Intégration numérique d'équations différentielles

Alessandro Torcini et Andreas Honecker

LPTM

Université de Cergy-Pontoise

Stabilité

On veut assurer que la solution numérique est stable dans le sens que l'erreur ne diverge pas

Appelons donc

1. y_i la solution exacte au point x_i
2. $\tilde{y}_i$ la solution numérique
3. l'erreur $e_i := \tilde{y}_i - y_i \longrightarrow y_i = \tilde{y}_i + e_i$

La méthode numérique est une application T que fait un pas d'intégration Δt

$$\tilde{y}_{i+1} = T(\tilde{y}_i)\,,$$

avec la définition de l'erreur on obtient

$$y_{i+1} + e_{i+1} = T(y_i + e_i) \approx T(y_i) + T'(y_i)\, e_i$$

en supposant que l'erreur soit petite nous pouvons utiliser une approximation linéaire.
Par conséquent, comme $y_{i+1} \approx T(y_i)$, on obtient

$$e_{i+1} \approx T'(y_i)\, e_i\,.$$

Stabilité

Afin de ne pas avoir d'erreur divergente, nous demandons maintenant que $|e_{i+1}| \leq |e_i|$ et trouvons la condition suivante pour la stabilité de la méthode numérique :

$$|T'(y_i)| \leq 1\,.$$

Exemple

Pour illustrer cette idée générale je reviens à l'équation pour la croissance exponentielle

$$\frac{\mathrm{d}\,y}{\mathrm{d}t} = \lambda\, y(t) \longrightarrow y(t) = y(0)\mathrm{e}^{\lambda t}$$

Ici la méthode d'Euler correspond à

$$T(y) = y + \lambda\,\Delta t\, y \longrightarrow T'(y) = 1 + \lambda\,\Delta t\,.$$

La condition de stabilité nécessite alors que $|1 + \lambda\,\Delta t| \leq 1$.

1. Pour $\lambda > 0$, cette condition n'est jamais satisfaite.
2. Pour $\lambda < 0$, la méthode d'Euler est stable seulement pour $\Delta t \leq \frac{2}{|\lambda|}$.

Stabilité

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.legend_handler import HandlerLine2D

Tmax = 4
def Fcroiss(y,x): # la fonction F dans y'=F(y,x) -- lambda=-10
  return -10*y

def euler(F, t0, y0, deltaT, Tfin, tv, yv):  # la methode d'Euler
  t = t0
  y = y0
  tv.append(t)
  yv.append(y)
  while t<=Tfin+1e-8:
    y += deltaT*F(y,t)
    t += deltaT
    tv.append(t)
    yv.append(y)
```

Stabilité

```
t1v, y1v = [], []
euler(Fcroiss, 0, 1, 0.21, Tmax, t1v, y1v)
t2v, y2v = [], []
euler(Fcroiss, 0, 1, 0.09, Tmax, t2v, y2v)

exact = []
for t in t2v:
 exact.append(np.exp(-10*t))       # solution exacte

plt.scatter(t1v, y1v, marker='o', color='red', label="Delta t=0.21")
plt.plot(t1v, y1v,  color='red')
plt.scatter(t2v, y2v, marker='s', color='blue', label="Delta t=0.09")
plt.plot(t2v, exact, color='black', label="exact  lambda=-10 ")
plt.xlabel("t")
plt.ylabel("y")
plt.xlim(-0.01,Tmax+0.01)
plt.legend(loc=2)                  # afficher les legendes a gauche
plt.show()                         # montrer le graphe
```

Stabilité

Imaginons que la dynamique d'une densité de population n suit l'équation différentielle suivante :

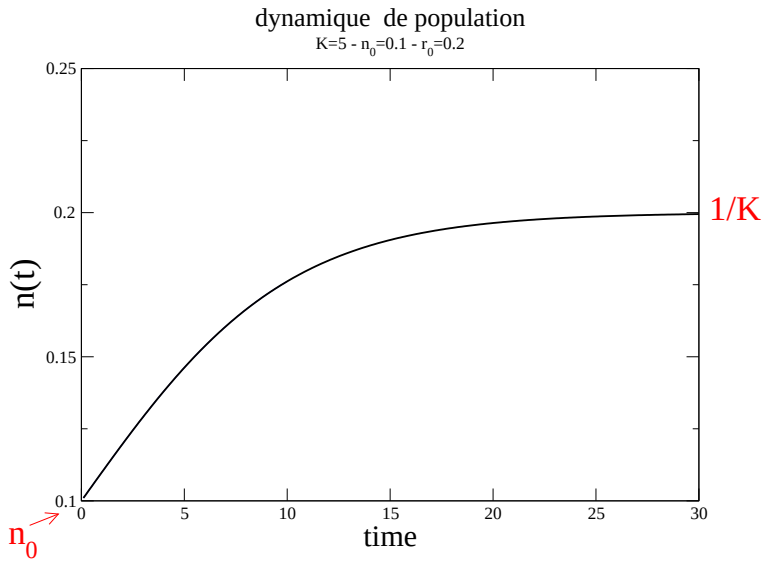
$$\frac{\mathrm{d}\, n}{\mathrm{d}t} = r_0 \left(1 - K\, n(t)\right) n(t)$$

avec des paramètres r_0, K.

On peut vérifier que la solution exacte pour $n(0) = n_0$ est

$$n(t) = \frac{n_0\, \mathrm{e}^{r_0\, t}}{1 + K\, n_0 \left(\mathrm{e}^{r_0\, t} - 1\right)}\,.$$

En particulier, pour $n_0 \neq 0$ la solution converge pour des temps grands à $\lim_{t\to\infty} n(t) = 1/K$



Stabilité

De l'autre côté la méthode d'Euler donne pour l'équation pour la dynamique de population

$$n_{i+1} = n_i + \Delta t\, r_0\, (1 - K\, n_i)\, n_i\, .$$

Cette équation est identique à la suite logistique $x_{i+1} = 4x_i(1 - x_i)$ si on définit les paramètres comme

$$K = 1 \qquad 4\, r = 1 + \Delta t\, r_0\, .$$

Maintenant, on peut vérifier que la condition de stabilité est

$$|1 + \Delta t r_0(1 - 2n_i)| < 1$$

Pour des temps suffisamment longs $n_i \to 1/K = 1$, donce la condition est équivalente à

$$4\, r \leq 3$$

soit le regime de la suite logistique avec un seul point fixe attractif
Si on prend un Δt en peu plus grand, la solution numérique de l'équation commence à osciller et après elle devient même chaotique contrairement à la solution exacte qui est toujours très régulière

Les méthodes de Runge-Kutta

Les méthodes de Runge-Kutta constituent une approche systématique pour augmenter l'ordre de l'approximation en utilisant le principe de l'itération, c'est-à-dire qu'une première estimation de la solution est utilisée pour calculer une seconde estimation, plus précise, etc.
Habituellement, on devrait intégrer l'équation suivant

$$\frac{dy}{dt} = F(y(t), t)$$

En intégrant l'équation différentielle entre t_n et $t_{n+1} = t_n + h$ on a la relation

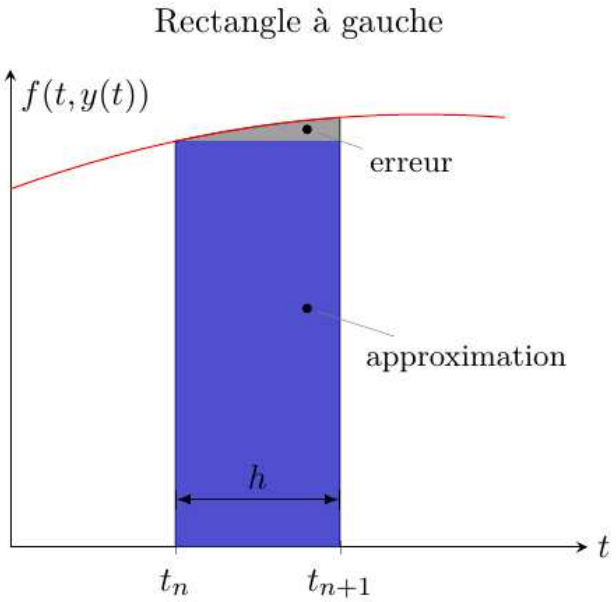
$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} F(y(t), t)dt$$

oú $y_n = y(t_n)$ et $y_{n+1} = y(t_{n+1})$.
L'idée consiste à approcher cette intégrale de façon plus précise que ne le fait la méthode d'Euler. Mais avant de voir comment, revenons sur la méthode d'Euler.

Retour sur Euler

L'intégrale peut s'approcher par la méthode du rectangle à gauche :

$$\int_{t_n}^{t_{n+1}} F(y(t),t)dt \approx h \times F(y(t_n),t_n))$$



D'oú le schéma itératif suivant

$$y_{n+1} = y_n + h \times F(y_n,t_n))$$

oú h est le pas d integration

L'erreur produite correspond à l'aire grisée de forme quasi-triangulaire et de côtés h et ph où p est la pente de F à l'instant t_n . L'erreur vaut donc à peu près

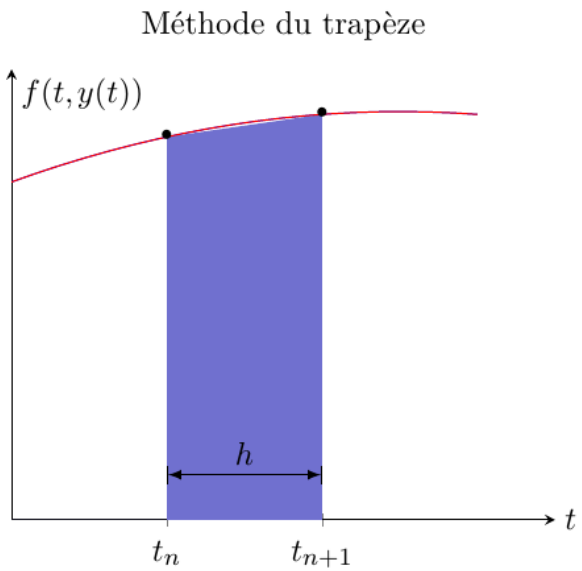
$$e_{EU} \simeq \frac{1}{2}ph^2$$

Après N itérations, on commet une erreur globale de l'ordre de $N\frac{1}{2}ph^2 = \frac{1}{2}Tph$ où T est la durée totale. Pour une durée donnée, l'erreur globale augmente linéairement avec le pas h : on dit que la méthode d'Euler est d'ordre un

Runge-Kutta de ordre 2

On voit immédiatement que l'on peut améliorer l'estimation de l'intégrale en calculant l'aire d'un trapèze au lieu de celui d'un rectangle. La méthode du trapèze consiste en l'approximation suivante :

$$\int_a^n f(x)dx \approx \frac{b-a}{2}[f(a)+f(b)]$$



Donc

$$\int_{t_n}^{t_{n+1}} F(y(t),t)dt \approx \frac{h}{2}\times[F(y(t_n),t_n))+F(y(t_{n+1}),t_{n+1}))]$$

On utilise la méthode d'Euler afin estimer la valeur y_{n+1} qui intervient dans $f(y(t_{n+1}),t_{n+1})$. On obtient le schéma itératif suivant :

$$y_{n+1} = y_n + \frac{h}{2}(k_1+k_2) \qquad \text{avec} \qquad \begin{cases} k_1 &= F(y_n,t_n) \\ k_2 &= F(y_n+hk_1,t_n+h) \end{cases}$$

Modèle pour le RK2

```
# un pas avec la methode Runge-Kutta d'ordre deux

def pasRK2(F, x, y, deltaX):
 k1 = deltaX*F(y,x)
 k2 = deltaX*F(y+0.5*k1, x+0.5*deltaX)
 return y+(k1+ k2)/2.0

# et l'integrateur complet avec la methode Runge-Kutta d'ordre deux

def rk2(F, t0, y0, deltaT, Tfin, tv, yv):
  t = t0
  y = y0
  tv.append(t)
  yv.append(y)
  while t<=Tfin+1e-8:
    y = pasRK2(F, t, y, deltaT) # un pas de la methode
    t += deltaT                 # aussi actualisier t
    tv.append(t)
    yv.append(y)
```

Runge-Kutta de ordre 4

La méthode de Runge-Kutta d'ordre 4 est une étape supplémentaire dans le raffinement du calcul de l'intégrale. Au lieu d'utiliser la méthode des trapèzes, on utilise la méthode de Simpson.

$$\int_a^n f(x)dx \approx \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right]$$

Donc

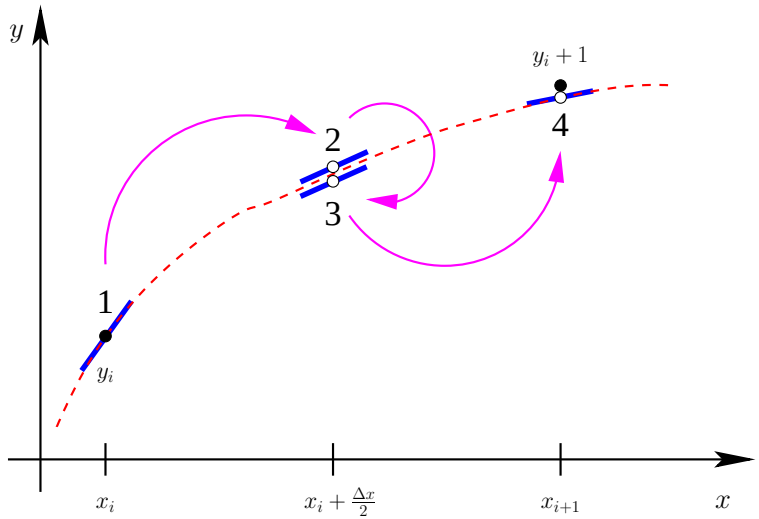
$$\int_{t_n}^{t_{n+1}} F(y(t),t)dt \approx \frac{h}{6} \times \left[F(y(t_n),t_n)) + 4F(y(t_{n+1/2}),t_{n+1/2}) + F(y(t_{n+1}),t_{n+1}))\right]$$

On obtient le schéma itératif suivant :

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \qquad \text{avec} \qquad \begin{cases} k_1 &= F(y_n, t_n) \\ k_2 &= F(y_n + \frac{h}{2}k_1, t_n + h/2) \\ k_3 &= F(y_n + \frac{h}{2}k_2, t_n + h/2) \\ k_4 &= F(y_n + hk_3, t_n + h) \end{cases}$$

On peut démontrer que la méthode RK4 est une méthode d'ordre 4, ce qui signifie que l'erreur commise à chaque étape est de l'ordre de $\mathcal{O}(h^5)$.

Runge-Kutta de ordre 4



Pour comprendre la procédure, nous regardons la figure :

1. On prend la pente au début de l'intervalle k_1 pour faire un pas avec la méthode d'Euler jusqu'au milieu de l'intervalle et on obtient une première approximation k_2 de la pente au milieu.
2. Après on répète le pas, mais maintenant avec la pente k_2 afin d'obtenir une approximation meilleure k_3 de la pente au milieu.
3. Après on utilise k_3 pour aller à la fin de l'intervalle et on utilise l'approximation k_4 pour la pente à la fin d'intervalle pour faire le pas final.

Modèle pour le RK4

```
# un pas avec la methode Runge-Kutta d'ordre quatre

def pasRK4(F, x, y, deltaX):
 k1 = deltaX*F(y,          x)
 k2 = deltaX*F(y+0.5*k1,  x+0.5*deltaX)
 k3 = deltaX*F(y+0.5*k2,  x+0.5*deltaX)
 k4 = deltaX*F(y+k3,      x+deltaX)
 return y+(k1+2*k2+2*k3+k4)/6.0

# et l'integrateur complet avec la methode Runge-Kutta d'ordre quatre
def rk4(F, t0, y0, deltaT, Tfin, tv, yv):
  t = t0
  y = y0
  tv.append(t)
  yv.append(y)
  while t<=Tfin+1e-8:
    y = pasRK4(F, t, y, deltaT) # un pas de la methode
    t += deltaT                 # aussi actualisier t
    tv.append(t)
    yv.append(y)
```