|$")
    plt.legend(loc="upper right")
    plt.show()
```

code matlab

solution_analytique.m

```
function [t_list, x_list] =  solution_analytique(x0, x0_dot, epsilon, w0, T, delta_t)

t_list = 0:delta_t:T;

% exp(-epsilon*w0*t)
outside = exp(-epsilon * w0 * t_list);
omega = w0 * sqrt(1 - epsilon * epsilon);

% inside parentheses
term1 = x0 * cos(omega * t_list);
term2 = ((epsilon * w0 * x0 + x0_dot) / omega) * sin(omega * t_list);

x_list = outside * (term1 + term2);

end
```

euler_explicite.m

```
function [t_list, x_list] = euler_explicite(x0, x0_dot, epsilon, w0, T, delta_t)
A = [1, delta_t;-w0 * w0 * delta_t, 1 - 2 * epsilon * w0 * delta_t];


etat_init = [x0, x0_dot];
% transpose;
etat_init = etat_init';

i = 0;
x_list = [];
t_list = [];
etat_i = etat_init;

while (i * delta_t < T)
    t_list = [t_list, i*delta_t];
    x_list = [x_list, etat_i(1, 1)];
    i = i + 1;
    etat_i = A*etat_i;
end
end
```

main.m

```
x0 = 0.01;
x0_dot = 0;
t0 = 1;
epsilon = 0.02;
w0 = 2 * pi / t0;

% delta_t = 0.01;

dt2 = 2 * epsilon / w0;
dt1 = 0.5 * dt2;
dt3 = 0.1 * dt2;
dt4 = 0.01 * dt3;
T = 10 * t0;  % we consider only (0, T)

[t_list, x_list] = solution_analytique(x0, x0_dot, epsilon, w0, T, dt1);
[t_list1, x_list1] = euler_explicite(x0, x0_dot, epsilon, w0, T, dt1);
erreur_list1 = abs(x_list1 - x_list);

[t_list, x_list] = solution_analytique(x0, x0_dot, epsilon, w0, T, dt2);
[t_list2, x_list2] = euler_explicite(x0, x0_dot, epsilon, w0, T, dt2);
erreur_list2 = abs(x_list2 - x_list);

[t_list, x_list] = solution_analytique(x0, x0_dot, epsilon, w0, T, dt3);
[t_list3, x_list3] = euler_explicite(x0, x0_dot, epsilon, w0, T, dt3);
erreur_list3 = abs(x_list3 - x_list);

[t_list, x_list] = solution_analytique(x0, x0_dot, epsilon, w0, T, dt4);
[t_list4, x_list4] = euler_explicite(x0, x0_dot, epsilon, w0, T, dt4);
erreur_list4 = abs(x_list4 - x_list);

plot(t_list1, erreur_list1, 'DisplayName', sprintf('gamma=%0.2f', 0.5))
hold on
plot(t_list2, erreur_list2, 'DisplayName', sprintf('gamma=%0.2f', 1.0))
hold on
plot(t_list3, erreur_list3, 'DisplayName', sprintf('gamma=%0.2f', 0.1))
hold on
plot(t_list4, erreur_list4, 'DisplayName', spritnf('gamma=%0.2f', 0.01))

title('Euler explicite')
xlabel('t')
ylabel('delta x')
hold off
legend
```

Euler Implicite

1.2) Il faut chercher d'abord la matrice d'amplification:

Pour le schéma Euler implicite, on sait que

$$x_{j+1} = x_j + \Delta t \dot{x}_{j+1}$$

$$\dot{x}_{j+1} = \dot{x}_j + \Delta t \ddot{x}_{j+1}$$

On a aussi:

$$\ddot{x}_{j+1} = -2\epsilon w_0 \dot{x}_{j+1} - w_0^2 x_{j+1}$$

Donc on a

$$\begin{aligned} x_{j+1} &= x_j + \Delta t \dot{x}_{j+1} \\ &= x_j + \Delta t(\dot{x}_j + \Delta t \ddot{x}_{j+1}) \\ &= x_j + \Delta t \dot{x}_j + \Delta^2 t \ddot{x}_{j+1} \\ &= x_j + \Delta t \dot{x}_j - 2\epsilon w_0 \Delta^2 t \dot{x}_{j+1} - w_0^2 \Delta^2 t x_{j+1} \end{aligned}$$

Donc on a

$$(1 + w_0^2 \Delta^2 t) x_{j+1} + 2\epsilon w_0 \Delta^2 t \dot{x}_{j+1} = x_j + \Delta t \dot{x}_j \tag{5}$$

$$\begin{aligned} \dot{x}_{j+1} &= \dot{x}_j + \Delta t \ddot{x}_{j+1} \\ &= \dot{x}_j + \Delta t(-2\epsilon w_0 \dot{x}_{j+1} - w_0^2 x_{j+1}) \end{aligned}$$

Donc on a

$$w_0^2 \Delta t x_{j+1} + (1 + 2\epsilon w_0 \Delta t) \dot{x}_{j+1} = \dot{x}_j \tag{6}$$

D'après l'équation (5) et (6), on a l'equation:

$$\begin{bmatrix} 1 + w_0^2 \Delta^2 t & 2\epsilon w_0 \Delta^2 t \\ w_0^2 \Delta t & 1 + 2\epsilon w_0 \Delta t \end{bmatrix} \begin{bmatrix} x_{j+1} \\ \dot{x}_{j+1} \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_j \\ \dot{x}_j \end{bmatrix}$$

On le note

$$B \begin{bmatrix} x_{j+1} \\ \dot{x}_{j+1} \end{bmatrix} = C \begin{bmatrix} x_j \\ \dot{x}_j \end{bmatrix}$$

Donc la matrice d'amplification $A = B^{-1}C$

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

On a

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

et

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

Donc on peut calculer la matrice d'amplification:

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

Ensuiste on cherche valeurs propres:

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

On note

$$\beta = \frac{1}{1 + 2\epsilon w_0 \Delta t + w_0^2 \Delta^2 t}$$

Puis on a

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

et on note $r = \lambda - \beta$ donc

$$r^2 - 2\epsilon w_0 \Delta t \beta r + w_0^2 \Delta t^2 \beta = 0$$

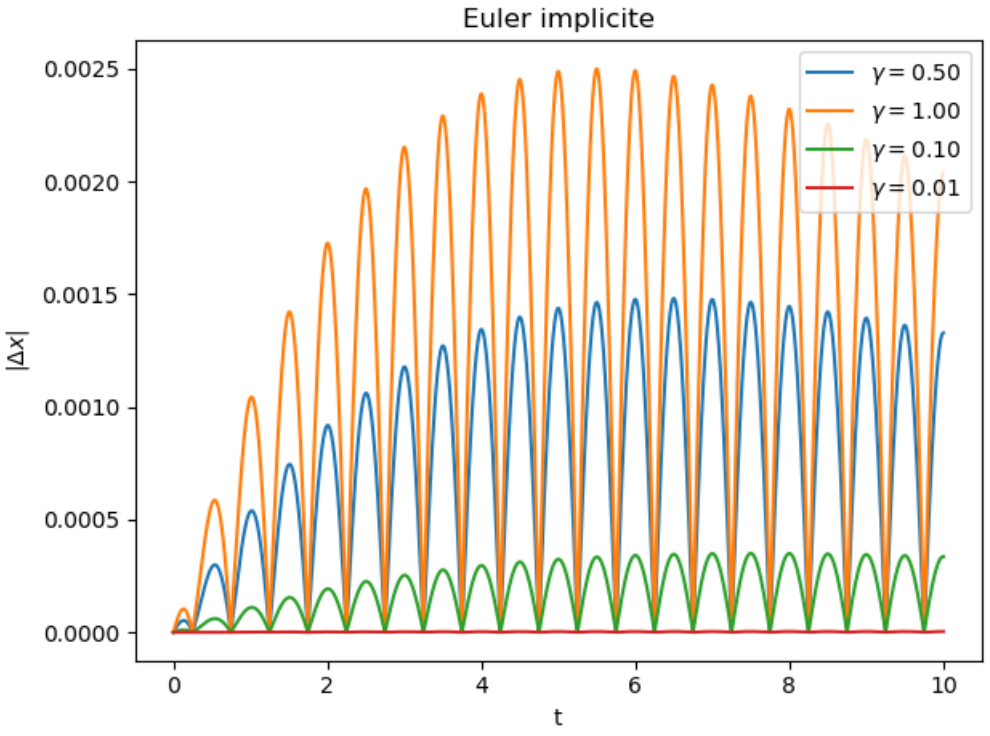
On a $b^2 - 4ac = 4w_0^2 \Delta^2 t \beta^2 (\epsilon^2 - 1) < 0$, Donc on peut calculer

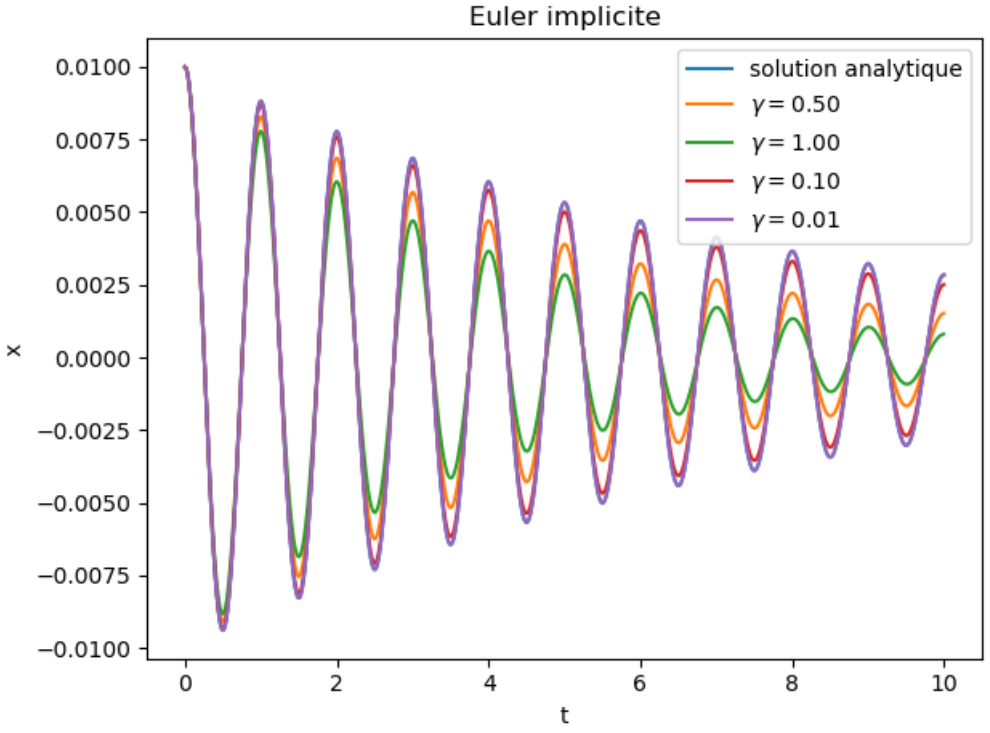
$$\lambda = (1 + \epsilon w_0 \Delta t \pm i w_0 \Delta t \sqrt{1 - \epsilon^2})\beta$$

Donc

$$|\lambda| = \beta = \frac{1}{1 + 2\epsilon w_0 \Delta t + w_0^2 \Delta^2 t}$$

Donc le pas de temps critique est 0, Il est donc toujour stable. On propose $\Delta t = \gamma \frac{2\epsilon}{w_0}$ et on a deux figures ci-dessous





On peut voir quand Δt devient de plus en plus petit, l'erreur devient de plus en plus petit. Et la courbe dvient de plus en plus proche de la solution analytique
Code python:

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


def solution_analytique(x0, x0_dot, epsilon, w0, T, delta_t):
    t_list = np.arange(0, T, delta_t)

    # exp(-epsilon*w0*t)
    outside = np.exp(-epsilon * w0 * t_list)
    omega = w0 * np.sqrt(1 - epsilon * epsilon)

    # inside parentheses
    term1 = x0 * np.cos(omega * t_list)
    term2 = ((epsilon * w0 * x0 + x0_dot) / omega) * np.sin(omega * t_list)

    x_list = outside * (term1 + term2)
    return t_list, x_list


def euler(x0, x0_dot, epsilon, w0, T, delta_t, mode="explicite"):
    if mode == "explicite":
        A = np.array([[1, delta_t],
                      [-w0 * w0 * delta_t, 1 - 2 * epsilon * w0 * delta_t]])
    elif mode == "implicite":
        m = (1 + 2 * epsilon * w0 * delta_t + w0 * w0 * delta_t * delta_t)
        A = np.array([[1 + 2 * epsilon * w0 * delta_t, delta_t],
                      [-w0 * w0 * delta_t, 1]])
        A = A/m
    else:
        raise NotImplementedError("Unknown mode: {}".format(mode))

    etat_init = np.array([x0, x0_dot])
    # transpose
    etat_init = etat_init.reshape(-1, 1)

    i = 0
    x_list = []
    t_list = []
    etat_i = etat_init

    while (i * delta_t < T):
        t_list.append(i * delta_t)
        x_list.append(etat_i[0][0])
        i = i + 1
        etat_i = A.dot(etat_i)

    return np.array(t_list), np.array(x_list)


if __name__ == '__main__':
    x0 = 0.01
    x0_dot = 0
    t0 = 1
    epsilon = 0.02
    w0 = 2 * np.pi / t0

    # delta_t = 0.01

    dt2 = 2 * epsilon / w0
    dt1 = 0.5 * dt2
    dt3 = 0.1 * dt2
    dt4 = 0.01 * dt3
    T = 10 * t0  # we consider only (0, T)

    t_list, x_list = solution_analytique(x0, x0_dot, epsilon, w0, T, dt1)
    t_list1, x_list1 = euler(x0, x0_dot, epsilon, w0, T, dt1, mode="implicite")
    erreur_list1 = np.abs(x_list1 - x_list)
```

```
    t_list, x_list = solution_analytique(x0, x0_dot, epsilon, w0, T, dt2)
    t_list2, x_list2 = euler(x0, x0_dot, epsilon, w0, T, dt2, mode="implicite")
    erreur_list2 = np.abs(x_list2 - x_list)

    t_list, x_list = solution_analytique(x0, x0_dot, epsilon, w0, T, dt3)
    t_list3, x_list3 = euler(x0, x0_dot, epsilon, w0, T, dt3, mode="implicite")
    erreur_list3 = np.abs(x_list3 - x_list)

    t_list, x_list = solution_analytique(x0, x0_dot, epsilon, w0, T, dt4)
    t_list4, x_list4 = euler(x0, x0_dot, epsilon, w0, T, dt4, mode="implicite")
    erreur_list4 = np.abs(x_list4 - x_list)

    # plt.plot(t_list1, erreur_list1, label="$\gamma = %.2f$" % (0.5))
    # plt.plot(t_list2, erreur_list2, label="$\gamma = %.2f$" % (1.0))
    # plt.plot(t_list3, erreur_list3, label="$\gamma = %.2f$" % (0.1))
    # plt.plot(t_list4, erreur_list4, label="$\gamma = %.2f$" % (0.01))

    plt.plot(t_list, x_list, label="solution analytique")
    plt.plot(t_list1, x_list1, label="$\gamma = %.2f$" % (0.5))
    plt.plot(t_list2, x_list2, label="$\gamma = %.2f$" % (1.0))
    plt.plot(t_list3, x_list3, label="$\gamma = %.2f$" % (0.1))
    plt.plot(t_list4, x_list4, label="$\gamma = %.2f$" % (0.01))

    plt.title("Euler implicite")
    plt.xlabel("t")
    plt.ylabel("x")
    # plt.ylabel("$|\Delta x|$")
    plt.legend(loc="upper right")
    plt.show()
```

solution_analytique.m

```
function [t_list, x_list] =  solution_analytique(x0, x0_dot, epsilon, w0, T, delta_t)

t_list = 0:delta_t:T;


% exp(-epsilon*w0*t)
outside = exp(-epsilon * w0 * t_list);
omega = w0 * sqrt(1 - epsilon * epsilon);

% inside parentheses
term1 = x0 * cos(omega * t_list);
term2 = ((epsilon * w0 * x0 + x0_dot) / omega) * sin(omega * t_list);

x_list = outside * (term1 + term2);

end
```

euler_implicite.m

```
function [t_list, x_list] = euler_implicite(x0, x0_dot, epsilon, w0, T, delta_t):

m = 1 + 2 * epsilon * w0 * delta_t + w0 * w0 * delta_t * delta_t;
A = [1 + 2 * epsilon * w0 * delta_t, delta_t;
              -w0 * w0 * delta_t, 1];
A = A./m;


etat_init = [x0, x0_dot];
% transpose
etat_init = etat_init';

i = 0;
x_list = [];
t_list = [];
etat_i = etat_init;

while (i * delta_t < T)
    t_list = [t_list, i*delta_t];
    x_list = [x_list, etat_i(0, 0)];
    i = i + 1;
    etat_i = A*etat_i;
end
end
```

main.m

```
x0 = 0.01;
x0_dot = 0;
t0 = 1;
epsilon = 0.02;
w0 = 2 * pi / t0;

dt2 = 2 * epsilon / w0;
dt1 = 0.5 * dt2;
dt3 = 0.1 * dt2;
dt4 = 0.01 * dt3;
T = 10 * t0;  % we consider only (0, T)

[t_list, x_list] = solution_analytique(x0, x0_dot, epsilon, w0, T, dt1);
[t_list1, x_list1] = euler_implicite(x0, x0_dot, epsilon, w0, T, dt1);
erreur_list1 = abs(x_list1 - x_list);

[t_list, x_list] = solution_analytique(x0, x0_dot, epsilon, w0, T, dt2);
[t_list2, x_list2] = euler_implicite(x0, x0_dot, epsilon, w0, T, dt2);
erreur_list2 = abs(x_list2 - x_list);

[t_list, x_list] = solution_analytique(x0, x0_dot, epsilon, w0, T, dt3);
[t_list3, x_list3] = euler_implicite(x0, x0_dot, epsilon, w0, T, dt3);
erreur_list3 = abs(x_list3 - x_list);

[t_list, x_list] = solution_analytique(x0, x0_dot, epsilon, w0, T, dt4);
[t_list4, x_list4] = euler_implicite(x0, x0_dot, epsilon, w0, T, dt4);
erreur_list4 = abs(x_list4 - x_list);

plot(t_list1, erreur_list1, 'DisplayName', sprintf('gamma=%0.2f', 0.5))
hold on
plot(t_list2, erreur_list2, 'DisplayName', sprintf('gamma=%0.2f', 1.0))
hold on
plot(t_list3, erreur_list3, 'DisplayName', sprintf('gamma=%0.2f', 0.1))
hold on
plot(t_list4, erreur_list4, 'DisplayName', spritnf('gamma=%0.2f', 0.01))

title('Euler explicite')
xlabel('t')
ylabel('delta x')
hold off
legend
```

Runge Kutta

1.3) Pour l'equation 1

$$\ddot{x} + 2\epsilon w_0 \dot{x} + w_0^2 x = 0 \tag{1}$$

On intorduit une variable $z = \dot{x}$, donc on a $\dot{z} = -2\epsilon w_0 z - w_0^2 x$. Donc on a deux équation différentiel d'ordre 1.

Puis on utilise la méthode Runge Kutta d'ordre 4.

$$\begin{cases} k_1 = z_0 = \dot{x}_0 \\ j_1 = -2\epsilon w_0 z_0 - w_0^2 x_0 \\ k_2 = z_0 + \frac{j_1}{2}\Delta t \\ j_2 = -2\epsilon w_0 (z_0 + \frac{j_1}{2}\Delta t) - w_0^2 (x_0 + \frac{k_1}{2}\Delta t) \\ k_3 = z_0 + \frac{j_2}{2}\Delta t \\ j_3 = -2\epsilon w_0 (z_0 + \frac{j_2}{2}\Delta t) - w_0^2 (x_0 + \frac{k_2}{2}\Delta t) \\ k_4 = z_0 + j_3 \Delta t \\ j_4 = -2\epsilon w_0 (z_0 + j_3 \Delta t) - w_0^2 (x_0 + k_3 \Delta t) \end{cases}$$

Donc on a :

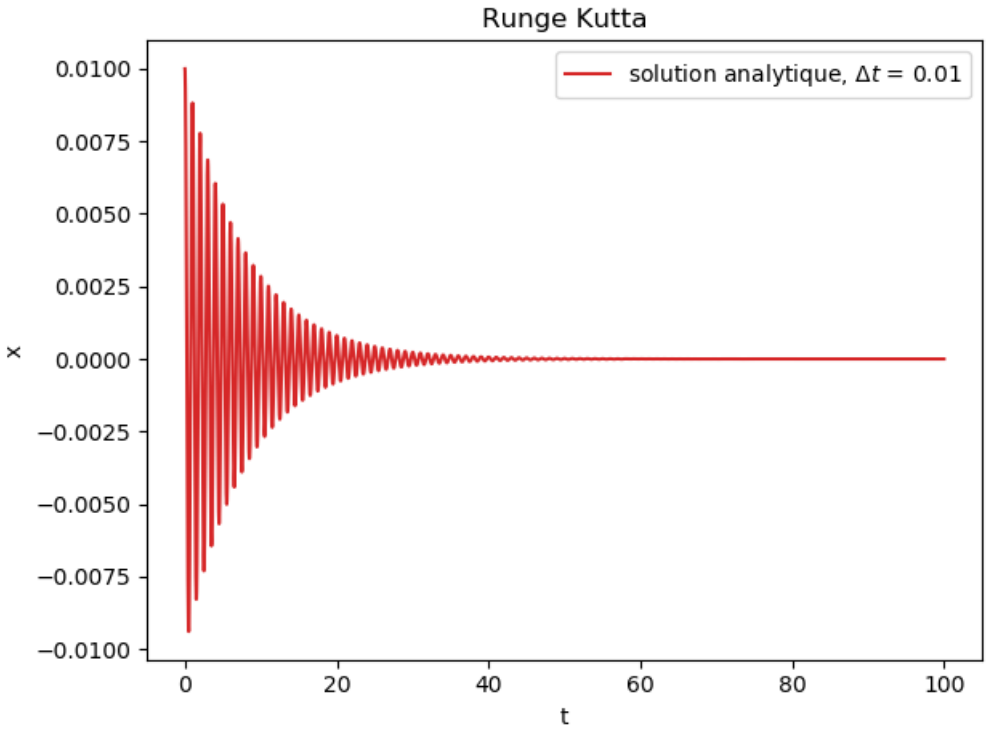
$$x_1 = x_0 + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6}\Delta t$$

et

$$z_1 = z_0 + \frac{j_1 + 2j_2 + 2j_3 + j_4}{6}\Delta t$$

Ici x_0, z_0 signifie état entrée et x_1, z_1 signifie état sortie.

Avant regarder la question 1.3.a, on doit savoir la solution analytique sur l'intervalle du temps $[0, 100T_0]$. On a la figure ci-dessous.



1.3.a) D'aprys les formules qu'on a obtenue on peut le programmer:
code python

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


def solution_analytique(x0, x0_dot, epsilon, w0, T, delta_t):
    t_list = np.arange(0, T, delta_t)

    # exp(-epsilon*w0*t)
    outside = np.exp(-epsilon * w0 * t_list)
    omega = w0 * np.sqrt(1 - epsilon * epsilon)

    # inside parentheses
    term1 = x0 * np.cos(omega * t_list)
    term2 = ((epsilon * w0 * x0 + x0_dot) / omega) * np.sin(omega * t_list)

    x_list = outside * (term1 + term2)
    return t_list, x_list

def one_step(x0, x0_dot, epsilon, w0, delta_t):
    z0 = x0_dot

    k1 = z0
    j1 = - 2 * epsilon * w0 * z0 - w0 * w0 * x0
    k2 = z0 + j1 / 2. * delta_t
    j2 = - 2 * epsilon * w0 * (z0 + j1 / 2. * delta_t) - w0 * w0 * (x0 + k1 / 2. * delta_t)
    k3 = z0 + j2 / 2. * delta_t
    j3 = - 2 * epsilon * w0 * (z0 + j2 / 2. * delta_t) - w0 * w0 * (x0 + k2 / 2. * delta_t)
    k4 = z0 + j3 * delta_t
    j4 = -2 * epsilon * w0 * (z0 + j3 * delta_t) - w0 * w0 * (x0 + k3 * delta_t)

    x1 = x0 + (k1 + 2 * k2 + 2 * k3 + k4) / 6. * delta_t
    z1 = z0 + (j1 + 2 * j2 + 2 * j3 + j4) / 6. * delta_t

    return x1, z1

def runge_kutta(x0, x0_dot, epsilon, w0, T, delta_t):
    t_list = []
    x_list = []

    i = 0
    xi = x0
    xi_dot = x0_dot
    while (i * delta_t < T):
        t_list.append(i * delta_t)
        x_list.append(xi)
        xi, xi_dot = one_step(xi, xi_dot, epsilon, w0, delta_t)
        i = i + 1

    return t_list, x_list


if __name__ == '__main__':
    x0 = 0.01
    x0_dot = 0
    t0 = 1
    epsilon = 0.02
    w0 = 2 * np.pi / t0

    T = 100 * t0  # we consider only (0, T)

    base_delta_t = 2 * np.sqrt(2) / w0
    h = np.array([0.04, 0.96, 1.04])  # choix de h
    dt = h * base_delta_t

    t_list, x_list = solution_analytique(x0, x0_dot, epsilon, w0, T, 0.01)

    plt.plot(t_list, x_list,
             color="C3",
```

```
            label="solution analytique, $\Delta t$ = {}".format(0.01))

    for i in range(h.shape[0]):
        t_list_i, x_list_i = runge_kutta(x0, x0_dot, epsilon, w0, T, dt[i])
        plt.plot(t_list_i, x_list_i, label="Runge Kutta, $h = %.2f$" % h[i])

    plt.title("Runge Kutta")
    plt.xlabel("t")
    plt.ylabel("x")
    plt.ylim(-0.015, 0.015)
    plt.legend(loc="upper right")
    plt.show()
```

code matlab

solution_analytique.m

```
function [t_list, x_list] =  solution_analytique(x0, x0_dot, epsilon, w0, T, delta_t)

t_list = 0:delta_t:T;


% exp(-epsilon*w0*t)
outside = exp(-epsilon * w0 * t_list);
omega = w0 * sqrt(1 - epsilon * epsilon);

% inside parentheses
term1 = x0 * cos(omega * t_list);
term2 = ((epsilon * w0 * x0 + x0_dot) / omega) * sin(omega * t_list);

x_list = outside * (term1 + term2);

end
```

one_step.m

```
function [x1, z1] =  one_step(x0, x0_dot, epsilon, w0, delta_t)

z0 = x0_dot;

k1 = z0;
j1 = - 2 * epsilon * w0 * z0 - w0 * w0 * x0;
k2 = z0 + j1 / 2.0 * delta_t;
j2 = - 2 * epsilon * w0 * (z0 + j1 / 2.0 * delta_t) - w0 * w0 * (x0 + k1 / 2.0 * delta_t);
k3 = z0 + j2 / 2.0 * delta_t;
j3 = - 2 * epsilon * w0 * (z0 + j2 / 2.0 * delta_t) - w0 * w0 * (x0 + k2 / 2.0 * delta_t);
k4 = z0 + j3 * delta_t;
j4 = -2 * epsilon * w0 * (z0 + j3 * delta_t) - w0 * w0 * (x0 + k3 * delta_t);

x1 = x0 + (k1 + 2 * k2 + 2 * k3 + k4) / 6.0 * delta_t;
z1 = z0 + (j1 + 2 * j2 + 2 * j3 + j4) / 6.0 * delta_t;

end
```

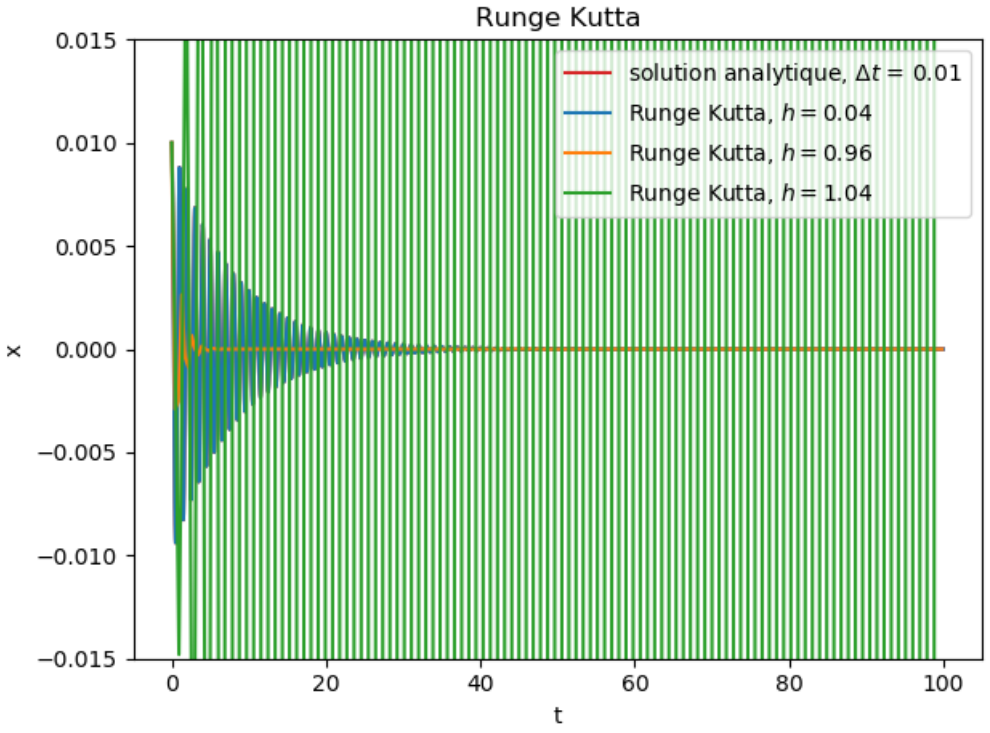
runge_kutta.m

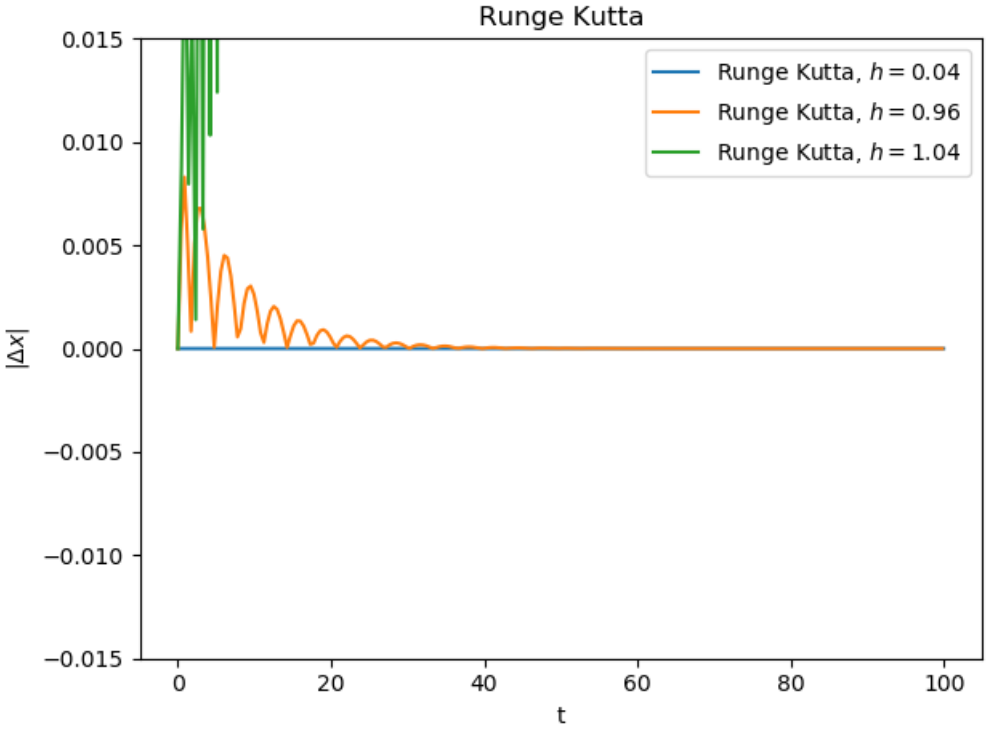
```
function [t_list, x_list] = runge_kutta(x0, x0_dot, epsilon, w0, T, delta_t)

t_list = ;
x_list = [];

i = 0;
xi = x0;
xi_dot = x0_dot;
while (i * delta_t < T)
    t_list = [t_list, i*delta_t];
    x_list = [x_list, xi];
    [xi, xi_dot] = one_step(xi, xi_dot, epsilon, w0, delta_t);
    i = i + 1;
end

end
```

main.m

```
x0 = 0.01;
x0_dot = 0;
t0 = 1;
epsilon = 0.02;
w0 = 2 * np.pi / t0;

T = 100 * t0;  % we consider only (0, T)

base_delta_t = 2 * sqrt(2) / w0;
h = [0.04, 0.96, 1.04];  % choix de h
dt = h * base_delta_t;

[t_list, x_list] = solution_analytique(x0, x0_dot, epsilon, w0, T, 0.01);

plot(t_list, x_list, 'DisplayName', 'solution analytique')
hold on

for i = 1:3
    [t_list_i, x_list_i] = runge_kutta(x0, x0_dot, epsilon, w0, T, dt(i));
    plot(t_list_i, x_list_i, 'DisplayName', sprintf('runge kutta, h=%0.2f', dt(i)));
    hold on
end

title('Runge Kutta')
xlabel('t')
ylabel('x')
hold off
legend
```

Et on a la figure:





On peut voir très clairement quand $h = 0.04$, la courbe est proche de la solution analytique. Mais quand $h = 0.96$ il y une atténuation forte comparé avec la solution analytique. Quand $h = 1.04$, Il existe une divergence forte en revenche, il n'est pas stable. Pour la précision, quand $h = 0.04$, l'erreur est la plus petite. Qunad $h = 0.96$ l'erreur devient de plus en plus petite car la courbe tends vers 0 quand x tends ver l'infinie. Car la méthode RK4 est une méthode d'ordre 4 au sens où l'erreur commise à chaque étape $(x_i, t_i) \to (x_{i+1}, t_{i+1})$ est de l'ordre $\Delta^5 t$, donc l'erreur total accumulé est de $\Delta^4 t$.

1.3.b) En effet, je le trouve par la programmation.

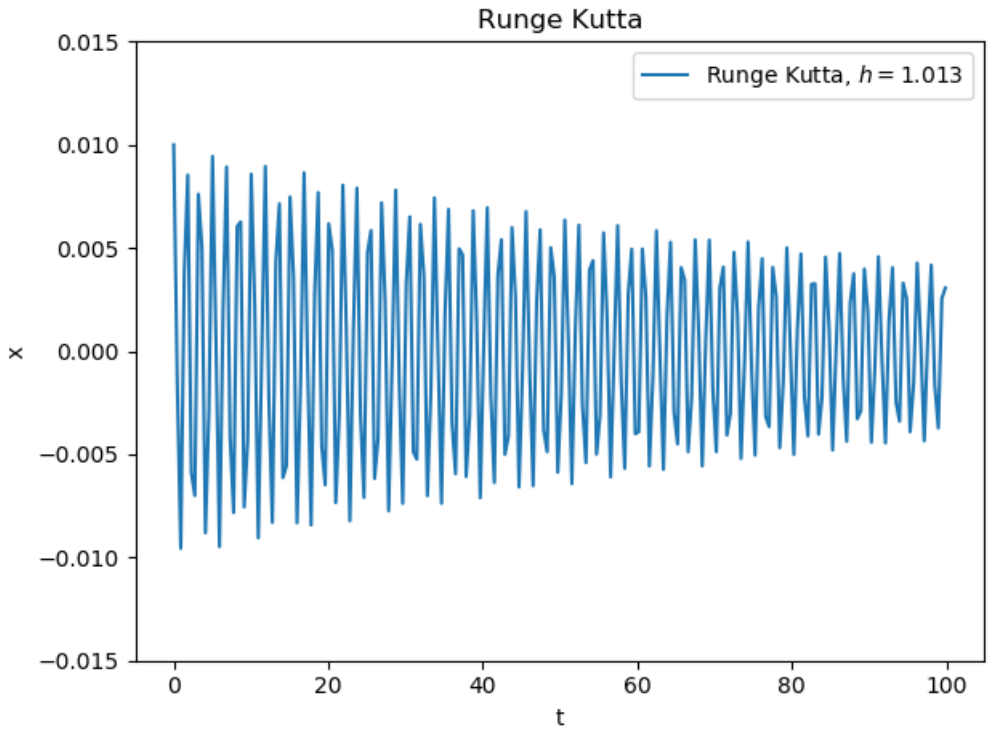
$$h_{min} = 1.013, h_{max} = 1.014$$

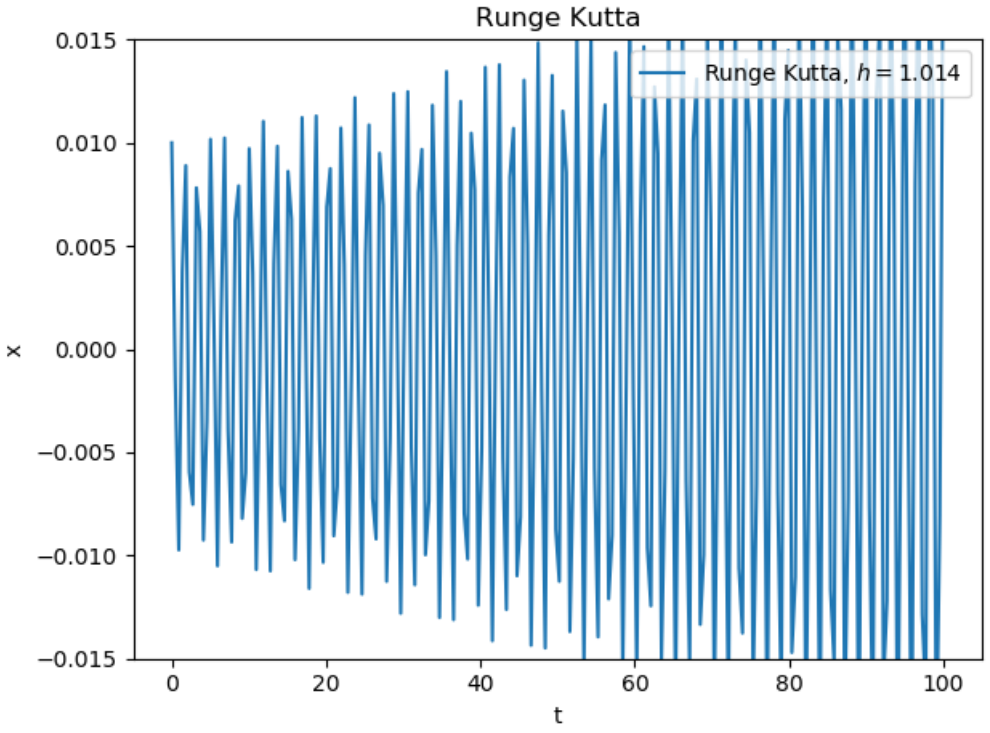
code python:

```python
base_delta_t = 2 * np.sqrt(2) / w0
for h in list(np.arange(0.96, 1.04, 0.001)):
    h_min = h
    h_max = h+0.001

    t_list_min, x_list_min = runge_kutta(x0, x0_dot, epsilon, w0, T, h_min*base_delta_t)
    t_list_max, x_list_max = runge_kutta(x0, x0_dot, epsilon, w0, T, h_max*base_delta_t)

    flag1 = np.max(x_list_min)<=0.01 # si h = h_min, convergence
    flag2 = np.max(x_list_max)>0.01 # si h = h_max, divergence
    flag = flag1 and flag2
    if flag:
        print("Found: hmin = {}, hmax={}".format(h_min, h_max))
```

code matlab

```matlab
base_delta_t = 2 * sqrt(2) / w0;

for h = 0.96:0.001:1.04
    h_min = h;
    h_max = h+0.001;

    [t_list_min, x_list_min] = runge_kutta(x0, x0_dot, epsilon, w0, T, h_min*base_delta_t);
    [t_list_max, x_list_max] = runge_kutta(x0, x0_dot, epsilon, w0, T, h_max*base_delta_t);

    flag1 = max(x_list_min)<=0.01 % si h = h_min, convergence;
    flag2 = max(x_list_max)>0.01 % si h = h_max, divergence;
    flag = flag1 & flag2;
    if (flag)
        disp(sprintf('Found: hmin = %0.3f, hmax=%0.3f', h_min, h_max));
    end

end
```




Runge Kutta
Runge Kutta, $h = 1.014$
0.015
0.010
0.005
0.000
−0.005
−0.010
−0.015
x
0
20
40
60
80
100
t