Chapitre de révision 4

Equations différentielles

4.1 Equations différentielles du 1er ordre

Soit ϕ une application réelle définie sur une partie de $\mathbb{R}^2$. Résoudre, ou intégrer, une **équation différentielle d'ordre 1** de la forme :

$$y' = \phi(y, x) \qquad (E)$$

C'est déterminer toutes les application y définies et dérivables sur un ouvert I de $\mathbb{R}$ et vérifiant :

$$\forall\, x \in I, \quad y'(x) = \phi(y(x), x)$$

• Résoudre une équation différentielle $y' = \phi(y, x)$ avec condition initiale $y_0 = y(x_0)$ qui peut s'écrire aussi (x_0, y_0) (avec (x_0, y_0) dans le domaine D de ϕ) consiste à déterminer une solution y à l'équation différentielle vérifiant $y(x_0) = y_0$. On parle aussi de **Problème de Cauchy**.

- Sur un segment $[a, b]$, lorsqu'elle existe, une solution à un problème de Cauchy est unique.
- Le **Théorème de Cauchy-Lipschitz** donne des conditions suffisantes sur ϕ pour qu'une solution au problème de Cauchy existe.

4.2 Intégration d'équations différentielle sous `scipy`

4.2.1 Fonction `odeint`

• La fonction `odeint` du sous-module `integrate` de `scipy` permet l'intégration d'équations différentielles avec condition initiale :

• **Principe :** lorsqu'une solution existe,

Pour résoudre sur l'intervalle $[a, b]$ le problème de Cauchy :

$$y' = \phi(y, t) \quad \text{avec } y(a) = y_0$$

```
import numpy as np
from scipy import integrate
a, b = ..., ...    # Bornes d'intégrations à renseigner
phi = lambda y,t: ...    # fonction second membre à renseigner
y0 = ...            # condition initiale à renseigner
N = ...             # Nombre de points à renseigner
T = np.linspace(a,b,N)    # tableau des temps
Y = integrate.odeint(phi, y0, t)    # Y = tableau des y(T[k])
# tracé
import matplotlib.pyplot as plt
plt.plot(T,Y)
```

4.2.2 Equa. diff. d'ordre supérieur

• La fonction `integrate.odeint` permet aussi de résoudre des systèmes d'équations différentielles (2 ou plus) :

$$\begin{cases} y_1' = f_1(y_1, t) \\ y_2' = f_2(y_2, t) \end{cases} \qquad \text{avec conditions initiales } y_1(x_0) \text{ et } y_2(x_0)$$

Pour cela on passera pour arguments à `odeint` : `integrate.odeint(F,Y0,t)` avec `F` et `Y0` :

`F = np.array([f1,f2])` et `Y0 = np.array([y1(x0),y2(x0)])`.

• C'est cette fonctionnalité que l'on utilise pour résoudre des équations différentielles d'ordre supérieur :

• Exemple : $y'' = \cos(t)$ avec $y(0) = -1$ et $y'(0) = 0$
a pour solution évidente $y(t) = -\cos(t)$.

L'équation différentielle est équivalente au système (problème de Cauchy) :

$$\begin{cases} \frac{d}{dt}y = y' \\ \frac{d}{dt}y' = \cos(t) \end{cases} \quad \text{avec : } y(0) = -1 \text{ et } y'(0) = 0$$

- Le système :

$$\begin{cases} \frac{d}{dt}y = y' \\ \frac{d}{dt}y' = \cos(t) \end{cases} \quad \text{avec : } y(0) = -1 \text{ et } y'(0) = 0$$

s'écrit comme un problème de Cauchy vectoriel :

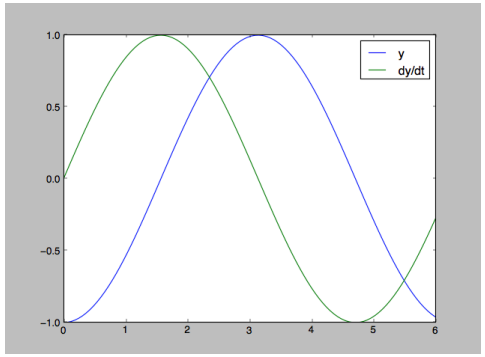
$$\frac{d}{dt}\begin{pmatrix} y \\ y' \end{pmatrix} = \begin{pmatrix} y' \\ \cos(t) \end{pmatrix} \quad \text{et} \quad \begin{pmatrix} y(0) \\ y'(0) \end{pmatrix} = \begin{pmatrix} -1 \\ 0 \end{pmatrix}$$

En posant $Y = \begin{pmatrix} y \\ y' \end{pmatrix} = \begin{pmatrix} Y[0] \\ Y[1] \end{pmatrix}$ et $\phi : \begin{pmatrix} Y[0] \\ Y[1] \end{pmatrix}, t \longmapsto \begin{pmatrix} Y[1] \\ \cos(t) \end{pmatrix}$

```
import numpy as np
F = lambda Y, t : np.array([Y[1], np.cos(t)])
from scipy import integrate
t = np.linspace(0,6,100)    # Subdivision régulière de [0,6]
y0 = np.array([-1,0])        # array des conditions initiales
y = integrate.odeint(F,y0,t)    # appel de odeint
```

Tracé du graphe :

```
plt.plot(t,y)
plt.legend(('y','dy/dt'))
plt.show()
```



On obtient 2 courbes, celle de $y(t) = -\cos(t)$ et celle de $y'(t) = \sin(t)$.

4.3 Méthode d'Euler

4.3.1 Méthode d'Euler scalaire

Soit l'équation différentielle $f' = \phi(f, x)$ où ϕ est définie sur $I \times \mathbb{R}$ avec I un intervalle ouvert non-vide de $\mathbb{R}$.

Soit un segment $[a, b]$ inclus dans I et la condition initiale $(x_0, y_0) = (a, f(a))$. La méthode d'Euler-Cauchy consiste à considérer une subdivision régulière de $[a, b]$ en n segments, soit $n+1$ points $x_k = a + k\frac{b-a}{n}$ et à calculer de proche en proche une valeur approchée de $y_k = f(x_k)$ **en approchant la fonction f par son DL d'ordre 1 en x_k (tangente)**.

La suite $(y_k)_{0 \leqslant k \leqslant n}$ est ce que l'on entend par une résolution approchée du problème de Cauchy.

- C'est à dire, une fois y_k déterminé, on trouve y_{k+1} par :

$$y_{k+1} = y_k + \int_{x_k}^{x_{k+1}} f'(x)dx = y_k + \int_{x_k}^{x_{k+1}} \phi(f(t), t)dt$$

$$\simeq y_k + f'(x_k)(x_{k+1} - x_k) \simeq \boxed{y_k + \phi(y_k, x_k).\frac{b-a}{n} \simeq y_{k+1}}$$

4.3.2 Code

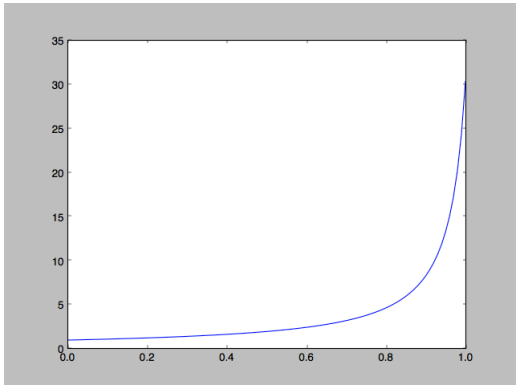
```
import numpy as np
import matplotlib.pyplot as plt
a, b = ..., ...    # Bornes à renseigner
Phi = lambda y,x : ....    # Fonction second membre à
renseigner
y0 = ...        # Condition initiale à renseigner
X = np.linspace(a,b,n+1)    # Subdivision régulière de [a,b]
Y = np.empty(n+1)         # tableau (1D) vide de n+1 elts
Y[0] = y0                 # Condition initiale
pas = (b-a)/n
for k in range(n):
    Y[k+1] = Y[k] + pas * Phi(Y[k],X[k])
plt.plot(X, Y)    # Tracé
```

- Attention `np.empty()` retournant un tableau unidimensionnel, cette écriture de la méthode d'Euler-Cauchy ne fonctionne qu'avec des équations différentielles scalaires du premier ordre (nous verrons plus loin comment en donner une version vectorielle : remplacer par une liste que l'on remplit successivement de vecteurs).

4.3.3 Exemple

• Intégration de $y' = y^2$ avec $y(0) = 1$ (essai sur $[0, 1]$).

```
Phi = lambda y,x : y**2
n = 100
X = np.linspace(0,1,n+1)
Y = np.empty(n+1)
Y[0] = 1
pas = 1/n
for k in range(n):
    Y[k+1] = Y[k] + pas * Phi(Y[k],X[k])
plt.plot(X, Y)    # Tracé
```



La solution exacte est $y(x) = \dfrac{1}{1-x}$ (qui n'est pas définie en $x = 1$...).

4.3.4 Méthode d'Euler vectorielle

Elle prend pour paramètre :

1. La fonction Φ vectorielle,
2. Les conditions initiales, sous forme d'un vecteur Y_0,
3. Les bornes a, b de résolution,
4. le pas h (ou le nombre de points $n+1$ avec $h = \frac{b-a}{n}$, au choix).

```
# Euler vectoriel
def EulerVect(F,y0,a,b,h):
    y = y0
    t = a
    T = [a]
    Y = [y0]
    while t+h <= b:
        y = y + h * F(y,t)
        Y.append(y)
        t += h
        T.append(t)
    return T, Y
```

4.3.5 Exemple

• Exemple : $y'' = \cos(t)$ avec $y(0) = -1$ et $y'(0) = 0$ (vue précedemment).

s'écrit comme un problème de Cauchy vectoriel :

$$\frac{d}{dt}\begin{pmatrix} y \\ y' \end{pmatrix} = \begin{pmatrix} y' \\ \cos(t) \end{pmatrix} \quad \text{et} \quad \begin{pmatrix} y(0) \\ y'(0) \end{pmatrix} = \begin{pmatrix} -1 \\ 0 \end{pmatrix}$$

En posant $Y = \begin{pmatrix} y \\ y' \end{pmatrix} = \begin{pmatrix} Y[0] \\ Y[1] \end{pmatrix}$ et $\phi : \begin{pmatrix} Y[0] \\ Y[1] \end{pmatrix}, t \longmapsto \begin{pmatrix} Y[1] \\ \cos(t) \end{pmatrix}$

```
import numpy as np
import matplotlib.pyplot as plt
F = lambda Y, t : np.array([Y[1], np.cos(t)])
a, b = 0, 6
h = 1e-2
y0 = np.array([-1,0])          # array des conditions initiales
X,Y = EulerVect(F,y0,a,b,h)    # appel de EulerVect
plt.plot(T,Y)
```

4.4 Exemples

4.4.1 Cinétique chimique

• Prenons le cas d'une solution composée de réactifs A, B, C, où deux réactions chimiques $A \longrightarrow B$ et $B \longrightarrow C$ d'ordre 1 rentrent en jeu selon les

lois.

$$\begin{cases} \frac{d[A]}{dt} &= -[A] \\ \frac{d[B]}{dt} &= [A]-[B] \\ \frac{d[C]}{dt} &= [B] \end{cases} \iff \frac{d}{dt}\begin{pmatrix}[A]\\[B]\\[C]\end{pmatrix} = \begin{pmatrix}-[A]\\[A]-[B]\\[B]\end{pmatrix}$$

On s'intéresse à l'évolution des concentrations des 3 réactifs A , B, C entre 0 et 8s, en partant des concentrations initiales $[A]=1$, $[B]=[C]=0$.

• Résolution :

```
import numpy as np
Phi = lambda Y, t: np.array([-Y[0], Y[0] - Y[1], Y[1]])
Y0 = np.array([1, 0, 0])
```

```
T, Sol = EulerVect(Phi,Y0,0,8,0.1)
```

• Avec `odeint` :

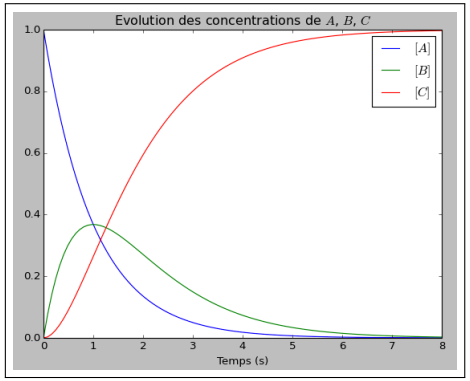
```
T = np.arange(0,8,0.1) ; Sol = integrate.odeint(Phi,Y0,t)
```

• Résolution par la méthode d'Euler scalaire.

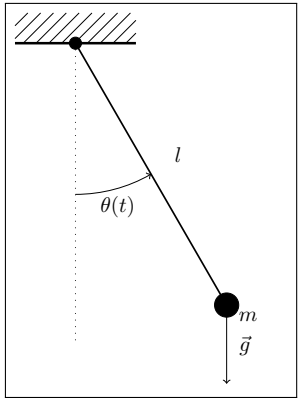
```
n = 800
pas = 8/n
A, B, C = [None]*(n+1), [None]*(n+1), [None]*(n+1)
A[0], B[0], C[0] = 1, 0, 0
T = [0] + [None]*n
for k in range(n):
    T[k+1] = T[k] + pas
    A[k+1] = A[k] + pas *(-A[k])
    B[k+1] = B[k] + pas *(A[k]-B[k])
    C[k+1] = C[k] + pas * B[k]
Sol = [A, B, C]
```

• Tracé.

```
import matplotlib.pyplot as plt
plt.plot(T, Sol)
plt.title('Evolution des concentrations de $A$, $B$, $C$')
plt.legend(('$[A]$', '$[B]$', '$[C]$'))
plt.xlabel('Temps (s)')
plt.show()
```



4.4.2 Oscillateur harmonique : pendule sans frottement



• Un pendule de masse m soumis à un champ de pesanteur constant se déplace dans un plan vertical ; il est situé au bout d'une tige rigide de longueur l et de masse nulle tournant sans frottement autour de son extrémité fixe.

Résoudre numériquement l'équation différentielle entre 0 et 6s avec une position initiale égale à $\theta(0)=\frac{\pi}{2}$, une vitesse initiale nulle, et avec $g=9,81 m.s^{-1}$ et $l=10cm$.
L'angle θ vérifie l'équation différentielle du second ordre :

$$\boxed{\ddot{\theta} = -\frac{g}{l}\sin\theta}$$

$$\Longrightarrow \boxed{\frac{d}{dt}\begin{pmatrix}\theta(t)\\ \dot{\theta}(t)\end{pmatrix} = \begin{pmatrix}\dot{\theta}(t)\\ -\frac{g}{l}\sin(\theta(t))\end{pmatrix}}$$

avec : $\boxed{\theta(0) = \frac{\pi}{2}}$ et $\boxed{\dot{\theta}(0) = 0}$.

- Résolution numérique avec $\theta(0) = \frac{\pi}{2}$, vitesse initiale nulle et $l = 10cm$.

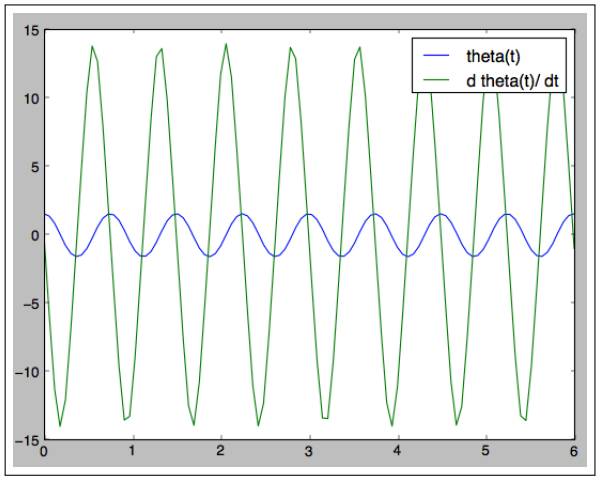
```
import numpy as np
import matplotlib.pyplot as plt
g = 9.81
l = 0.1
F = lambda Y,t: np.array([Y[1] , -g * np.sin(Y[0]) / l])
y0 = np.array([np.pi/2 , 0])    # Condition intiale
t, y = EulerVect(F,y0,0,6,0.01)    # Résolution
plt.plot(t,y)
plt.legend(('theta(t)','d theta(t)/ dt'))
plt.show()
```

On utilise comme fonction :

$$\boxed{F : \begin{pmatrix} y_0 \\ y_1 \end{pmatrix}, t \longmapsto \begin{pmatrix} y_1 \\ -\frac{g}{l}\sin(y_0) \end{pmatrix}} \quad \text{avec } \frac{d}{dt}Y = F(Y,t)$$

- Résolution numérique avec Euler scalaire :

```
n = 600
pas = 6/n
T = np.empty(n+1)
Th = np.empty(n+1)
dTh = np.empty(n+1)
T[0] = 0
Th[0] = np.pi/2
dTh[0] = 0
for k in range(n):
    T[k+1] = T[k] + pas
    Th[k+1] = Th[k] + pas*dTh[k]
    dTh[k+1] = dTh[k] + pas * (-g + np.sin(Th[k])/l)
Y = [Th, dTh] plt.plot(T,Y)
plt.legend(('theta(t)','d theta(t)/ dt'))
plt.show()
```



Essayer la commande (LaTeX) :

```
plt.legend(('$\theta(t)$','$\frac{\theta(t)}{dt}$'))
```

pour obtenir un plus joli résultat.