Simuler la réponse d'un système linéaire du deuxième ordre à une excitation de forme quelconque

Rappelons le programme officiel :

Équations différentielles d'ordre supérieur ou égal à 2	Transformer une équation différentielle d'ordre n en un système différentiel de n équations d'ordre 1. Utiliser la fonction **odeint** de la bibliothèque **scipy.integrate** (sa spécification étant fournie).

I. Méthode

Une possibilité pour résoudre une équation différeentielle est la fonction odeint du module scipy.integrate – attention, il s'agit d'un calcul approché qui ne fonctionne qu'avec des dérivées premières.

On écrit : solution = odeint (équation vérifiée par la dérivée, CI, t)

La solution contient alors un tableau de valeurs numériques que l'on peut pointer aux dates correspondantes pour tracer la solution.

Odeint ne permet pas de résoudre que des équations différentielles d'ordre 1, mais peut gérer un système de plusieurs équations différentielles d'ordre 1, ce qui va nous permettre de résoudre les équations d'ordre n .

Considérons par exemple une équation différentielle scalaire d'ordre 2, c'est-à-dire de la forme :
$\ddot{x} = f(x, \dot{x}, t)$
La méthode utilisée ici consiste à se ramener à un système différentiel en considérant le vecteur $X = (x, \dot{x})$.
Ce dernier est en effet solution du système différentiel : $\begin{cases} \dot{x} = y \\ \dot{y} = f(x, y, t) \end{cases}$ soit $\dot{X} = F(X, t)$

Par exemple, pour $\ddot{u} + \frac{\omega_0}{Q}\dot{u} + \omega_0^2 u = 0$, on écrira : $\dot{u} = u_p$ (cela définit donc la dérivée de u)
et $\dot{u_p} = -\frac{\omega_0}{Q}u_p - \omega_0^2 u$ (la dérivée seconde de u est la dérivée première de u_p).

Alors, on peut appliquer la méthode d'Euler, qui consiste à discrétiser l'intervalle de temps [t_0, tmax] et à calculer la suite de valeurs $(u_k, \dot{u}_k)$ à partir des valeurs initiales (u_0, $\dot{u}_0$) et des relations :

$$\begin{cases} u_{k+1} = u_k + pas.\dot{u}_k \\ \dot{u}_{k+1} = \dot{u}_k - pas.\frac{\omega_0}{Q}\dot{u}_k - pas.\omega_0^2 u_k \end{cases}$$

II. Exemple 1 : Charge ou décharge d'un condensateur

On a vu que lors de la charge d'un condensateur C par une tension E la tension u(t) aux bornes du condensateur vérifiait l'équation différentielle :

$$\dot{u} + \frac{u}{\tau} = \frac{E}{\tau}, \text{ soit encore : } \dot{u} = \frac{E-u}{\tau}$$

```
# Charge d'un condensateur dans RC sous la tension E

# On importe les bibliothèques.
from scipy.integrate import odeint
import numpy as np
import matplotlib.pyplot as plt

# On introduit les conditions initiales.
E = 10          # tension d'entrée en Volt
u0 = 0          # tension initiale en Volt
tau = 0.1       # temps caractéristique du circuit
t0 = 0          # date initiale
tmax = 0.5      # date finale, à adapter

# On définit la fonction f égale à la dérivée de u.
def f(u, t):
        return (E - u)/tau

# On fera varier le temps entre t0 et tmax.
t = np.linspace( t0, tmax, 100)   # avec 100 valeurs (à adapter)

# On calcule les valeurs de u aux dates t.
u = odeint ( f, u0, t)

# On trace la solution.
plt.plot ( t, u , 'g')          # courbe en vert (à adapter)
plt.xlabel ("Temps t en s")
plt.ylabel("Tension en Volt")
plt.title ("Tension aux bornes de C en fonction du temps")
plt.grid(True)
plt.show()
```

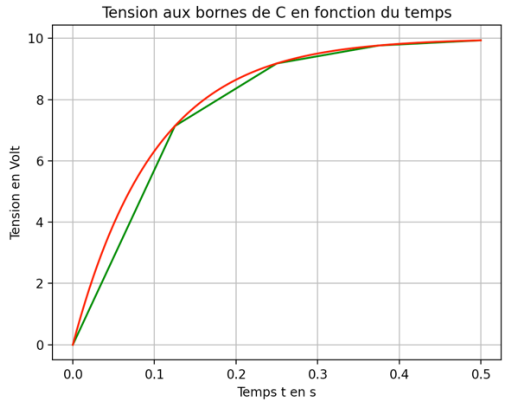
Sur le même graphe, on peut faire tracer l'évolution théorique de u(t), puisque le calcul a été fait en cours :

$$u(t) = E\left(1 - e^{-t/\tau}\right)$$

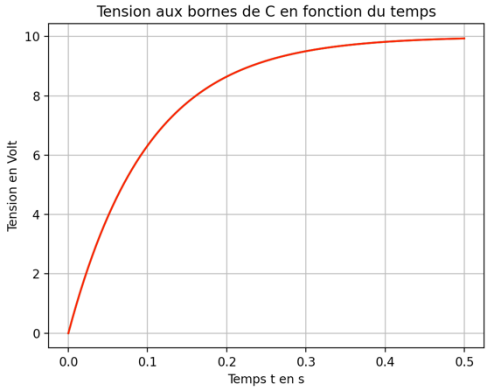
```
# Tracé de la courbe calculée
temps = []      # liste de valeurs des temps
Utheo = []      # liste des valeurs de u(t)

for t in np.linspace( t0, tmax, 101):
        u = E * (1 - np.exp ( t / (-tau) ) )
        temps.append (t)
        Utheo.append(u)

plt.plot (temps, Utheo, 'r')
plt.show()
```

Attention à discrétiser suffisamment le temps quand on applique la méthode d'Euler.

Exemple avec 5 dates :



Exemple avec 100 dates :



III. Exemple 2 : Etude du RLC

Exemple de la décharge d'un condensateur RLC :

La tension u aux bornes de C vérifie : $\ddot{u} + \frac{\omega_0}{Q}\dot{u} + \omega_0^2 u = 0$. On pose $\dot{u} = u_p$. Alors up est solution de :

$$\dot{u_p} = -\frac{\omega_0}{Q}u_p - \omega_0^2 u$$

On obtient un système de 2 équations différentielles d'ordre 1 :

$$\begin{cases} \dot{u} = u_p \\ \dot{u_p} = -\frac{\omega_0}{Q}u_p - \omega_0^2 u \end{cases}$$

```
# Décharge d'un condensateur dans RLC

# On importe les bibliothèques.
from scipy.integrate import odeint
import numpy as np
import matplotlib.pyplot as plt

# On introduit les conditions initiales
u0 = 10          # tension initiale en V
up0 = 0          # dérivée initiale en V/s
t0 = 0           # temps caractéristique en s
tmax = 40        # à adapter
w0 = 1           # pulsation propre en rad/s
Q = 5            # facteur de qualité

# On définit la fonction f, qui contient le système d'équations différentielles:
def f(s, t):
        [u, up] = s        # On crée un tableau avec u et sa dérivée up. s est le couple (u, up).
        return np.array ( [ up , - w0 * up / Q - w0**2 * u ])     # Cela correspond au système d'équations à résoudre.

# On fera varier le temps entre t0 et tmax
t =np.linspace ( t0, tmax, 500 )          # On pourra adapter le nombre de points.

# On calcule les valeurs de u et up aux dates t
s = odeint (f, [u0, up0], t)
```

```
# Tracé de la courbe
plt.plot (t, s[:, 0])          # on ne retient que la première colonne de s pour avoir u
plt.xlabel ("Temps t en s")
plt.ylabel ("Tension en Volt")
plt.title ("Tension aux bornes de C en fonction du temps")
plt.grid (True)
plt.show ()
```

Travail à faire : Tracer la courbe avec odeint pour la décharge du condensateur dans un RLC série avec Q = 6, Q = 0.5 et Q = 0.2. On superposera les courbes.