BLAISE PASCAL
PT 2022-2023

Fiche outil

Méthode d'Euler

Plan du cours

Au programme

Extrait du programme officiel de PTSI : annexe 3 « Outils numériques », bloc 4 « Équations différentielles ».

Notions et contenus	Capacités exigibles
Équations différentielles d'ordre 1.	Mettre en œuvre la méthode d'Euler explicite afin de résoudre une équation différentielle d'ordre 1. Utiliser la fonction **odeint** de la bibliothèque **scipy.integrate**, sa spécification étant fournie.
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.

En **gras**, les points devant faire l'objet d'une approche expérimentale.

I - Schéma d'Euler explicite

Raisonnons sur un exemple simple : le tension aux bornes d'un condensateur d'un bête circuit RC régi par l'équation différentielle

$$\frac{\mathrm{d}u}{\mathrm{d}t}+\frac{1}{\tau}u=\frac{1}{\tau}e \qquad \text{avec} \qquad \begin{cases}\tau = RC \\ e(t) \text{ un forçage quelconque mais connu}\end{cases}$$

Comme il n'est pas possible de résoudre une équation différentielle « à tout instant » avec un ordinateur, il est nécessaire de passer par une résolution à N instants discrets $t_0, t_1, ..., t_n, ..., t_{N-1}$ en cherchant à ce que la valeur obtenue numériquement u_n soit aussi proche que possible de ce que donnerait la solution analytique $u(t_n)$. Pour toute la suite, le pas de temps Δt sera pris constant : $t_n = n\,\Delta t$.

En pratique, il faut transformer l'équation différentielle en une relation de récurrence permettant de calculer u_{n+1} à partir uniquement de ce que l'on connaît, c'est-à-dire la tension aux instants passés $u_n, u_{n-1}, ...$, la tension de forçage à tout instant et les paramètres caractéristiques du système. La « recette » pour passer de l'équation différentielle à la relation de récurrence est appelée **schéma numérique**.

Schéma d'Euler explicite :

La dérivée est approximée par un taux de variation entre les instants t_n et t_{n+1}, et toutes les autres grandeurs sont prises à l'instant t_n.

Ainsi, l'équation différentielle devient

$$\frac{u_{n+1}-u_n}{\Delta t}+\frac{1}{\tau}u_n=\frac{1}{\tau}e(t_n) \qquad \text{soit} \qquad u_{n+1}=u_n+\frac{\Delta t}{\tau}\big(e(t_n)-u_n\big)\,.$$

Bien sûr, il existe des schémas numériques plus astucieux, plus précis et/ou plus rapides, d'où l'intérêt d'utiliser des fonctions issues de bibliothèques. Notons d'ailleurs qu'il existe des domaines de recherche comme la climatologie, la turbulence, l'astrophysique, la physique des plasmas, la météorologie, etc., dans lesquels une grosse partie, pour ne pas dire la majorité, du travail théorique consiste à construire des schémas numériques performants pour résoudre des équations « assez simples » à poser.

II - Exemples d'implémentations en Python

Rappelons que pour un même algorithme, ici une même équation différentielle et un même schéma numérique, il existe souvent de nombreuses implémentations différentes. Donnons ici trois exemples pour une entrée sinusoïdale, commençant tous les trois par le même préambule.

```
import numpy as np

R = 1e3          # en ohms
C = 1e-6         # en farad
tau = R*C        # en secondes

E0 = 2           # amplitude du forcage, en volts
f = 1e3          # fréquence, en hertz

dt = 2e-5        # pas de temps, en s
N = 500          # nbre de pas de temps
```

- **Premier exemple**

```
t = [n*dt for n in range(N)]      # tps, en secondes
e = [E0 * np.cos(2*np.pi*f*tn) for tn in t]
                                  # tension d'entrée, en volts
u = [None for n in range(N)]      # tension du condensateur, en V
                                  # tout est initialisé à None,
                                  # c-a-d "rien du tout"
u[0] = 0        # condition initiale u(0) = 0 V

for n in range(N-1):
    u[n+1] = u[n] + dt/tau * ( e[n] - u[n] )
```

- **Deuxième exemple**

```
t = np.linspace(0,(N-1)*dt,N)     # N points entre 0 et (N-1)*dt
e = E0 * np.cos(2*np.pi*f*t)      # numpy calcule terme à terme !
u = np.empty_like(e)              # initialisée à un tableau vide
                                  # de même forme que e
u[0] = 0

for n in range(N-1):
    u[n+1] = u[n] + dt/tau * ( e[n] - u[n] )
```

- **Troisième exemple**

```
tn = 0       # initialisation des valeurs courantes
un = 0

t = [tn]     # listes ne contenant que la valeur initiale
u = [un]

while tn < (N-1)*dt:
    tn += dt           # actualisation des valeurs courantes
    un += dt/tau * ( E0 * np.cos(2*np.pi*f*tn) - un )
    t.append(tn)       # nouveaux éléments ajoutés à chq pas de tps
    u.append(un)
```

Cette dernière implémentation est clairement moins directement compréhensible que les deux précédentes ... mais elle renvoie le même résultat dans la liste `u` ! Lorsque le nombre de pas de temps est connu, utiliser une boucle **while** au lieu d'une boucle **for** est une mauvaise idée ... mais cela devient nécessaire si on ne connaît pas ce nombre à l'avance : imaginez que cherchiez à résoudre le PFD jusqu'à ce qu'un objet touche le sol, les itérations seraient à poursuivre « tant que $z(t) > 0$ ».

III - Pertinence des résultats

Un schéma numérique ne propose qu'une résolution numérique *approchée* de l'équation différentielle. Il importe donc de s'interroger sur la pertinence de ce résultat, qui peut très nettement s'écarter du « vrai » résultat auquel conduirait une résolution analytique. Au niveau PT, on se limite à une discussion qualitative, sans entrer dans une estimation précise des erreurs numériques, faisable mais très éloignée des objectifs du programme. Cette discussion qualitative est assez simple dans le cas de la méthode d'Euler, puisque le pas de temps Δt est le seul paramètre à choisir.

Pour que la méthode d'Euler puisse conduire à un résultat précis,
le pas de temps Δt doit être très petit devant tous les temps caractéristiques du système étudié.

Attention ! Ces temps caractéristiques peuvent intervenir dans l'équation différentielle, mais aussi dans le forçage ... et ce critère est une condition nécessaire mais pas toujours suffisante.

Exemple 1 : circuit RC soumis à un échelon de tension, le seul temps caractéristique est celui du système $\tau = RC$.

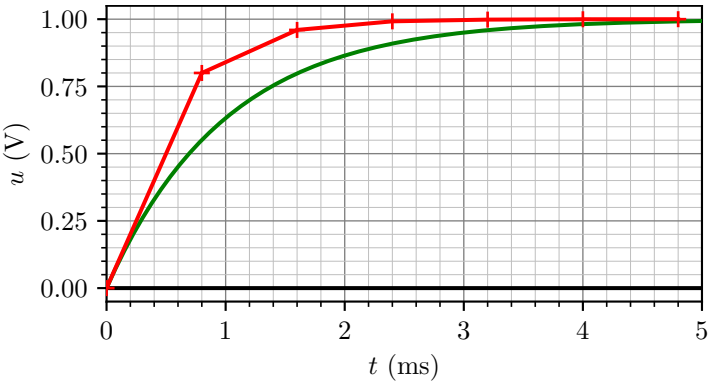

Figure 1 – Réponse du circuit RC à un échelon de tension. Courbe en trait plein vert : $\Delta t = \tau/1000$. Points rouges : $\Delta t = 0{,}8\,\tau$.

Exemple 2 : circuit RC soumis à un forçage sinusoïdal, qui fait intervenir un second temps caractéristique, la période T du forçage. Les simulations sont réalisées pour $\tau = 100\,T \gg T$. Pour les deux courbes représentées, le pas de temps est bien tel que $\Delta t \ll \tau$ ... mais on constate que ce n'est pas une condition suffisante, puisque pour $\Delta t \sim T$ la courbe obtenue est très éloignée de la courbe exacte !

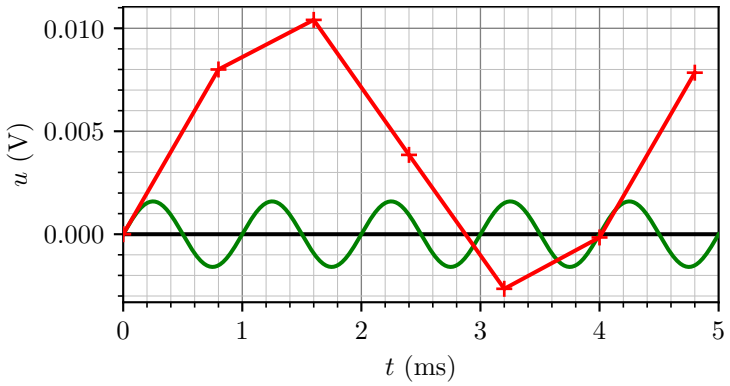

Figure 2 – Réponse du circuit RC à un forçage sinusoïdal. Courbe en trait plein vert : $\Delta t = T/1000$. Points rouges : $\Delta t = 0{,}8\,T$.

> ***Remarque :*** *Il existe même des situations dans lesquelles la méthode d'Euler devient instable, c'est-à-dire qu'elle renvoie des « solutions » divergentes, alors même que la solution analytique est parfaitement convergente ... et ce quelle que soit la valeur du pas de temps choisi, un pas de temps plus faible ne faisant que retarder l'échéance. Cet effet est illustré dans l'application 3.*

IV - Utilisation d'une bibliothèque

La méthode d'Euler est assez simple à comprendre, mais elle donne des résultats assez vite limités ... et la coder à chaque fois serait perdre beaucoup de temps dans un contexte autre que scolaire ! Ainsi, il est toujours préférable d'utiliser des fonctions déjà implémentées dans des bibliothèques.

Conformément au programme, on utilisera la fonction `odeint` de la bibliothèque `scipy.integrate`. Cette fonction demande d'écrire l'équation différentielle sous la forme

$$\frac{\mathrm{d}u}{\mathrm{d}t} = f(u,t)\,,$$

la fonction d'évolution f devant être codée séparément. La fonction s'utilise ensuite avec la syntaxe suivante :

```
from scipy.integrate import odeint # importe uniquement la fonction
u = odeint(f, u0, time)
```

avec `u0` la condition initiale et `time` un tableau `numpy` des instants auxquels on cherche la fonction `u`.

Exemple : reprenons l'exemple du circuit RC discuté précédemment. L'équation différentielle s'écrit sous la forme

$$\frac{\mathrm{d}u}{\mathrm{d}t} = \frac{1}{\tau}\,(e-u)$$

ce qui permet d'identifier la fonction d'évolution dans laquelle le temps intervient de manière implicite

$$f(u,t) = \frac{1}{\tau}\,(e(t)-u) = \frac{E_0}{\tau}\cos(2\pi\,f\,t) - \frac{u}{\tau}$$

en reprenant le cas particulier du forçage sinusoïdal pour clarifier. Une implémentation possible est la suivante, en réutilisant le même préambule que dans la partie II pour définir les grandeurs. La fonction d'évolution est dans cet exemple nommée `fonc` à cause du conflit de notation avec la fréquence.

```
def fonc(u,t):
    return E0/tau * np.cos(2 * np.pi * f * t) - u/tau

time = np.linspace(0,(N-1)*dt,N)     # N points entre 0 et (N-1)*dt
u = odeint(fonc,0,time)
```

V - Au delà des équations du premier ordre

V.A - Systèmes différentiels

La résolution de système de plusieurs équations différentielles du premier ordre avec le schéma d'Euler ne pose aucune difficulté particulière.

Exemple : considérons la réaction chimique $2\,\mathrm{NO} + \mathrm{O_2} \longrightarrow 2\,\mathrm{NO_2}$, dont la loi de vitesse s'écrit

$$v = k\,[\mathrm{NO}]^2\,[\mathrm{O_2}]\,.$$

On en déduit directement le système d'équations différentielles vérifiées par les concentrations des réactifs et du produit,

$$\begin{cases} -\dfrac{1}{2}\dfrac{\mathrm{d}[\mathrm{NO}]}{\mathrm{d}t} = k\,[\mathrm{NO}]^2\,[\mathrm{O_2}] \\ -\dfrac{\mathrm{d}[\mathrm{O_2}]}{\mathrm{d}t} = k\,[\mathrm{NO}]^2\,[\mathrm{O_2}] \\ \dfrac{1}{2}\dfrac{\mathrm{d}[\mathrm{NO_2}]}{\mathrm{d}t} = k\,[\mathrm{NO}]^2\,[\mathrm{O_2}] \end{cases}$$

que l'on transforme en relations de récurrence avec le schéma d'Euler,

$$\begin{cases} [\mathrm{NO}]_{n+1} = [\mathrm{NO}]_n - 2k\,\Delta t\,[\mathrm{NO}]_n^2\,[\mathrm{O_2}]_n \\ [\mathrm{O_2}]_{n+1} = [\mathrm{O_2}]_n - k\,\Delta t\,[\mathrm{NO}]_n^2\,[\mathrm{O_2}]_n \\ [\mathrm{NO_2}]_{n+1} = [\mathrm{NO_2}]_n + 2k\,\Delta t\,[\mathrm{NO}]_n^2\,[\mathrm{O_2}]_n \end{cases}$$

Il suffit ensuite simplement d'inclure ces trois relations de récurrence dans chaque itération de la boucle **for**.

V.B - Équations différentielles d'ordre 2

Le schéma d'Euler ne permet pas de résoudre directement des équations différentielles d'ordre 2. Pour contourner la difficulté, il est nécessaire de ruser en introduisant une deuxième fonction inconnue égale à la dérivée première, ce qui permet de se ramener à un système différentiel formé de deux équations du premier ordre.

Exemple : considérons un oscillateur masse-ressort horizontal amorti par une force de frottement fluide, dont l'équation du mouvement s'écrit sous forme canonique

$$\ddot{x} + \frac{\omega_0}{Q}\,\dot{x} + {\omega_0}^2\,x = {\omega_0}^2\,x_{\text{éq}}\,.$$

On introduit alors $v = \dot{x}$, ce qui permet d'écrire l'équation sous la forme d'un système différentiel constitué de deux équations du premier ordre :

$$\begin{cases} \dot{x} = v \\ \dot{v} + \dfrac{\omega_0}{Q}v + {\omega_0}^2\,x = {\omega_0}^2\,x_{\text{éq}}\,. \end{cases}$$

On peut alors appliquer le schéma d'Euler pour ces deux équations,

$$\begin{cases} \dfrac{x_{n+1} - x_n}{\Delta t} = v_n \\ \dfrac{v_{n+1} - v_n}{\Delta t} = -\dfrac{\omega_0}{Q}v_n + {\omega_0}^2\,(x_{\text{éq}} - x_n) \end{cases}$$

et en déduire les deux relations de récurrence à implémenter dans la boucle **for**,

$$\begin{cases} x_{n+1} = x_n + v_n\,\Delta t \\ v_{n+1} = v_n + \Delta t\left({\omega_0}^2\,(x_{\text{éq}} - x_n) - \dfrac{\omega_0}{Q}v_n\right) \end{cases}$$

VI - Mise en pratique

Pour tous les exemples ci-dessous, on suppose avoir importé au préalable les bibliothèques `numpy` et `matplotlib` avec les alias classiques.

Application 1 : Pression atmosphérique

Dans la troposphère, couche occupant les 11 premiers kilomètres de l'atmosphère, on peut montrer que le champ de pression $P(z)$ varie avec l'altitude z selon l'équation différentielle

$$\frac{\mathrm{d}P}{\mathrm{d}z} = -\frac{Mg}{R(T_0 - \alpha z)}P$$

avec $M = 29 \cdot 10^{-3}\,\mathrm{kg \cdot mol^{-1}}$ la masse molaire de l'air, $g = 9{,}81\,\mathrm{m \cdot s^{-2}}$ l'accélération de la pesanteur, $R = 8{,}3\,\mathrm{J \cdot mol^{-1} \cdot K^{-1}}$ la constante des gaz parfaits, $T_0 = 288\,\mathrm{K}$ la température moyenne à la surface de la Terre et $\alpha = 6 \cdot 10^{-3}\,\mathrm{K \cdot m^{-1}}$ le gradient thermique.

1 - Écrire la relation de récurrence issue du schéma d'Euler explicite appliqué à cette équation différentielle.

2 - L'implémenter en Python pour trouver le champ de pression de 0 à 11 km d'altitude avec un pas de 1 m.

L'équation différentielle porte ici sur l'espace plutôt que sur le temps ... ce qui ne change rien de plus que les notations.

1 *L'altitude z est égale à $z_n = n\,\Delta z$, ce qui donne avec le schéma d'Euler explicite*

$$\frac{P_{n+1} - P_n}{\Delta z} = -\frac{Mg}{R(T_0 - \alpha\, z_n)}P_n \qquad \textit{soit} \qquad P_{n+1} = \left(1 - \frac{Mg}{R(T_0 - \alpha\, z_n)}\,\Delta z\right)P_n$$

2 *Une proposition d'implémentation utilisant des listes pourrait être la suivante :*

```
M = 29e-3          # en kg.mol-1
R = 8.3            # en J.mol-1.K-1
T0 = 288           # en K
alpha = 6e-3       # en K.m-1
g = 9.81           # en m.s-2

dz = 1             # pas de 1m
N = 11000          # nbre de pas d'espace : de 0 à 11000 m

z = [n*dz for n in range(N)]  # altitude, en m
P = [None for n in range(N)]  # pression, en Pa
P[0] = 1e5         # 1 bar au niveau du sol

for n in range(N-1):
    P[n+1] = (1 - (M * g * dz) / ( R * (T0-alpha*z[n]) ) ) * P[n]
```

Bien que ce ne soit pas demandé dans l'énoncé, il est toujours souhaitable de tracer ensuite le résultat des calculs, ne serait-ce que pour vérifier qu'il est vraisemblable. La syntaxe de base est rappelée ci-dessous, se reporter à la fiche outil Python pour améliorer la décoration.

```
plt.figure()
plt.plot(z,P)
```

Application 2 : Vidange d'un réservoir

Par application des lois de la mécanique des fluides, on peut montrer que la hauteur d'eau h dans un réservoir se vidant par un petit orifice situé à sa base vérifie l'équation différentielle suivante, appelée relation de Torricelli,

$$\frac{\mathrm{d}h}{\mathrm{d}t} = -\alpha\sqrt{2gh}$$

avec g l'accélération de la pesanteur et α le rapport des sections de l'orifice de vidange et du réservoir.

1 - Écrire la relation de récurrence issue du schéma d'Euler explicite appliqué à cette équation différentielle.

2 - L'implémenter en Python pour déterminer la hauteur d'eau restant dans le réservoir jusqu'à la vidange complète. On considérera le cas d'un réservoir d'eau de pluie prévu pour l'arrosage d'un potager : $\alpha = 1 \cdot 10^{-3}$, $h(t{=}0) = 1{,}50\,\mathrm{m}$, et un pas de temps $\Delta t = 0{,}1\,\mathrm{s}$.

1 *Sans difficulté, on trouve*

$$\frac{h_{n+1} - h_n}{\Delta t} = -\alpha\sqrt{2gh_n} \qquad \textit{soit} \qquad h_{n+1} = h_n - \Delta t\,\alpha\sqrt{2gh_n}\,.$$

2 *Devoir résoudre « jusqu'à » la vidange complète laisse entendre qu'il faut privilégier une implémentation avec une boucle* ***while****. Une proposition d'implémentation pourrait être la suivante :*

```
g = 9.81          # en m.s-2
alpha = 1e-3      # sans unité

# Initialisation des valeurs courantes :
tn = 0
hn = 1.5      # hauteur initiale 1m50

dt = .01      # pas de temps, en s

t = [tn]      # listes ne contenant que la valeur initiale
h = [hn]

while hn > 0:
    # Actualisation des valeurs courantes :
    tn += dt
    hn += -dt * alpha * np.sqrt(2*g*hn)

    # Nouveaux éléments ajoutés à chq pas de tps :
    t.append(tn)
    h.append(hn)
```

Application 3 : Pendule simple

L'angle formé par un pendule simple de longueur $\ell = 20\,\mathrm{cm}$ avec la verticale vérifie l'équation différentielle

$$\frac{\mathrm{d}^2\theta}{\mathrm{d}t^2} + \frac{g}{\ell}\sin\theta = 0\,.$$

1 - Écrire les relations de récurrence issues du schéma d'Euler explicite appliqué à cette équation différentielle.

2 - Analyser l'équation différentielle pour proposer une valeur de pas de temps raisonnable a priori.

3 - L'implémenter en Python pour résoudre l'équation sur une durée totale correspondant à trois oscillations. On supposera le pendule lâché sans vitesse initiale depuis un angle de 45°.

4 - Vérifier que, en augmentant la durée totale, la solution numérique finit toujours par diverger, et ce quel que soit le pas de temps, comme mentionné dans la dernière remarque du paragraphe III.

1 *L'équation différentielle étant du second ordre, il faut se ramener à un système différentiel en introduisant la vitesse angulaire $\omega = \dot{\theta}$. Le système différentiel équivalent s'écrit donc*

$$\begin{cases} \dot{\theta} = \omega \\ \dot{\omega} = -\dfrac{g}{\ell}\sin\theta \end{cases}$$

ce qui devient en version discrétisée

$$\begin{cases} \dfrac{\theta_{n+1}-\theta_n}{\Delta t} = \omega_n \\ \dfrac{\omega_{n+1}-\omega_n}{\Delta t} = -\dfrac{g}{\ell}\sin\theta_n \end{cases} \quad \text{soit} \quad \begin{cases} \theta_{n+1} = \theta_n + \Delta t\,\omega_n \\ \omega_{n+1} = \omega_n - \Delta t\,\dfrac{g}{\ell}\sin\theta_n \end{cases}$$

2 *L'équation différentielle fait apparaître la pulsation propre $\omega_0 = \sqrt{g/\ell}$, et donc la période propre $T_0 = 2\pi\sqrt{\ell/g}$ comme temps caractéristique. On peut donc proposer comme possible valeur raisonnable $\Delta t = T_0/1000$.*

3 *Un exemple d'implémentation, utilisant des tableaux* `numpy` *pour varier les plaisirs, pourrait être :*

```
g = 9.81 # en m.s-2
L = .2       # en m

T0 = 2 * np.pi * np.sqrt(L/g) # période propre
dt = T0/1000

N = 3*1000     # trois périodes de 1000 pas de temps

t = np.linspace(0,(N-1)*dt,N)     # N points entre 0 et (N-1)*dt
theta = np.empty_like(t)
omega = np.empty_like(t)

theta[0] = 45 * np.pi/180
omega[0] = 0

for n in range(N-1):
    theta[n+1] = theta[n] + dt * omega[n]
    omega[n+1] = omega[n] - dt * g/L * np.sin(theta[n])
```

4 *Le code précédent est utilisé pour tracer l'angle du pendule sur une durée égale à 15 périodes propres, avec différents pas de temps. On constate que, plus le pas de temps est élevé, plus la solution numérique diverge rapidement.*

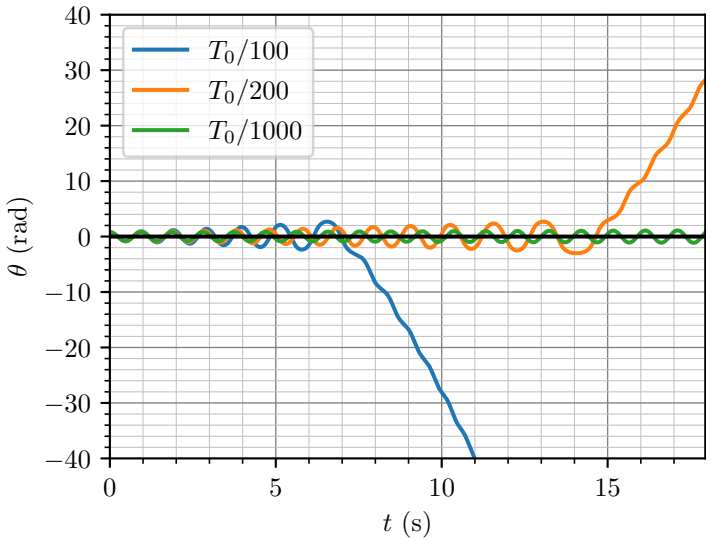

Figure 3 – Méthode d'Euler appliquée au pendule simple avec différents pas de temps. La période propre T_0 étant le temps caractéristique du système, c'est à elle qu'il faut comparer le pas de temps.