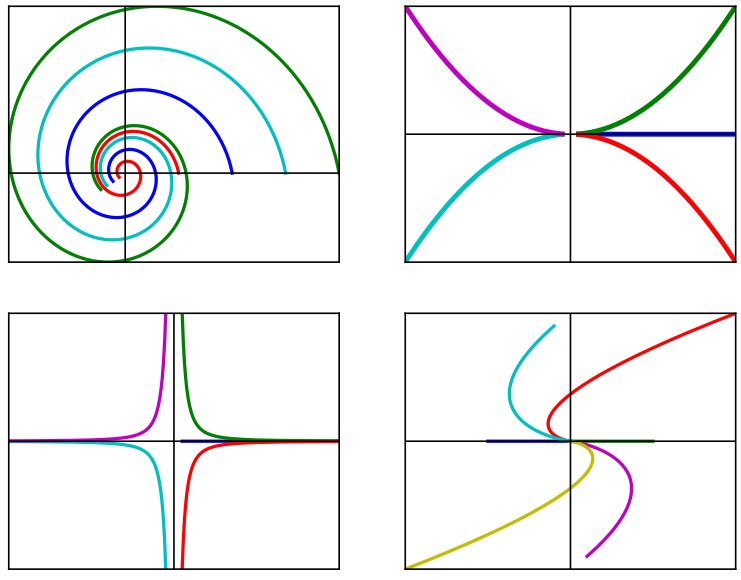
Équations différentielles



Skander Zannad et Judicaël Courant

Lycée La Martinière-Monplaisir

2014-01-18

1 Cadre : Problème de Cauchy

$$y' = F(t, y) \tag{1}$$

avec la condition initiale $y(t_0) = y_0$.

Où :

- y et F sont à valeurs dans $E = \mathbb{R}^n$ ou $E = \mathbb{C}^n$ ($n \in \mathbb{N}^*$) ;
- F est définie sur $I \times E$ où I intervalle de $\mathbb{R}$;
- $t_0 \in I$ et $y_0 \in E$.

2 Notion de solution

Donnée :

- d'un sous-intervalle J de I contenant t_0
- et de $y : J \to E$ dérivable vérifiant $y(t_0) = y_0$ et

$$\forall t \in J \quad y'(t) = F(t, y(t))$$

Solution *maximale* : solution sur un intervalle J qu'il est impossible de prolonger en une solution sur un intervalle strictement grand.

3 Exemple

Pour

$$\begin{array}{rcl} [0,1] & \to & \mathbb{R} \\ t & \mapsto & 2e^t \end{array}$$

est solution de l'équation (1) avec la condition initiale $y(1) = 2e$ où

$$\begin{array}{rcccl} F: & \mathbb{R}\times\mathbb{R} & \to & \mathbb{R} \\ & (t,y) & \mapsto & y \end{array}$$

mais solution *non maximale* car prolongeable (par exemple) en

$$\begin{array}{rcl} [0,17] & \to & \mathbb{R} \\ t & \mapsto & 2e^t \end{array}$$

Seule solution maximale dans ce cas :

$$\begin{array}{rcl} \mathbb{R} & \to & \mathbb{R} \\ t & \mapsto & 2e^t \end{array}$$

4 Exemple (bis)

On cherche à résoudre sur $\mathbb{R}$ l'équation

$$y' = \frac{3}{7}y^3$$

avec condition initiale $y(0) = \frac{1}{6}$

- Il n'y a pas de solution définie sur $\mathbb{R}$ tout entier (même en changeant la c.i., sauf pour une c.i. $y(t_0) = 0$) ;
- Il y a une unique solution maximale : $t \mapsto \sqrt{\frac{7}{252-6t}}$;
- Elle est définie sur $]-\infty, 42[$.

5 Théorème de Cauchy-Lipschitz

Sous des hypothèses raisonnables sur F :

Existence et unicité d'une solution maximale pour le problème de Cauchy.

Attention : Pas nécessairement définie sur tout l'intervalle où on cherche.

6 En pratique

Bien souvent en pratique, $I = [a, b]$ (où $a, b \in \mathbb{R}$ avec $a < b$) et les solutions seront définies sur I tout entier.

On se placera dans ce cadre pour la suite de ce cours. On supposera que la condition initiale est donnée en a et on notera y_0 la valeur initiale de y en a. On note y l'unique solution maximale du problème de Cauchy.

7 La méthode d'Euler

On choisit un *pas* $h = \frac{b-a}{n}$, où $n \in \mathbb{N}^*$.

On subdivise l'intervalle $[a, b]$, en posant $t_k = a + kh$ pour $k = 1, \ldots, n$ ($t_0 = a$, $t_n = b$)

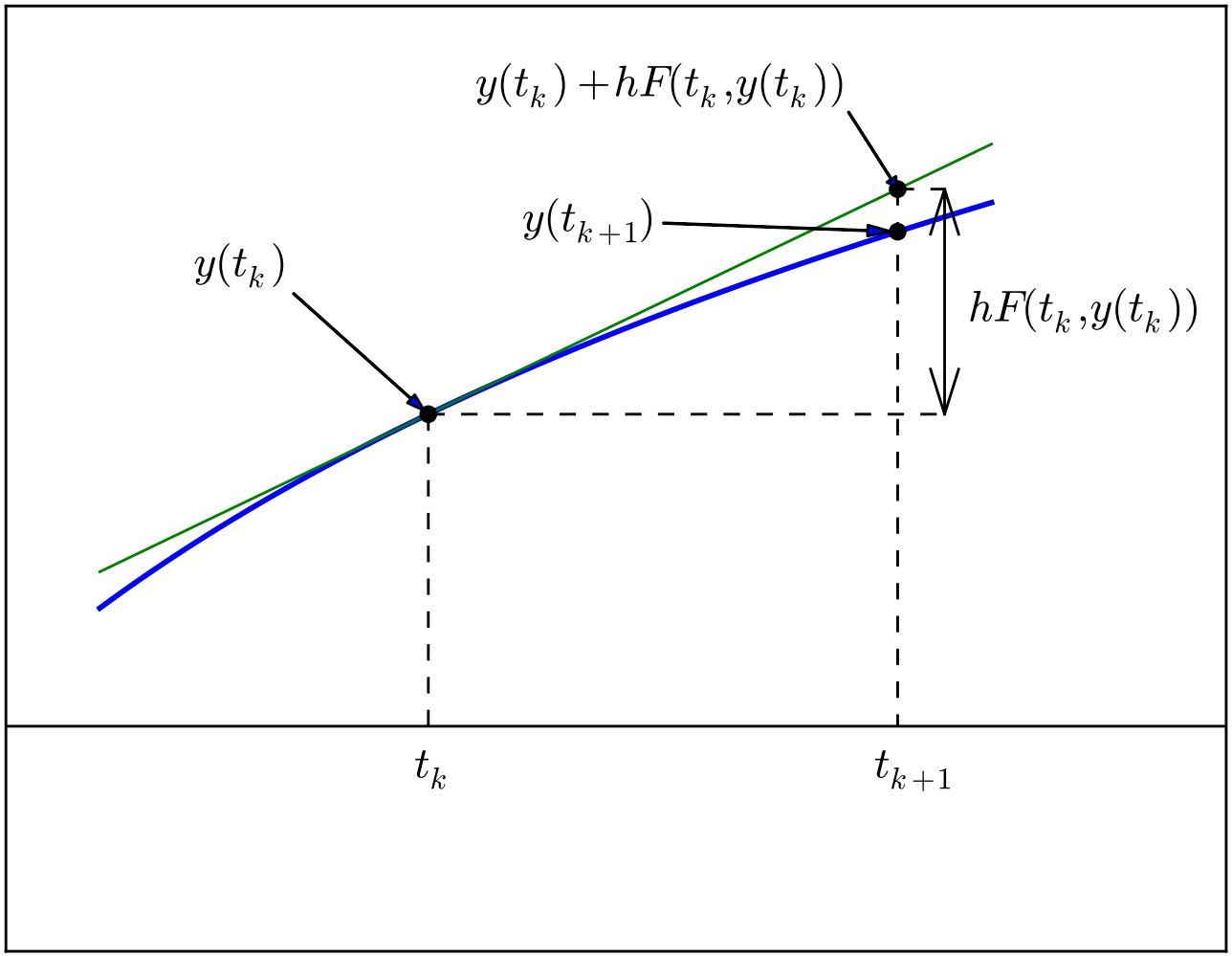
On sait que pour tout $k \in [\![0, n[\![$, il existe c_k tel que

$$\begin{aligned} y(t_{k+1}) - y(t_k) &= (t_{k+1} - t_k)y'(c_k) = hF(c_k, y(c_k)) \\ &\approx hF(t_k, y(t_k)) \end{aligned}$$

Ce qui conduit à poser pour $k \in 0, n$

$$y_{k+1} = y_k + hF(t_k, y_k)$$

(on espère qu'on aura $y_k \approx y(t_k)$ pour tout k).


$y(t_k) + hF(t_k, y(t_k))$
$y(t_{k+1})$
$y(t_k)$
$hF(t_k, y(t_k))$
t_k
t_{k+1}

8 Exemple

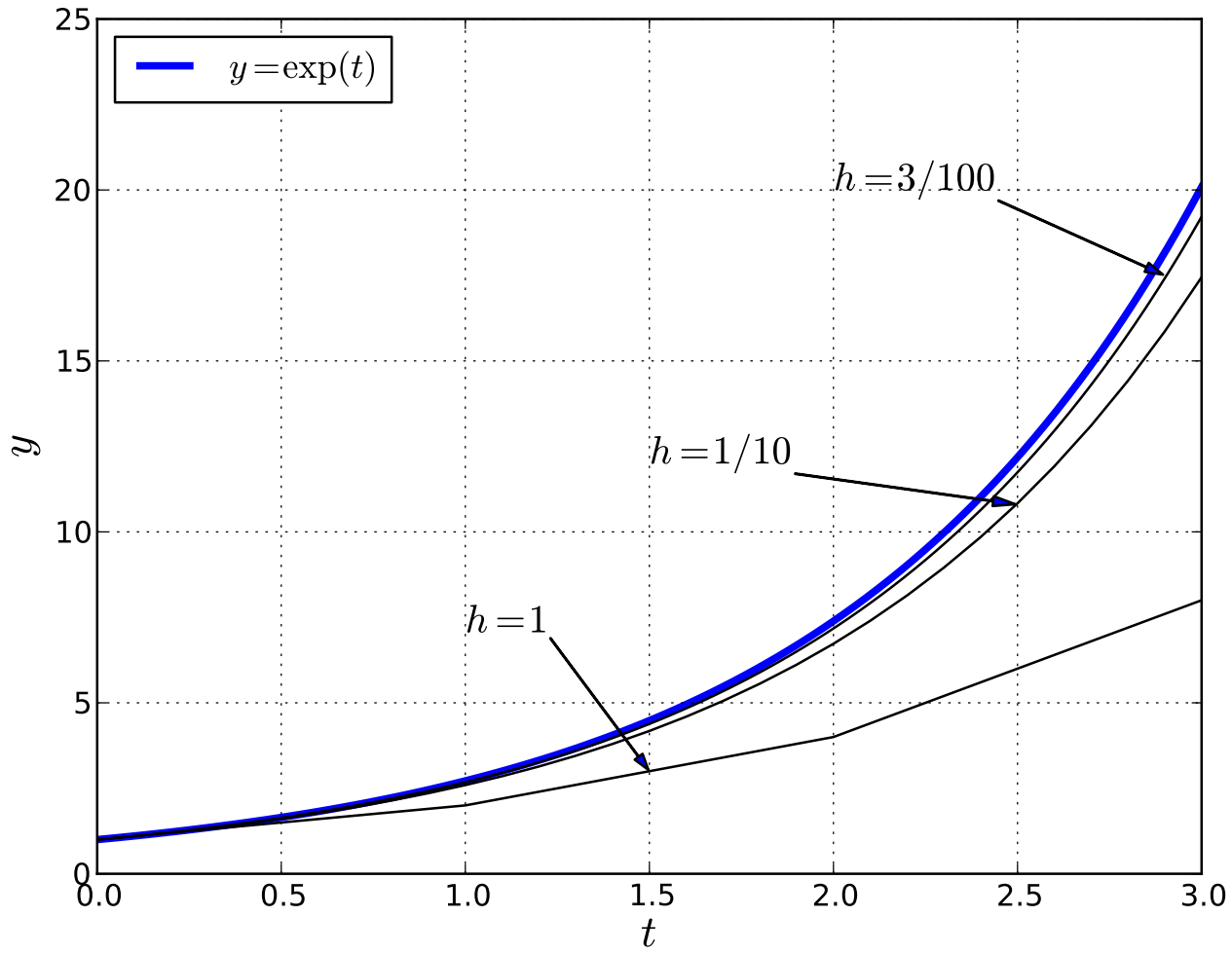
On cherche la solution (maximale) sur $[0, 3]$ de

$$y' = y$$

avec condition initiale $y(0) = 1$.

(On sait que c'est $t \mapsto e^t$).

Numériquement, on regarde successivement avec $n = 3$, $n = 30$ et $n = 100$.


$y = \exp(t)$
$h = 3/100$
$h = 1/10$
$h = 1$
y
t
0
5
10
15
20
25
0.0
0.5
1.0
1.5
2.0
2.5
3.0

9 Quelques remarques

On peut conjecturer que

1. quand le pas diminue, l'approximation s'améliore ;
2. la méthode d'Euler ne corrige pas les erreurs d'approximation mais au contraire les propage en les augmentant.

En fait :

1. Le premier point est vrai mathématiquement.
2. Le second dépend des équations considérées.

Informatiquement :

1. Si le pas est grand, l'approximation est mauvaise ;
2. Si le pas est petit, elle est longue à calculer ;
3. S'il est vraiment trop petit, les erreurs d'arrondis causent en plus d'autres problèmes.

10 Mise en œuvre

```
def euler(F, a, b, y0, h):
    y = y0
    t = a
    les_y = [y0] # la liste des valeurs renvoyées
    les_t = [a]
    while t+h <= b:
        y += h * F(t, y)
        les_y.append(y)
        t += h
        les_t.append(t)
    return les_t, les_y
```

11 Exemple

Problème de cinétique chimique : on s'intéresse à une réaction chimique faisant disparaître un réactif A d'une solution.

On suppose que la vitesse de disparition de A est proportionnelle à sa concentration (réaction d'ordre 1) :

$$\frac{d[A]}{dt} = -\alpha[A]$$

où α est une constante s'exprimant en s^{-1} (la valeur $\frac{\ln 2}{\alpha}$ est un temps appelé temps de demi-réaction).

L'équation est bien de la forme

$$[A]' = F(t, [A]) \qquad \text{où } F : (t, y) \mapsto -\alpha y$$

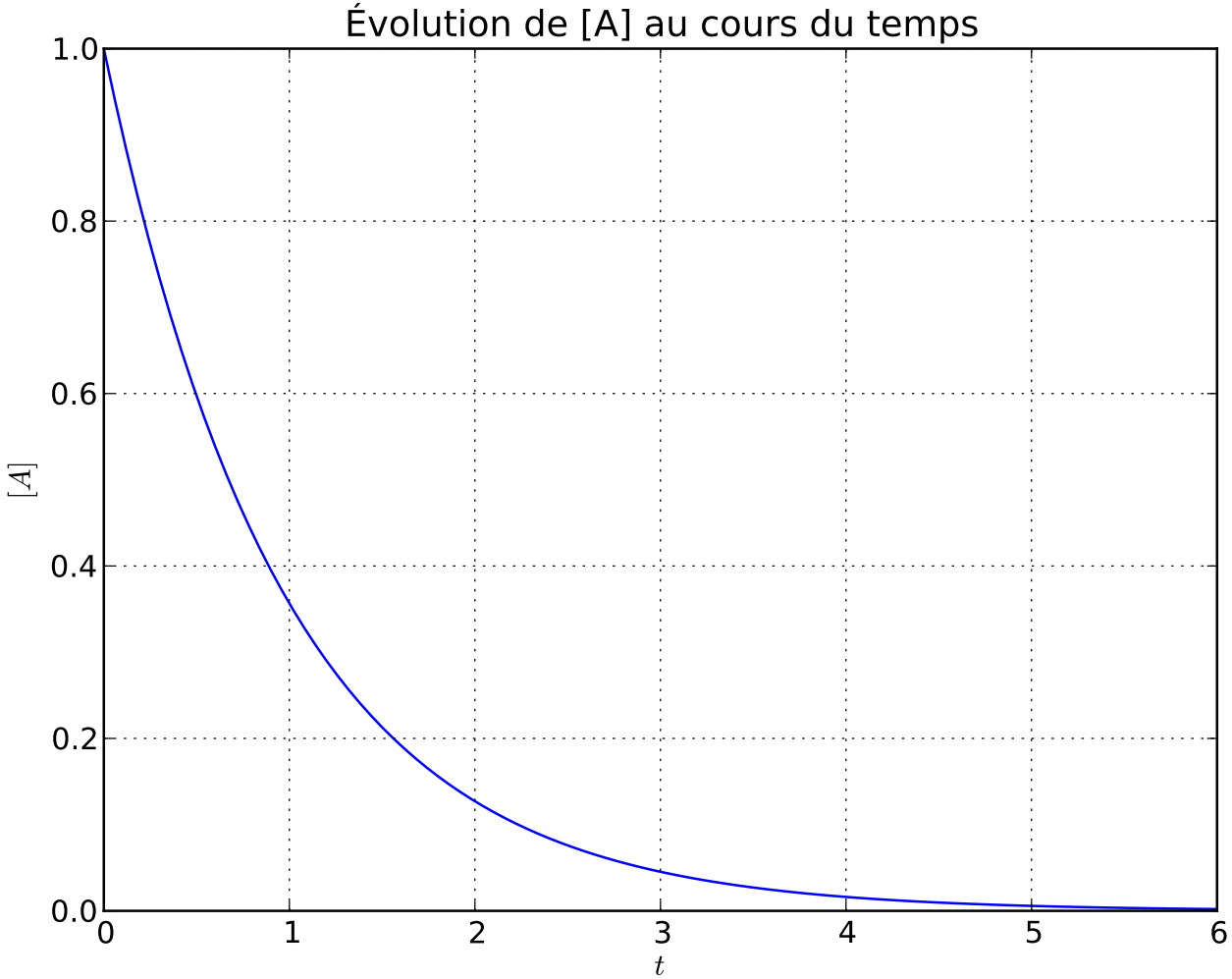
On suppose $\alpha = 1\text{s}^{-1}$ et au temps $t = 0$, $[A] = 1\text{mol/l}$. On veut regarder l'évolution jusqu'à $t = 6\text{s}$.

```
alpha = 1 # s ** -1
A0 = 1 # mol/l
t0 = 0
t1 = 6 # s
n = 100
h = float(t1 - t0) / n
les_t, les_y = euler(lambda t, y: -alpha * y, t0, t1,
```

12 Représentation de graphes

```
import matplotlib.pyplot as pl


pl.plot(les_t, les_y)
pl.grid()
pl.title(u'Évolution de [A] au cours du temps')
pl.xlabel('$t$')
pl.ylabel('$[A]$')
pl.savefig('graphe-A.pdf') # pl.show() afficher
pl.clf() # pour effacer le graphe
```


Évolution de [A] au cours du temps
1.0
0.8
0.6
0.4
0.2
0.0
[A]
0
1
2
3
4
5
6
t

13 Exemple en dimension 3

Problème de cinétique chimique : avec trois réactifs, A, B et C. A se transforme en B qui se transforme en C (réactions toutes d'ordre 1).

$$\frac{d[A]}{dt} = -\alpha[A]$$
$$\frac{d[B]}{dt} = \alpha[A] - \beta[B]$$
$$\frac{d[C]}{dt} = \beta[B]$$

On supposera $\alpha = 1\text{s}^{-1}$, $\beta = 10\text{s}^{-1}$ et au temps $t = 0$, $[A] = 1\text{mol/l}$, $[B] = [C] = 0$. On veut regarder l'évolution jusqu'à $t = 6\text{s}$.

On utilise **presque** la même fonction `euler` que précédemment

Mais avec une fonction F qui rendra des éléments de $\mathbb{R}^3$ et non plus de $\mathbb{R}$.

Problème technique : l'addition sur les triplets ne donne pas ce qu'on veut :

```
>>> (1, 2, 3) + (4, 5, 6)
(1, 2, 3, 4, 5, 6)
```

Heureusement : la bibliothèque de calcul numérique `numpy` fournit un type `array` de tableau sur lequel les opérations usuelles sont appliquées terme à terme :

```
>>> from numpy import array
>>> array([1, 2, 3]) * array([4, 5, 6])
array([4, 10, 18])
```

Point à ne pas rater : les tableaux sont modifiables (pas les entiers)

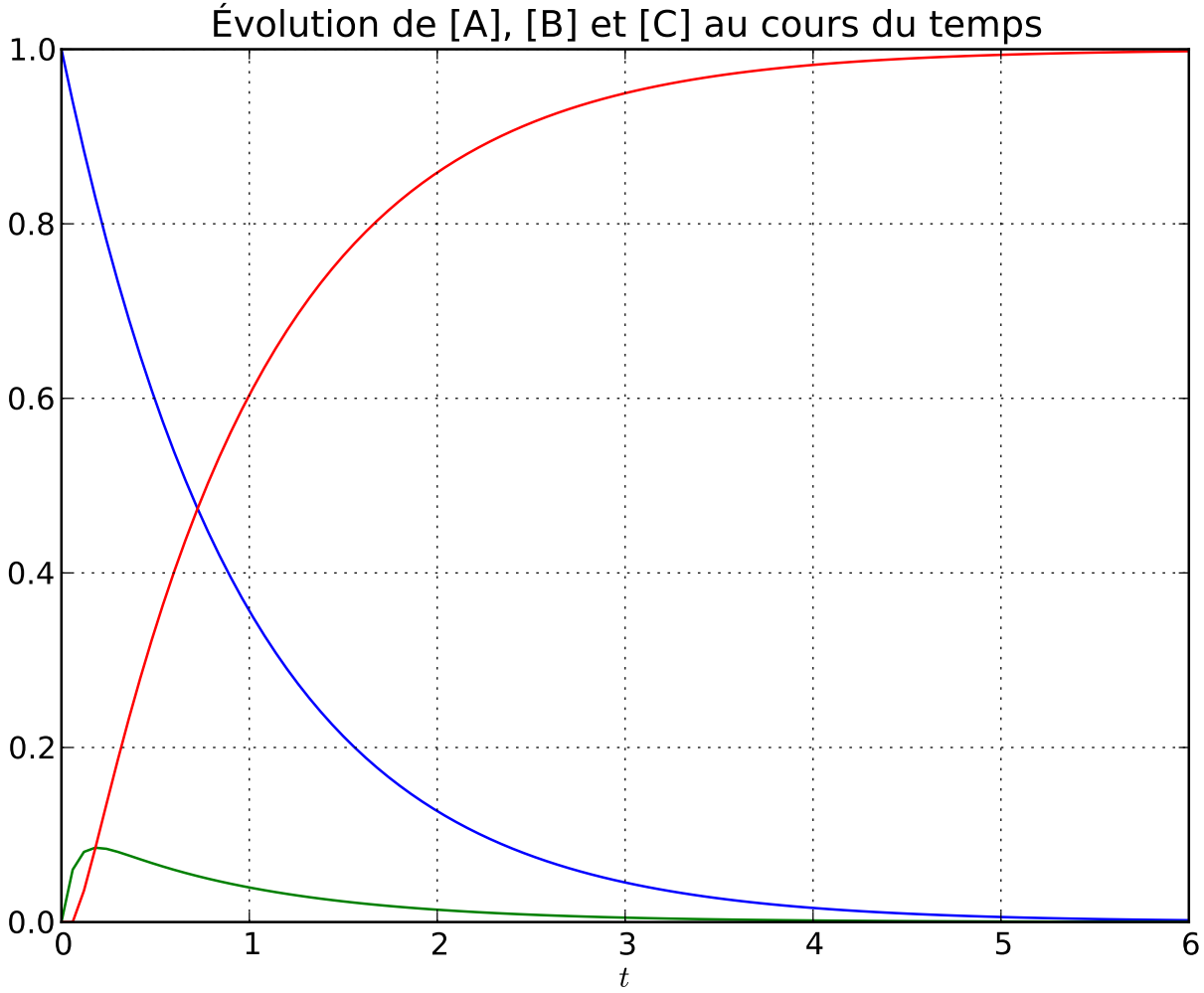
Conséquence : différence entre

```
x = array([1, 2, 3])
y = x
y += array([4, 5, 6])
```

et

```
x = array([1, 2, 3])
y = x
y = y + array([4, 5, 6])
```

```
def euler_vectoriel(F, a, b, y0, h):
    y = y0
    t = a
    les_y = [y0] # la liste des valeurs renvoyées
    les_t = [a]
    while t+h <= b:
        y = y + h * F(t, y) # surtout pas += !
        les_y.append(y)
        t += h
        les_t.append(t)
    return les_t, les_y
```


Évolution de [A], [B] et [C] au cours du temps
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
6
t

14 Utilisation de scipy

Nous ne sommes pas les seuls à vouloir résoudre numériquement des équations différentielles, donc il doit déjà exister des implantations pour ça.

Intérêt de réutiliser une implantation existante :

1. gain de temps (pas à la reprogrammer) ;
2. bugs connus (trouvés par d'autres) ;
3. problèmes de performance connus (remarqués par d'autres).

De plus on peut espérer :

1. que les bugs ont été corrigés ;
2. que les performances ont été optimisées.

Dans le cas du logiciel libre :

1. c'est souvent le cas ;
2. sinon, on peut réparer soi-même (ou faire réparer).

Bibliothèque `scipy` : propose une foultitude de méthodes de calcul numérique.

`odeint` :

- fonction de `scipy.integrate` ;
- résout numériquement des EDO ;
- utilise une méthode plus raffinée que celle d'Euler ;
- choisit elle-même le pas à utiliser.

Utilisation :

```
from scipy.integrate import odeint
les_y = odeint(F, y0, les_t)
```

Attention :

1. Résout $y' = F(y,t)$ et non $y' = F(t,y)$.
2. On ne donne pas les extrémités de l'intervalle mais les points où on veut des valeurs (`y0` : condition initiale en `les_t`[0])

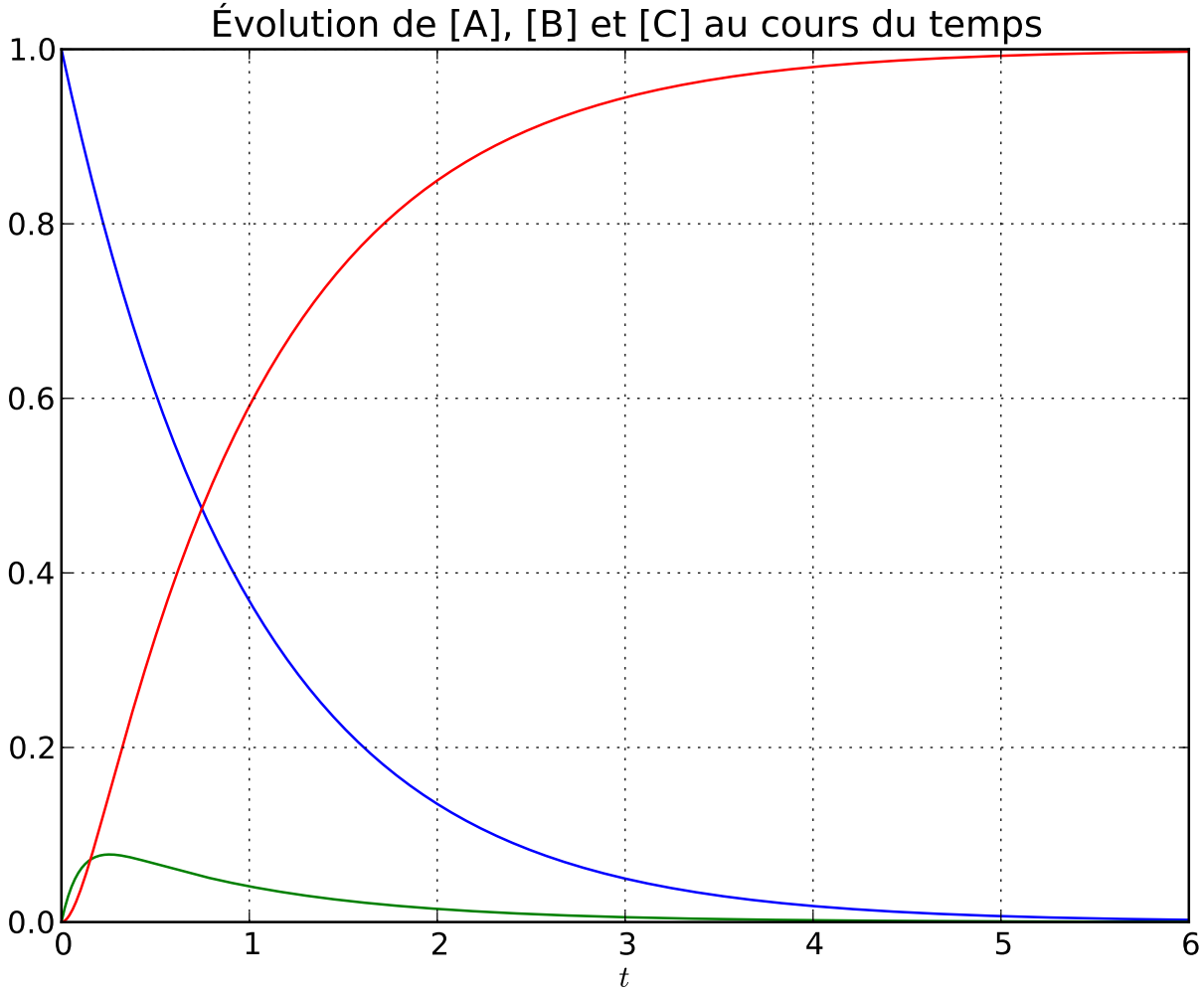
Exemple :

```
from numpy import array, linspace
from scipy.integrate import odeint
alpha = 1 # s ** -1
beta = 10 # s ** -1
A0 = 1 # mol / l
B0 = 0
C0 = 0
t0 = 0
t1 = 6 # s
n = 1000
h = float(t1 - t0) / n
```

```
def F(y, t):
    A, B, C = y
    return array([-alpha*A, alpha*A - beta*B, beta*B])

les_t = linspace(0, 6, n + 1) # nb points

les_y = odeint(F, array([A0, B0, C0]), les_t)
```


Évolution de [A], [B] et [C] au cours du temps
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
6
t

15 Résumé

1. on a vu le principe de la méthode d'Euler ;
2. il s'applique aux EDO d'ordre 1 ;
3. la fonction cherchée peut être à valeurs dans $\mathbb{R}^n$ ou $\mathbb{C}^n$ pour tout $n \in \mathbb{N}^*$;
4. `scipy.integrate` propose une fonction `odeint` qui implante des méthodes un peu plus raffinées.

Reste à voir :

1. Quelles garanties/risques on a dans l'utilisation de méthodes numériques ;
2. Quelles méthodes on peut envisager pour faire mieux que la méthode d'Euler ;
3. Comment gérer des EDO d'ordre 2, 3, ...