Méthodes numériques

Méthode d'Euler (équations différentielles du 1er ordre)

La méthode d'Euler est une méthode numérique qui permet de résoudre numériquement une équation différentielle en approximant les dérivées par des développements limités.

Ici on s'intéresse à une équation différentielle du premier ordre, par exemple :

$$\frac{\mathrm{d}u}{\mathrm{d}t} = \underbrace{-\frac{u}{\tau} + \frac{E}{\tau}}_{F}, \qquad \text{CI : } u(t=0) = u_0$$

dont l'inconnue est $u(t)$.
Ici E est une tension imposée par le générateur pour $t > 0$ (et nulle pour $t < 0$).

Dans la suite on note le membre de droite F pour alléger les notations.

Étape 1 : écrire l'équation sous forme "discrète". Ceci consiste à utiliser une approximation pour la dérivée. La méthode d'Euler *explicite* utilise l'approximation suivante :

$$\frac{\mathrm{d}u}{\mathrm{d}t} \simeq \frac{u(t+\mathrm{d}t) - u(t)}{\mathrm{d}t} \quad \text{avec} \quad \mathrm{d}t \text{ une durée très courte.}$$

$\rightsquigarrow_1$ On s'en sert pour isoler $u(t+\mathrm{d}t)$:

$$\frac{\mathrm{d}u}{\mathrm{d}t} = F \quad \Rightarrow \quad \frac{u(t+\mathrm{d}t) - u(t)}{\mathrm{d}t} = F$$

et on cherche à isoler $u(t+\mathrm{d}t)$:

$$u(t+\mathrm{d}t) - u(t) = F \times \mathrm{d}t \quad \Rightarrow \quad \boxed{u(t+\mathrm{d}t) = u(t) + F \times \mathrm{d}t.}$$

Remarque : une autre façon de voir ceci est d'utiliser un développement limité,

$$u(t+\mathrm{d}t) = u(t) + \frac{\mathrm{d}u}{\mathrm{d}t}(t) \times \mathrm{d}t \quad \text{avec} \quad \frac{\mathrm{d}u}{\mathrm{d}t}(t) = F \text{ d'après l'équation à résoudre.}$$

Étape 2 : idée de l'algorithme.
L'équation encadrée permet de calculer $u(t+\mathrm{d}t)$ si on connaît $u(t)$.

$\rightsquigarrow_2$ Par exemple, que donne cette relation pour $t = 0$? pour $t = \mathrm{d}t$?

$u(\mathrm{d}t) = u_0 + F \times \mathrm{d}t$ et $u(2\mathrm{d}t) = u(\mathrm{d}t) + F \times \mathrm{d}t$.

On va donc procéder successivement :

- On part de u_0 qui est $u(t=0)$, on le met dans une liste `liste_u`.
- On en déduit $u(\mathrm{d}t) = u_0 + F \times \mathrm{d}t$ et on le met dans la liste.
- On en déduit $u(2\mathrm{d}t) = u(\mathrm{d}t) + F \times \mathrm{d}t$ et on le met dans la liste.
- Etc... on en déduit u à tous les instants $i \times \mathrm{d}t$, et on s'arrête quand cela suffit.

Détaillons :

- Ceci permet donc de connaître $u(t)$ à certains instants t :

  $$t_0 = 0,\ t_1 = \mathrm{d}t,\ t_2 = 2\mathrm{d}t,\ ...,\ t_N = N\mathrm{d}t.$$

  $\rightsquigarrow_3$ Si on se donne une durée notée `fin` et un pas de temps noté `dt`, alors le nombre total de points sera :

  $$N = \frac{\text{fin}}{\mathrm{d}t}.$$

  Par exemple si fin $= 10\,\mathrm{s}$ et $\mathrm{d}t = 0{,}1\,\mathrm{s}$, alors $N = \dfrac{\text{fin}}{\mathrm{d}t} =$ ... points.

- On initialise une liste pour le temps et une pour la tension u, avec les valeurs initiales : `liste_u = [u0]` et `liste_t = [0]`.

  Après exécution de tout l'algorithme,

  - `liste_t[i]` vaudra $t_i = i \times \mathrm{d}t$
  - `liste_u[i]` contiendra la valeur $u(t_i)$.

- L'algorithme remplit les listes avec une boucle `for`.

  $\rightsquigarrow_4$ Si on suppose connues les valeurs de t et de u à l'itération précédente, alors celle de l'itération en cours, qu'on va encore noter t et u, sont déduites de la relation

  $$u(t + \mathrm{d}t) = u(t) + (-\frac{u}{\tau} + \frac{E}{\tau}) \times \mathrm{d}t \ :$$

```
u = u + dt*(-u/tau + E/tau)
t = t + dt
```

  et on les ajoute à `liste_t` et `liste_u` avec la commande `append` :

```
liste_u.append(u)
liste_t.append(t)
```

  Remarque : on a donc la relation de récurrence suivante :

  $$u(t_{i+1}) = u(t_i) + dt * (-u(t_i)/tau + E/tau).$$

Le code :

```
tau = 1e3 * 1e-9 # calcul de tau = R*C
u0 = 0           # valeur initiale
E = 10           # valeur de la tension imposée pour t>0

fin = 5*tau      # durée de la simulation
dt = tau/10      # pas de temps, assez court
N = int(fin/dt)  # calcul du nombre total d'itérations nécessaires

#------- Méthode d'Euler -------#
# Initialisation des listes :
liste_t = [0]
liste_u = [u0]
t = 0
u = u0

for i in range(N):
    # on dispose des variables de l'itération précédente : t et u
    # et on calcule leurs nouvelles valeurs :
    u = u + dt*(-u/tau + E/tau)
    t = t + dt

    # On ajoute ces nouvelles valeurs à chaque liste :
    liste_t.append(t)
    liste_u.append(u)
```

Après exécution de ce code, `liste_t` et `liste_u` contiennent les valeurs de t et les valeurs de $u(t)$ à chacun des instants.

On peut ensuite tracer u en fonction du temps t (`plt.plot(liste_t,liste_u)`, cf partie du poly sur les tracés graphiques).

À vous de jouer

Ouvrir le script `info_Euler_condensateur.py`. **On ne modifiera pas ce fichier,** à la place, ouvrir l'interface Python, créer un script vierge, puis copier-coller le contenu du fichier `info_Euler_condensateur.py`.

1 - Compléter le script aux endroits indiqués. Exécuter le script : normalement ceci affiche la solution $u(t)$.

2 - Modifier le script pour que la simulation dure en tout 10τ et pour que la condition initiale soit $u_0 = 2\,\text{V}$.

3 - À l'aide de la fiche méthode, faire en sorte que le graphique ait un titre adéquat, et que la tension ne soit pas représentée par une ligne continue mais par des points (symbole `'o'`).

On souhaite ensuite tester la précision de la méthode d'Euler. Le cas étudié ici est simple et on connait la solution théorique : $u_{\text{théo}}(t) = (u_0 - E)\text{e}^{-t/\tau} + E$. Nous allons comparer ceci à la solution numérique.

4 - Le code nécessaire est écrit à la fin du script. Le décommenter et le couper-coller au-dessus du bloc "tracé graphique".

Comprendre ce que fait ce morceau de code.

Puis ajouter une ligne `plt.plot(...)` qui permet de tracer la solution théorique en fonction du temps sur le même graphique que précédemment.

5 - La solution théorique et la solution numérique sont-elles en bon accord ?

Changer le pas de temps `dt` : prendre $\mathrm{d}t = \tau/2$, $\mathrm{d}t = \tau$, etc. Que constatez-vous ? À partir de quelle valeur le pas de temps est-il assez petit pour avoir un bon accord ?

On souhaite maintenant étudier une autre équation différentielle, pour laquelle il est moins simple de connaître une solution analytique :

$$\frac{\mathrm{d}u}{\mathrm{d}t} = a \times (E - u)^2, \qquad \text{CI} : u(t=0) = u_0,$$

avec $a = 10^6\,\mathrm{s}^{-1} \cdot \mathrm{V}^{-1}$ un paramètre.

6 - Comment faut-il modifier votre algorithme pour que ce soit cette équation là qui est résolue ? Le faire et visualiser la solution.

Enfin, on s'intéresse à nouveau à la première équation, mais cette fois avec un générateur de tension variable :

$$\frac{\mathrm{d}u}{\mathrm{d}t} = -\frac{u}{\tau} + \frac{E(t)}{\tau}, \qquad \text{CI} : u(t=0) = u_0$$

avec $E(t) = E_0 \sin(\omega t)$ ($E_0 = 10\,\mathrm{V}$ et $\omega = 5 \times 10^6\,\mathrm{rad/s}$).

7 - Comment faut-il modifier votre algorithme pour que ce soit cette équation là qui est résolue ? (il faudra actualiser la valeur de E à l'intérieur de la boucle for)

Le faire et visualiser la solution.

Remarque, à lire plus tard : version avec des tableaux Numpy

Il est possible de faire la même chose non pas en utilisant des listes, mais des tableaux `numpy`. Le code est alors légèrement différent.

- On se donne une durée notée `fin` et un nombre de points `N`, et on crée directement un tableau de `N` points équirépartis entre 0 et `fin` avec la commande : `t = np.linspace(0,fin,N)`.

  Par exemple avec `t = np.linspace(0,10,100)`, on a `t[0]`, ..., `t[99]` sont 100 instants équirépartis entre 0 et 100.

  Le pas de temps est alors donné par `dt = t[1]-t[0]`.

- Puis on initialise un tableau qui contiendra les valeurs de u à chaque instant : `u = np.zeros(N)`.

  Ainsi `u[i]` contient la valeur $u(t_i)$ avec $t_i = i \times \mathrm{d}t$.

  La condition initiale est `u[0]=u0`.

- Enfin, il faut traduire la relation $u(t+\mathrm{d}t) = u(t) + (-\frac{u}{\tau} + \frac{E}{\tau}) \times \mathrm{d}t$ par une relation de récurrence entre les `u[i]` :

$$\text{si } t = t_i \text{ on a } u(\underbrace{t_i + \mathrm{d}t}_{t_{i+1}}) = u(t_i) + (-\frac{u(t_i)}{\tau} + \frac{E}{\tau}) \times \mathrm{d}t$$

  D'où en Python :

```
u[i+1] = u[i] + dt*(-u[i]/tau + E/tau)
```

Le code :

```
tau = 1e3 * 1e-9 # calcul de tau = R*C
u0 = 0           # valeur initiale
E = 10           # valeur de la tension imposée pour t>0

fin = 10*tau     # durée de la simulation
N = 1000         # nombre de points


# Initialisation des tableaux :
t  = np.linspace(0,fin,N)  # t est un tableau allant de 0 à fin avec N points
dt = t[1]-t[0]
u  = np.zeros(N)
u[0] = u0

for i in range(N-1): # on prend garde à s'arrêter à i=N-2
    u[i+1] = u[i] + dt*(-u[i]/tau + E/tau)  # connaissant u[i], on calcule u[i+1]
```