Méthode d'Euler

Révision du TP 1

Lycée Pierre Corneille – MP

2016-2017

Constantes

La constante cinétique k et le temps de demi-vie $t_{1/2}$ sont reliés par : $k \times t_{1/2} = \ell\mathrm{n}\, 2$.

```
import numpy as np
T1, T2 = 66, 6
k1, k2 = np.log(2)/T1, np.log(2)/T2
```

Méthode d'Euler

Les données sont :

- la fonction f qui définit le système différentiel du premier ordre,
- la condition initiale constituée de l'instant initial t_0 et de la position initiale,
- la durée T de l'étude,
- le pas dt de la discrétisation de l'intervalle $[0, T[$.

Toutes les données vectorielles sont mises au format `numpy.array` afin de simplifier les calculs (additions terme à terme, multiplication par un scalaire).

Code de la méthode d'Euler

```
def Euler(f, x0, t0, T, pas):
  t = np.arange(t0, T, pas)
  X = []
  # Position initiale
  x = np.array(x0)
  for instant in t:
    X.append(x)
    # Schéma explicite pour x' = f(x,t)
    x = x + pas*np.array(f(x, instant))
  return t, np.array(X)
```

Problème de Cauchy

Le système différentiel est défini par une fonction vectorielle $f : \mathbb{R}^3 \to \mathbb{R}^3$.

```
def f(x, t):
  return [-k1*x[0], k1*x[0]-k2*x[1], k2*x[1]]
```

Condition initiale : au début de la réaction, le réactif A est seul, donc les fractions molaires initiales sont $(100\%, 0, 0)$.

```
x0 = (1, 0, 0)
```

Résolution

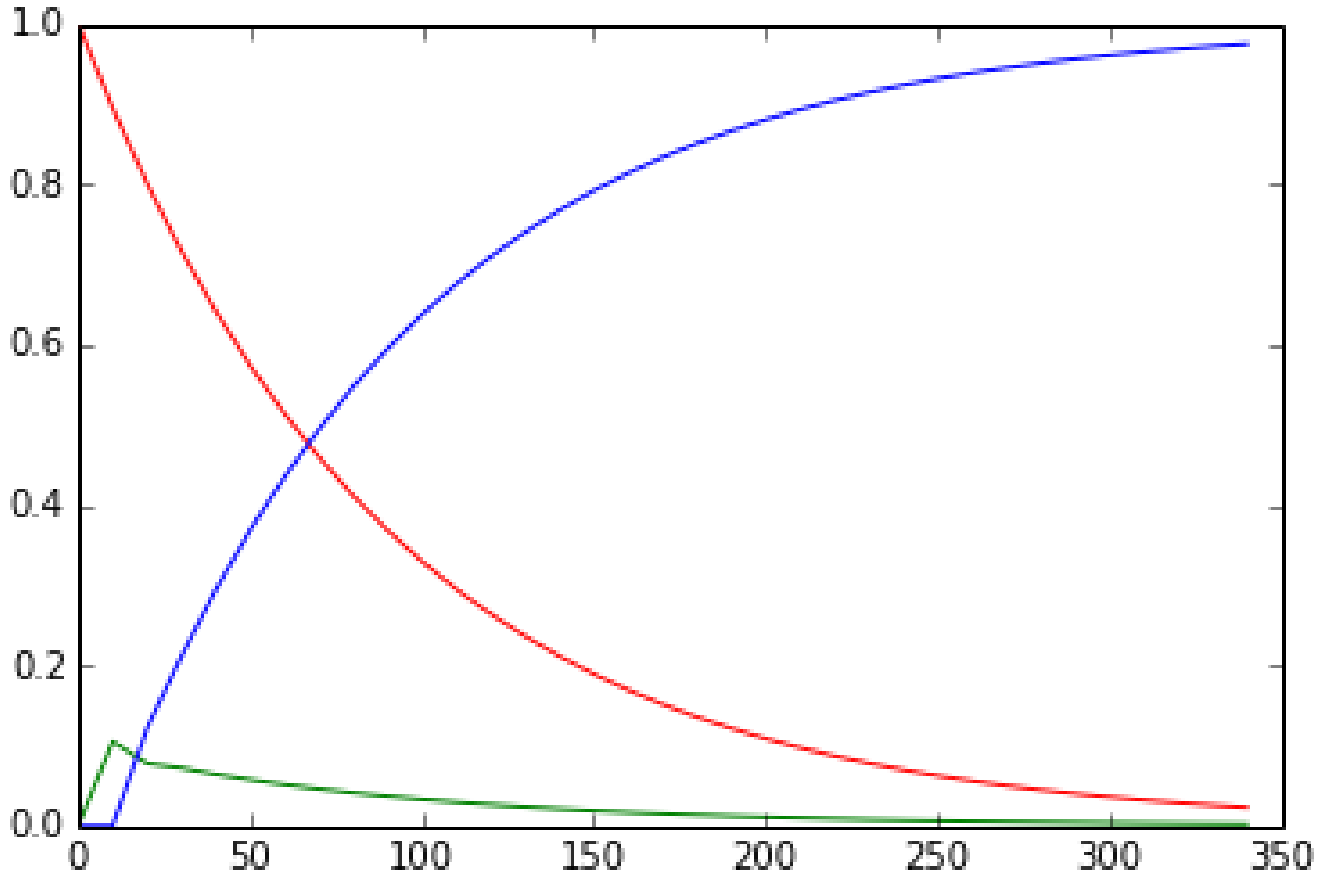
On choisit une durée d'étude assez longue devant la plus grande demi-vie et un pas de calcul assez court devant la plus petite demi-vie.

```
T = 350 # heures, soit environ 5*T1
pas1 = 0.1 # heures, soit environ T2/10
pas2 = 10 # heures


t1, V1 = Euler(f, x0, 0.0, T, pas1)
t2, V2 = Euler(f, x0, 0.0, T, pas2)

import matplotlib.pyplot as plt
plt.plot(t2, V2[:,0], 'r')
plt.plot(t2, V2[:,1], 'g')
plt.plot(t2, V2[:,2], 'b')
```

Graphes

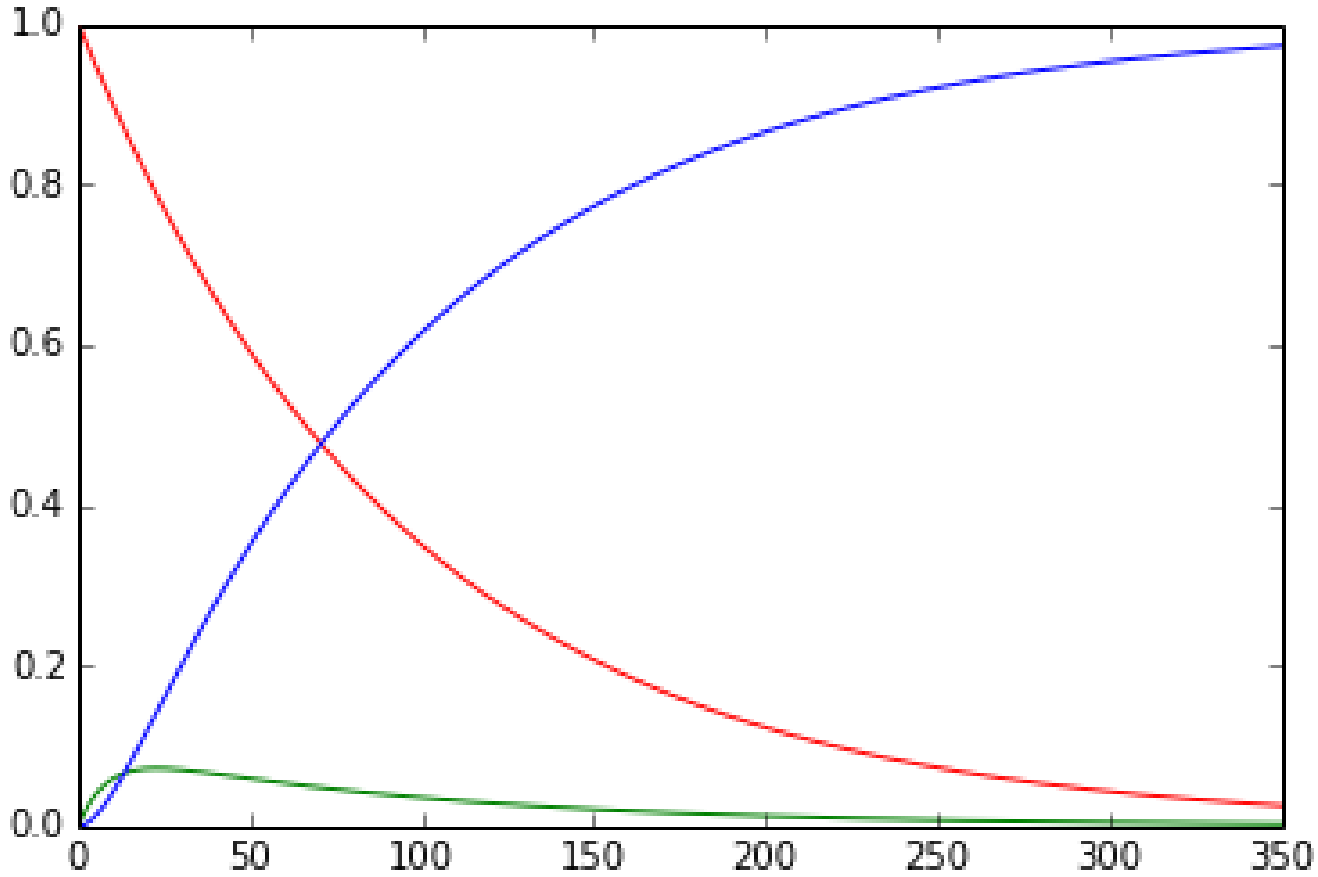


Avec les bons outils

```
from scipy.integrate import odeint as odeint
solution = odeint(f, x0, t1)

plot(t1, solution[:,0], 'r')
plot(t1, solution[:,1], 'g')
plot(t1, solution[:,2], 'b')
```

Graphes



Recherche d'un maximum

```
def max_metastable(frac_mol_Tech_meta, ech_tps):
  max_metastable = (0, 0)
  for i in range(len(frac_mol_Tech_meta)):
    valeur = frac_mol_Tech_meta[i]
    if (valeur > max_metastable[0]):
      max_metastable = (valeur, ech_tps[i])
  return max_metastable
```

Résultats

- Avec odeint

```
max_metastable(solution[:,1], t1)
```

$(0.0715\ldots, 22.8)$

- Avec le schéma d'Euler (petit pas)

```
max_metastable(V1[:,1], t1)
```

$(0.0716\ldots, 22.7)$

- Avec le schéma d'Euler (grand pas)

```
max_metastable(V2[:,1], t2)
```

$(0.1050\ldots, 10.0))$

Constantes

L'intervalle de temps est imposé : $[t_0, t_f] = [0, 500]$ (en secondes) et sa discrétisation aussi : 1000 sous-intervalles.

```
n = 1000
T = 500
pas = T/n
t = np.linspace(0, T, n+1)
```

Constantes

Constantes de la réaction

- Quantité initiale de constituant B

```
nB0 = 0.25 # (en moles)
```

- Volume initial

```
V0 = 5 # (en litres)
```

- Débit volumique du constituant A apporté

```
Q = 0.05 # (en litres par seconde)
```

- Concentration du constituant A apporté

```
CA0 = 0.025 # (en moles par litre)
```

- Constante cinétique

```
k = 2.2 # (en litres par mole et par seconde)
```

Évolution du volume occupé dans le réacteur

```
V = V0 + Q*t
```

Méthode d'Euler

On adapte le schéma d'Euler au problème étudié.
Le système différentiel n'est ni linéaire, ni autonome

$$\begin{cases} \frac{\mathrm{d}n_A(t)}{\mathrm{d}t} = QC_{A,0} - k \cdot \frac{n_A(t) \times n_B(t)}{V(t)} \\ \frac{\mathrm{d}n_B(t)}{\mathrm{d}t} = k \cdot \frac{n_A(t) \times n_B(t)}{V(t)} \end{cases}$$

mais le temps n'apparaît que par l'évolution (*connue*) du volume V.
On traduit donc le système par une fonction de la "position" $x = (n_A, n_B)$ et du volume V.

Code de la méthode d'Euler

```
def f(x, v):
  return [Q*CA0-k*x[0]*x[1]/v, -k*x[0]*x[1]/v]

def Euler(f, x0, pas, V):
  x = np.array(x0)
  X = [x]
  for v in V[1:]:
    x = x + pas*np.array(f(x,v))
    X.append(x)
  return np.array(X)
```

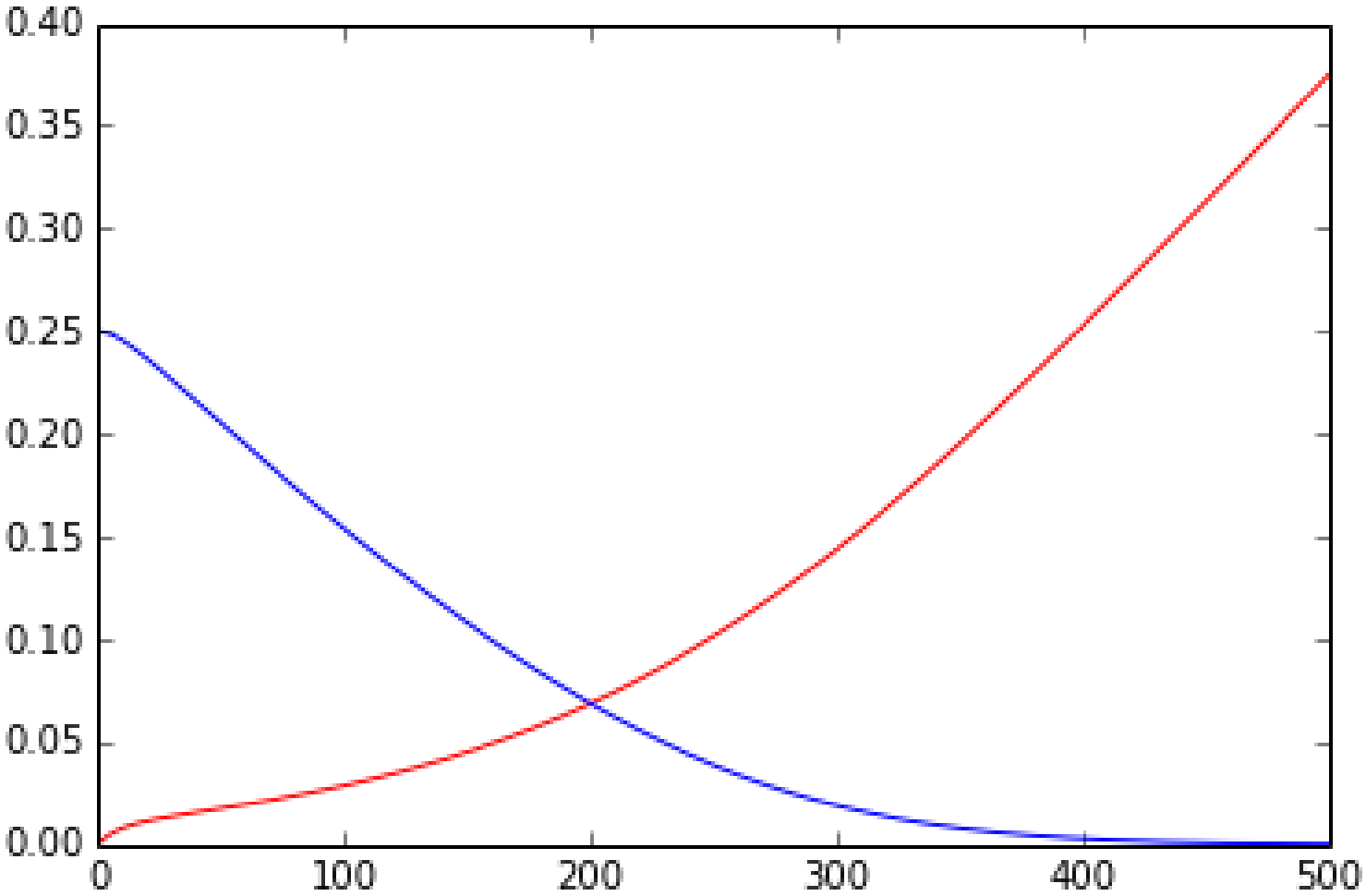
Résolution

```
solution = Euler(f, [0, nB0], pas, V)

nA, nB = solution[:,0], solution[:,1]

plot(t, nA, 'r')
plot(t, nB, 'b')
```

Graphes



Variante avec odeint

Système différentiel *autonome*, trois inconnues fonctions du temps : nA, nB et V.

$$\begin{cases} \frac{\mathrm{d}n_A(t)}{\mathrm{d}t} = QC_{A,0} - k \cdot \frac{n_A(t) \times n_B(t)}{V(t)} \\ \frac{\mathrm{d}n_B(t)}{\mathrm{d}t} = k \cdot \frac{n_A(t) \times n_B(t)}{V(t)} \\ \frac{\mathrm{d}V(t)}{\mathrm{d}t} = Q \end{cases}$$

Condition initiale : $\big(nA(0), nB(0), V(0)\big)$.

On résout un système différentiel de la forme

$$x'(t) = g\big(x(t), t\big)$$

d'inconnue $x : \mathbb{R}_+ \to \mathbb{R}^3$.

```
def g(x, t):
  return [Q*CA0-k*x[0]*x[1]/x[2], -k*x[0]*x[1]/x[2], Q]
```

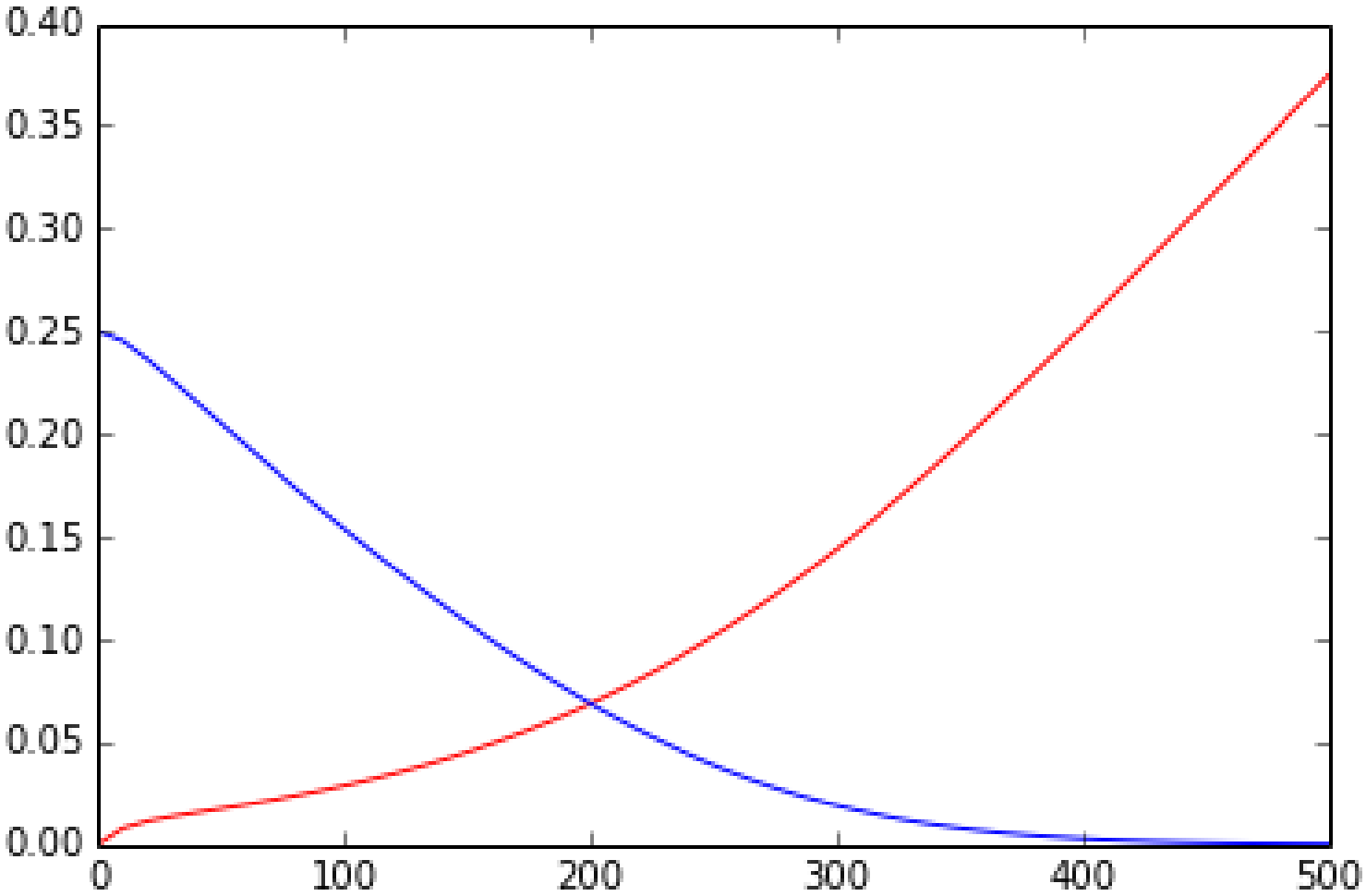
Condition initiale et résolution

Avec odeint, on peut se contenter d'une discrétisation assez grossière.

```
cond_init = [0.0, nB0, V0]
t = np.linspace(0, T, 50)

from scipy.integrate import odeint as odeint
solution = odeint(g, cond_init, t)
plot(t, solution[:,0], 'r')
plot(t, solution[:,1], 'b')
```

Graphes



Ne faites pas comme moi...

Ne faites pas comme moi...

Les figures doivent être légendées !