TP N° 2 : MÉTHODES D'EULER

Dans le TP, on travaille avec les tableaux de type NumPy [1].

Pour les interfaces graphiques, on utilisera le module `matplotlib.pyplot`

I. Méthodes d'Euler pour les EDO d'ordre 1

1) Méthode d'Euler explicite à pas constant

a) Programme de la méthode, en utilisant les tableaux Numpy

Écrire un programme `euler` de paramètres $f, y0, t0, tf$ et n :

– qui construit les listes $Lt = [t_0, \dots, t_n]$ et $Ly = [y_0, \dots, y_n]$ (voir cours pour les notations) ;
– qui affiche le graphe de la fonction affine par morceaux reliant les points de coordonnées (t_k, y_k).

– qui retourne le tuple (Lt, Ly).

b) Test 1

i) Tester `euler` sur l'exemple classique : $\begin{cases} \forall t \in [0,1], y'(t) = y(t) \\ y(0) = 1 \end{cases}$, sur les 3 valeurs de n :

$$n = 3 \qquad n = 10 \qquad n = 50$$

On tracera les 3 courbes sur un même graphique, ainsi que la courbe de la fonction exponentielle.

ii) Connaissant la solution exacte $x \mapsto \mathrm{e}^x$, écrire une fonction `erreur_exp(n)` qui quantifie l'erreur de la méthode numérique appliquée à l'équation du **i)** (*plusieurs indicateurs peuvent être proposés*).

c) Test 2

Tester `euler` sur l'exemple vu en cours : $\begin{cases} \forall t \in [0,20],\ y'(t) = t\sin(y(t)) \\ y(0) = 1 \end{cases}$, pour $n = 50$, puis $n = 80$.

On remarque l'instabilité du schéma numérique, pour les "grandes" valeurs de t.

2) Méthode d'Euler semi-implicite à pas constant

On rappelle son principe :

• On "discrétise" l'intervalle $[t_0, t_f]$ en n subdivisions régulières $[t_k, t_{k+1}]$, où $t_k = t_0 + k \times \dfrac{t_f - t_0}{n}$.

Posons $h = \dfrac{t_f - t_0}{n}$ le pas constant de cette discrétisation [2].

• On définit alors la suite $(y_k)_{0 \leqslant k \leqslant n}$ par : $\boxed{\begin{cases} y_0 = y(t_0) \\ \forall k \in [\![0, n-1]\!], \quad y_{k+1} = y_k + h \times f(t_k, y_{k+1}) \end{cases}}$

1. NumPy est une bibliothèque qui contient une collection d'algorithmes mathématiques et de fonctions enrichissant encore la puissance du langage Python, en fournissant à l'utilisateur un environnement de traitement de données et système de prototypage rivalisant avec Matlab, IDL, Octave, R-Lab, et SciLab.
2. Il existe encore une méthode à pas h adaptatif

a) Programme de la méthode, en utilisant les tableaux Numpy

Pour programmer ce schéma numérique, on a besoin d'une des méthodes numériques d'approximation du zéro d'une fonction (Newton ou dichotomie).
Écrire alors un programme `euler_imp` de paramètres $f, y0, t0, tf$ et n, et qui représente graphiquement la courbe obtenue par la méthode d'Euler semi-implicite.

b) Test de stabilité

On reprend le problème de Cauchy : $\begin{cases} \forall t \in [0,20],\ y'(t) = t\sin(y(t)) \\ y(0) = 1 \end{cases}$
Représenter les courbes obtenues par la méthode d'Euler implicite, pour $n = 50$, puis $n = 80$.
Remarquer le gain de stabilité.

II. Méthode d'Euler classique pour les EDO d'ordre 2

On vient de voir une première méthode d'Euler pour les équations différentielles du second ordre du type :

$$(E)\begin{cases} \forall t \in [t_0, t_f], \quad x''(t) = f(t, x(t), x'(t)) \\ x(t_0) = x_0 \\ x'(t_0) = v_0 \end{cases}$$

On utilise maintenant celle qui consiste à se ramener à un système différentiel d'EDO d'ordre 1.

1) Présentation de la méthode : voir cours

Écrire sur papier l'équation différentielle (E) sous la forme d'un système de deux équations différentielles du premier ordre 1, puis le schéma d'Euler permettant de calculer la position dans l'espace des phases (x, x') à l'instant t_{k+1} en fonction de la position à l'instant t_k.

2) Programmation de la méthode

Écrire une fonction `euler2(f,x0,v0,t0,tf,n)` retournant trois tableaux numpy x, v, t contenant respectivement les valeurs approchées de $x(t)$, $v(t)$ et t, pour t entre 0 et t_f, échantillonné sur $(n+1)$ nœuds de discrétisation.

3) Protocole de test : pendule simple (sans frottement)

On étudie :

$$(H)\begin{cases} \forall t \in [0,30],\ x''(t) + \sin(x(t)) = 0 \\ x(t_0) = x0 \\ x'(t_0) = 0 \end{cases}$$

1. Tester `euler2` sur (H) avec $n = 1000$ et $x0 = 1$.
2. • Tracer la solution numérique obtenue par `euler2`.
   • De même, tracer le portrait de phase.
   • Critiquer les résultats.
3. **Le problème de l'énergie mécanique**
   On définit l'énergie mécanique du système au temps t : $E_M(t) = \frac{1}{2}(v^2(t) + (1 - \cos(t))$.
   a) On prend toujours $n = 1000$.
   Définir la fonction `EM(t)` retournant l'énergie mécanique à partir des tableaux numpy x et v issus de `euler2`.

b) Tracer cette grandeur en fonction du temps. Que remarque-t-on ?

4. **Résolution du problème de stabilité**

   Proposer une méthode permettant de régler le problème de stabilité rencontré.

   Programmer alors numériquement cette méthode, et constater la conservation des amplitudes et de l'énergie mécanique du système au cours du temps.

5. Comparer la courbe obtenue avec cette dernière méthode et celle donnée par la fonction structurée `odeint()`.

   Extrait de l'aide :

```
odeint(func, y0, t)

Integrate a system of ordinary differential equations.
Solve a system of ordinary differential equations:  dy/dt = func(y,t0)
where y can be a vector.

Parameters :
func : callable(y, t0)
Computes the derivative of y at t0.
y0 : array
Initial condition on y (can be a vector).
t : array
A sequence of time points for which to solve for y. The initial value point should be
the first element of this sequence.

Returns :
y : array, shape (len(t), len(y0))
Array containing the value of y for each desired time in t, with the initial value y0
in the first row.
```

6. **Estimation de la période**

   a) Écrire une fonction `detect0(t,x)` qui retourne une liste contenant tous les temps de passages par $x = 0$.

   b) L'utiliser pour calculer une valeur approchée de la période des oscillations, dans le cas de l'oscillateur harmonique (pour $n = 10000$).

4) Protocole de test : pendule simple avec frottements

Reprendre le travaul précédent sur l'EDO :

$$(K)\begin{cases}\forall t \in [0,30],\ x''(t) + 2\lambda x'(t) + \omega_0^2 \sin(x(t)) = 0\\ x(t_0) = x0\\ x'(t_0) = 0\end{cases}$$

où $\lambda > 0$ est le coefficient de frottement et $\omega_0 > 0$ est la pulsation propre.