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 : utiliser une approximation pour la dérivée. Il y a plusieurs possibilités. 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 $u(t+\mathrm{d}t)$:

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

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

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

$$u(t+\mathrm{d}t) = u(t) + \frac{\mathrm{d}u}{\mathrm{d}t}(t) \times \mathrm{d}t$$

et on remplace $\frac{\mathrm{d}u}{\mathrm{d}t}(t)$ par F (car l'équation à résoudre est $\frac{\mathrm{d}u}{\mathrm{d}t} = F$). On obtient pareil.

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 $t = \mathrm{d}t$?

Pour $t = 0$ on a : $u(\mathrm{d}t) = u_0 + F \times \mathrm{d}t$. Pour $t = \mathrm{d}t$ on obtient : $u(2\mathrm{d}t) = u(\mathrm{d}t) + F \times \mathrm{d}t$.

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.

Voici le code complet, que l'on va expliquer :

```
1  R = 1e3
2  C = 1e-9
3  tau = R*C          # calcul de tau = R*C
4  u0 = 0             # valeur initiale
5  E = 10             # valeur de la tension imposée pour t>0
6
7  tmax = 5*tau       # on se donne la durée de la simulation
8  dt = tau/10        # ainsi que le pas de temps, assez court
9  N = int(tmax/dt)   # on en déduit par calcul le nombre total d'itérations nécessaires
10
11 #------- Méthode d'Euler -------#
12 # Initialisation des listes :
13 liste_t = [0]
14 liste_u = [u0]
15 t = 0
16 u = u0
17
18 for i in range(N):
19     # on dispose des variables de l'itération précédente : t et u
20     # et on calcule leurs nouvelles valeurs :
21     u = u + dt*(-u/tau + E/tau)
22     t = t + dt
23
24     # On ajoute ces nouvelles valeurs à chaque liste :
25     liste_t.append(t)
26     liste_u.append(u)
```

▶ **Lignes 1 à 5 :** définition des constantes.

▶ **Lignes 7 à 9 :**

⋆ On se donne une durée totale notée `tmax`, qu'on prend ici égale à $5 \times \tau$.

⋆ On se donne le pas de temps `dt`, qu'on prend ici égal à $\tau/10$.

→ On en déduit le nombre total N d'instants.

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

$\rightsquigarrow_3$ De manière générale, la formule est :

$$N = \frac{\mathrm{tmax}}{\mathrm{d}t}.$$

▶ **Lignes 13 à 16 :**

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

On initialise aussi les premières valeurs de u et de t : `u=u0` et `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 $u(t_i)$.

▶ **Lignes 18 et suivantes :** l'algorithme remplit les listes avec une boucle `for`.

⋆ Ligne 21 : on suppose connue l'ancienne valeur de u (elle vient de l'itération précédente). Alors la nouvelle valeur, qu'on note encore u, est obtenue à partir de la relation :

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

Ceci donne la ligne 21 : `u = u + dt*(-u/tau + E/tau)`.

⋆ Ligne 22 : même chose avec le temps. La nouvelle valeur de t est égale à l'ancienne, plus dt : `t = t + dt`.

⋆ Lignes 25 et 26 : on ajoute les nouvelles valeurs à `liste_t` et `liste_u` avec la commande `append`.

Remarque : si on note $u(t_i) = u_i$, on a en fait une relation de récurrence :

$$u_{i+1} = u_i + dt \times (-u_i/tau + E/tau),$$

qui permet de connaître $u(t)$ à certains instants t :

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

À vous de jouer

Aller sur l'ENT du lycée, puis sur Capytale, et démarrer l'activité `c84d-901065`.

Remarque : variantes possibles

En SII, vous allez également voir la méthode d'Euler. Ce sera la même chose qu'ici, à quelques détails près (mais qui ne changent pas la méthode) :

- Le pas de temps `dt` est noté `h`.
- La démonstration de la méthode passe non pas par l'approximation $\frac{\mathrm{d}u}{\mathrm{d}t} \simeq \frac{u(t+\mathrm{d}t)-u(t)}{\mathrm{d}t}$, mais par une intégration de l'équation et par l'utilisation d'une approximation pour calculer une intégrale numériquement. (Mais on arrive à la même chose.)
- Vous pouvez utiliser une boucle `while` à la place d'une boucle `for` et construire les listes un peu différemment.
- Il existe différentes méthodes d'Euler : ici nous avons vu la méthode explicite, en SII vous pourrez aussi voir la méthode implicite.

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 `tmax` et un nombre de points `N`, et on crée directement un tableau de `N` points équirépartis entre 0 et `tmax` avec la commande : `t = np.linspace(0,tmax,N)`.

  Par exemple avec `t = np.linspace(0,10,100)`, `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, après exécution du code, `u[i]` contiendra 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

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


# Initialisation des tableaux :
t  = np.linspace(0,tmax,N)  # t est un tableau allant de 0 à tmax 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]
```