TP3 : Équations différentielles ordinaires (EDO)

- Problèmes aux valeurs initiales -

F. Saïd

---

1 Équation différentielle ordinaire (EDO)

— Une équation différentielle est une relation entre une fonction, f, sa variable indépendante, x, et un nombre quelconque de ses dérivées.

— Une équation différentielle ordinaire ou ODE est une équation différentielle où la variable indépendante x, la fonction f et ses dérivées sont à une dimension.

— On adoptera dans la suite l'écriture suivante des EDO :

$$\frac{d^n f(x)}{dx^n} = F\left(x, f(x), \frac{df(x)}{dx}, \frac{d^2 f(x)}{dx^2}, \cdots, \frac{d^{n-1} f(x)}{dx^{n-1}}\right)$$

où F est une fonction arbitraire et n est l'ordre de l'équation différentielle. Cette équation est appelée ODE d'ordre n.

— Pour une ODE d'ordre n, les valeurs initiales sont des valeurs connues pour les dérivées 0 à $(n-1)$ en $x = x_0$, à savoir $f(x_0)$, $f'(x_0)$, $f''(x_0)$, . . . , $f^{(n-1)}(x_0)$.

— Pour une certaine classe d'équations différentielles ordinaires (équations différentielles linéaires par exemple), les valeurs initiales sont suffisantes pour trouver une équation spécifique unique.

— Trouver une solution à une EDO donnée par des valeurs initiales est appelé problème aux valeurs initiales.

2 Calcul symbolique dans Python et résolution d'EDO

Il y a plus d'équations différentielles que l'on ne sait pas résoudre algébriquement que d'équations que l'on sait résoudre. Python dispose d'une bibliothèque de calcul symbolique, *sympy*, qui permet de déterminer les solutions analytiques d'une EDO lorsque c'est possible.
Le code donné dans le Listing 1 en donne une illustration.

```
1  from sympy import *
2
3  x,C1=symbols('x,C1')           #dictionnaire de symboles
4  f=symbols('f', cls=Function)   #désigne f comme symbole de fonction
5
6  eq=Eq(f(x).diff(x,1),f(x)+x ) #définit l'équation à résoudre f'(x)=f(x)+x.
7                  #f(x).diff(x,1) désigne df(x)/dx soit f'(x)
8  print (eq)                     #affiche l'équation à résoudre
9  print('Classe EDO : ', classify_ode(eq)[0])) #caractérise l'EDO à résoudre
10                 #(linéaire, séparable, exacte...)
11
12 sol = dsolve(eq)               #résout l'EDO sans conditions initiales
13 print('La solution générale est : ',sol.rhs)#affiche la solution générale
14
15 sol_spe=dsolve(eq, ics={f(0): 1}) #résout l'EDO avec la condition initiale f(0)=1
16 print('La solution spécifique est : ',sol_spe.rhs) #affiche la solution spécifique
```

Listing 1 – Résolution analytique d'une EDO avec Sympy

Exercice 1 : Résoudre analytiquement les équations différentielles qui suivent (vues en TD)

1. $y' + 3y = 2$ avec $y(0) = 1$
2. $xy' - 2y + x\ln(x) = 0$
3. $xy' = y$
4. $y' - \dfrac{2y}{x+1} = (x+1)^3$

3 Champ de directions

Soit l'équation différentielle de premier ordre $\dfrac{df(x)}{dx} = F(x, f(x))$.

- En chaque point $M(x, y)$ du plan, on peut tracer une ligne de direction de pente $F(x, y)$.
- Si on se donne un quadrillage d'une portion du plan et qu'on trace en chaque point du quadrillage la ligne de direction associée, on visualise le champ de direction associé à l'équation différentielle.
- Ce champ de direction permet de connaître l'allure des solutions de l'EDO, même lorsqu'il est impossible de les déterminer analytiquement.

Exercice 1 :

1. Utiliser le script donné dans le Listing 2 pour représenter, sur un même graphique, le champ de directions associé sur $[-4, 4] \times [-4, 4]$ à l'équation différentielle (E), et la solution qui vérifie spécifiquement $y(0) = 0$:

$$y' = x + y \qquad (E)$$

avec y une fonction de la variable réelle x.

```
# Soit à représenter le champ de directions de l'équation y'=F(y,x)

#Import des bibliothèques
from matplotlib.pylab import *
from scipy.integrate import odeint
import numpy as np

# Saisie de l'équation différentielle
# def F(y,x):
# return y+x
# ou
F = lambda y,x : y+x  #définit la fonction F

# Solution exacte de l'EDO
x= linspace(-4,2,100)
sol=np.exp(x)-x-1

# Definit une grille et calcule la direction (ou pente) en chaque point
g1 = linspace(-4,4,12)
g2 = linspace(-4,4,12)
X,Y = meshgrid(g1,g2) #crée la grille
DX, DY = 1, F(Y,X)    #calcule les taux d'accroissement en chaque point de la grille
                      #accroissement de 1 dans la direction de x et Dy dans la direction de y
M = sqrt(DX**2+DY**2) #normes des vecteurs de directions
M[ M==0 ] = 1         #évite les divisions par zéro

#Représentations graphiques
figure(figsize=(10,7))#taille de la fenêtre graphique
quiver(X,Y, DX/M, DY/M, M, pivot='mid') #représente le champ de directions normalisé
plot(x,sol,'r')       #représente la solution exacte
grid()                #affiche la grille
xlabel('x')           #affiche une étiquette en abscisse
ylabel('y')           #affiche une étiquette en ordonnée
title('Champ de directions')#ajoute un titre
show()                #affiche la figure à l'écran
```

Listing 2 – Illustration du champ de directions

2. Vu en TD : Représenter, sur un même graphique, le champ de directions associé à l'équation différentielle (E) et la solution qui vérifie spécifiquement $y(2) = 3$:

$$9yy' + x = 0 \qquad (E)$$

avec y une fonction de la variable réelle x.

4 Méthode d'Euler explicite

On considère l'ODE d'ordre 1 :

$$\frac{df(x)}{dx} = F(x, f(x)) \tag{1}$$

et $\{x_0, \cdots, x_{j-1}, x_j, x_{j+1}, \cdots, x_n\}$ un maillage de l'intervalle $[a, b]$ avec un pas h.

L'approximation linéaire de $f(x)$ dans le voisinage de x_j s'écrit :

$$f(x) = f(x_j) + (x - x_j)\frac{df(x_j)}{dx}$$

soit en utilisant l'expression dans (1),

$$f(x) = f(x_j) + (x - x_j)F(x_j, f(x_j))$$

Nous avons en particulier, au point x_{j+1} :

$$f(x_{j+1}) = f(x_j) + (x_{j+1} - x_j)F(x_j, f(x_j))$$

soit,

$$\boxed{f(x_{j+1}) = f(x_j) + hF(x_j, f(x_j))}$$

Cette formule est appelée **formule d'Euler explicite**. Elle permet de calculer une approximation de f en x_{j+1} connaissant f en x_j.

Exercice 1 :

1. Utiliser le script donné en Listing 3 pour résoudre par le schéma d'Euler explicite, le problème aux valeurs initiales (S) sur l'intervalle $[0, 1]$:

$$\begin{cases} \dfrac{df(x)}{dx} = e^{-x} \\ \\ f(0) = -1, \quad h = 0.1 \end{cases} \tag{S}$$

2. Résoudre le problème (S) pour un pas $h = 0.01$ puis $h = 10^{-3}$. Que constatez-vous ?
3. Vu en TD : utiliser le schéma d'Euler explicite pour résoudre le problème $(S1)$ sur l'intervalle $[1, 2]$:

$$\begin{cases} x^2\dfrac{df(x)}{dx} + (1 - 2x)f(x) - x^2 = 0 \\ \\ x_0 = 1, \quad f(x_0) = 1, \quad h = 0.01 \end{cases} \tag{S1}$$

Représenter la solution approchée obtenue.

```python
# Soit à résoudre l'EDO y'=F(y,x) avec le schéma d'Euler explicite

#Import des bibliothèques
import matplotlib.pyplot as plt
import numpy as np
plt.style.use("seaborn-poster")

# Saisie de l'équation différentielle
# def F(y,x):
# return np.exp(-x)
# ou
F = lambda x, y : np.exp(-x)  #définit la fonction F

#Construction du maillage
h=0.1                                                   #pas du maillage
x=np.arange(0,1+h,h)                                    #maillage
y0=-1                                                   #condition initiale

#Schéma d'Euler explicite
y=np.zeros(len(x))
y[0]=y0

for i in range(0,len(x)-1):
  y[i+1]=y[i]+h*F(x[i],y[i])

#représentations graphiques
plt.figure(figsize=(12,8))                              #taille de la fenêtre graphique
plt.plot(x,y,"b-", label="Solution approchée")          #calcule la solution approchée
plt.plot(x,-np.exp(-x),"g", label="Solution exacte")    #calcule la solution exacte
plt.xlabel('x')                                         #affiche une étiquette en x
plt.ylabel('y')                                         #affiche une étiquette en y
plt.grid()                                              #affiche une grille
plt.legend(loc='lower right')                           #affiche une légende
plt.show()                                              #affiche le graphique
```

Listing 3 – Illustration du schéma d'Euler explicite

5 Solveur ODE de Python

Point vocabulaire : on désigne une équation différente par :
- — EDO en français pour Equation différentielle Ordinaire
- — ODE en anglais pour Ordinary Differential Equation

La bibliothèque **SciPy** de Python possède plusieurs fonctions pour la résolution de problèmes à valeurs initiales. Nous nous proposons de mettre en oeuvre dans la suite la fonction **scipy.integrate.solve_ivp**.

Exercice 1 : Utiliser le solveur de Python, *solve_ivp*, pour résoudre sur $[0, \pi]$ l'équation différentielle :

$$\frac{df(t)}{dt} = \cos(t)$$

avec une valeur initiale $f(0) = 0$. Représenter la solution exacte versus la solution approchée, ainsi que l'évolution de l'erreur relative dans le temps.

Indication : bien comprendre le code donné dans le Listing 4 et l'utiliser pour répondre aux questions posées.

```python
# Soit à résoudre avec solve_ivp l'EDO y'=F(y,t)

#Import des bibliothèques
import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import solve_ivp
plt.style.use("seaborn-poster")

# Saisie de l'équation différentielle
# def F(y,t):
# return np.cos(t)
# ou
F = lambda y,t : np.cos(t)  #définit l'EDO

#Construction du maillage
t_eval=np.arange(0,np.pi,0.1)

#Résolution avec le solveur
sol=solve_ivp(F,[0,np.pi],[0],t_eval=t_eval)

#représentations graphiques
plt.figure(figsize=(12,4))  #taille de la fenêtre graphique

plt.subplot(121)            #ouvre une sous-fenêtre graphique
plt.plot(sol.t,sol.y[0])    #représente la solution approchée
plt.xlabel('t')             #affiche une étiquette en abscisse
plt.ylabel('f(t)')          #affiche une étiquette en ordonnée

plt.subplot(122)            #ouvre une sous-fenêtre graphique
plt.plot(sol.t,sol.y[0]-np.sin(sol.t))#représente l'erreur commise
plt.xlabel('t')             #affiche une étiquette en abscisse
plt.ylabel('f(t)-sin(t)')   #affiche une étiquette en ordonnée

plt.tight_layout()          #définit une option graphique
plt.show()                  #affiche le graphique
```

Listing 4 – Illustration du solveur ODE de Python

Exercice 2 :

Dans l'exercice précédent, vous avez pu résoudre l'équation différentielle $\frac{df(t)}{dt} = \cos(t)$ en utilisant le solveur ODE de Python, *solve_ivp*. Vous avez pu constater dans les figures produites que les écarts relatifs et absolus entre la solution approchée et la solution analytique sont petits. Ces écarts peuvent être contrôlés en utilisant les arguments *rtol* et *atol* respectivement ; le solveur maintient les estimations des erreurs locales inférieures à $atol + rtol \times |f|$. Les valeurs par défaut sont $1e-3$ pour *rtol* et $1e-6$ pour *atol*.

Reprendre l'exercice précédent avec $rtol = 1e-8$ et $atol = 1e-8$. Commenter les résultats.

Exercice 3 : Utiliser *solve_ivp* pour résoudre sur $[0, 1]$ le problème à valeur initiale :

$$\frac{df(t)}{dt} = -f(t) \quad \text{avec } f(0) = 1$$

Représenter la solution approchée versus la solution exacte et l'évolution de l'erreur relative dans le temps.
Indication : La solution exacte est donnée par $f(t) = e^{-t}$.

6 Compléments

1. En Python, une fonction lambda est une fonction d'une seule ligne déclarée sans nom, qui peut avoir un nombre quelconque d'arguments, mais elle ne peut avoir qu'une seule expression. Il arrive souvent qu'une fonction lambda soit passée en argument d'une autre fonction.
2. Utiliser, si besoin, le compileur Python en ligne : Codabrainy