TD n°4 : la méthode d'Euler

Nous allons voir comment la méthode d'Euler (ainsi que ses diverses variantes) est généralement introduite et traitée dans les sujets de concours.

On note $t_{\min}$ et $t_{\max}$ deux nombres réels tels que $t_{\min} < t_{\max}$ et I l'intervalle $[\,t_{\min}\,;t_{\max}\,]$.

1) Équations différentielles du premier ordre

Soient F une fonction continue sur $\mathrm{I}\times\mathbb{R}$ et y la solution sur I du problème de Cauchy

$$\text{(S)}\quad \begin{cases} y'(t) = \mathrm{F}(y(t),t) \\ y(t_{\min}) = y_0 \end{cases}$$

Fixons $n\in\mathbb{N}^*$. On pose alors $\Delta t = (t_{\max}-t_{\min})/n$ et $t_i = t_{\min}+i\Delta t$ pour tout entier $i\in[\![\,0\,;n\,]\!]$. Ainsi, $t_0 = t_{\min}$ et $t_n = t_{\max}$. On cherche ensuite des approximations des nombres $y(t_i)$.

(a) *Version 1*

Pour tout $i\in[\![\,0\,;n-1\,]\!]$, on obtient grâce à un DL_1 de y en t_i

$$y(t_{i+1}) = y(t_i+\Delta t) = y(t_i)+y'(t_i)\,\Delta t+\mathrm{o}(\Delta t)$$

soit
$$y(t_{i+1}) \simeq y(t_i)+\mathrm{F}(y(t_i),t_i)\times\Delta t$$

En notant y_i une approximation de $y(t_i)$, il en découle que

$$y(t_{i+1}) \simeq y_i+\ \mathrm{F}(y_i,t_i)\times\Delta t$$

ce qui pousse à prendre pour approximation de $y(t_{i+1})$ le nombre

$$\boxed{y_{i+1} = y_i+\Delta t\times\mathrm{F}(y_i,t_i)}$$

(b) *Version 2*

D'après la propriété fondamentale de l'intégrale, on a pour tout $i\in[\![\,0\,;n-1\,]\!]$

$$y(t_{i+1})-y(t_i) = \int_{t_i}^{t_{i+1}} y'(t)\,\mathrm{d}t = \int_{t_i}^{t_{i+1}} \mathrm{F}(y(t),t)\,\mathrm{d}t$$

En approchant l'intégrande à l'aide de sa valeur en t_i, on obtient [1]

$$y(t_{i+1})-y(t_i) \simeq (t_{i+1}-t_i)\,\mathrm{F}(y(t_i),t_i)$$

soit
$$y(t_{i+1}) \simeq y(t_i)+\Delta t\times\mathrm{F}(y(t_i),t_i)$$

On retrouve la même relation que précédemment, ce qui nous pousse ainsi à définir une suite (y_i) de valeurs approchées des $y(t_i)$ en posant

$$\boxed{y_{i+1} = y_i+\Delta t\times\mathrm{F}(y_i,t_i)}$$

Notons que y_0 est la valeur initiale donnée par le problème de Cauchy.

1. c'est la méthode des rectangles à gauche

On peut alors écrire facilement une fonction Python qui prend en argument les paramètres du problème et renvoie les listes $[t_0, t_1, \dots, t_n]$ et $[y_0, y_1, \dots, y_n]$:

```
def euler(F,tmin,tmax,y0,n):
    dt=(tmax-tmin)/n
    T=[tmin]*(n+1)
    Y=[y0]*(n+1)
    for i in range(n):
        T[i+1]=T[i]+dt
        Y[i+1]=Y[i]+dt*F(Y[i],T[i])
    return T,Y

# Pour créer T, on pouvait aussi taper
# T=[tmin+i*dt for i in range(n+1)]
```

```
def euler(F,tmin,tmax,y0,n):
    dt=(tmax-tmin)/n
    t,y = tmin,y0
    T,Y =[t],[y]
    for i in range(n):
        t=t+dt
        y=y+dt*F(y,t)
        T.append(t)
        Y.append(y)
    return T,Y
```

On vient de revoir les deux façons de faire les calculs, la première en commençant par créer des listes de bonne longueur et la seconde en remplissant au fur et à mesure des listes.

2) Équations différentielles du second ordre

Soient F une fonction continue sur $\mathrm{I} \times \mathbb{R}^2$ et y la solution sur I du problème de Cauchy

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

avec $y(t_{\min}) = y_0$ et $y'(t_{\min}) = z_0$. En posant $z(t) = y'(t)$, on obtient le système différentiel

$$(\mathrm{S}) \quad \begin{cases} y'(t) = z(t) \\ z'(t) = \mathrm{F}(t, y(t), z(t)) \end{cases}$$

Fixons $n \in \mathbb{N}^*$. On pose alors $\Delta t = (t_{\max} - t_{\min})/n$ et $t_i = t_{\min} + i\Delta t$ pour tout entier $i \in [\![0\,;n]\!]$. En procédant comme précédemment, on obtient pour tout $i \in [\![0\,;n-1]\!]$

$$y(t_{i+1}) - y(t_i) = \int_{t_i}^{t_{i+1}} y'(t)\,\mathrm{d}t = \int_{t_i}^{t_{i+1}} z(t)\,\mathrm{d}t \simeq (t_{i+1} - t_i)\, z(t_i) = \Delta t \times z(t_i)$$

et $$z(t_{i+1}) - z(t_i) = \int_{t_i}^{t_{i+1}} z'(t)\,\mathrm{d}t = \int_{t_i}^{t_{i+1}} \mathrm{F}(t, y(t), z(t))\,\mathrm{d}t \simeq \Delta t \times \mathrm{F}(t_i, y(t_i), z(t_i))$$

En notant y_i une approximation de $y(t_i)$ et z_i une approximation de $z(t_i)$, il en découle que

$$(\mathrm{S}) \quad \begin{cases} y(t_{i+1}) \simeq y_i + \Delta t \times z_i \\ z(t_{i+1}) \simeq z_i + \Delta t \times \mathrm{F}(t_i, y_i, z_i) \end{cases}$$

ce qui pousse à prendre pour approximation de $y(t_{i+1})$ et $z(t_{i+1})$ les nombres

$$\boxed{\begin{cases} y_{i+1} = y_i + \Delta t \times z_i \\ z_{i+1} = z_i + \Delta t \times \mathrm{F}(t_i, y_i, z_i) \end{cases}}$$

On a ainsi défini deux suites (y_i) et (z_i) de valeurs approchées des $y(t_i)$ et $z(t_i)$, en notant que y_0 et z_0 sont les valeurs initiales fournies par le problème de Cauchy.

Voici ce que cela donne en Python :

```
def euler(F,tmin,tmax,y0,z0,n):
    dt=(tmax-tmin)/n
    T=[tmin]*(n+1)
    Y=[y0]*(n+1)
    Z=[z0]*(n+1)
    for i in range(n):
        T[i+1]=T[i]+dt
        Y[i+1]=Y[i]+dt*Z[i]
        Z[i+1]=Z[i]+dt*F(T[i],Y[i],Z[i])
    return T,Y,Z

# Pour créer T, on pouvait aussi taper
# T=[tmin+i*dt for i in range(n+1)]
```

```
def euler(F,tmin,tmax,y0,z0,n):
    dt=(tmax-tmin)/n
    t,y,z = tmin,y0,z0
    T,Y, Z =[t],[y],[z]
    for i in range(n):
        a=F(t,y,z)
        t=t+dt
        y=y+dt*z
        z=z+dt*a
        T.append(t)
        Y.append(y)
        Z.append(z)
    return T,Y,Z
```

Remarque : dans le cas d'un système conservatif, la fonction F ne dépend que de y.

3) Systèmes différentiels du premier ordre

On peut utiliser la même démarche pour calculer des suites de valeurs approchées des différentes fonctions intervenant dans le système considéré. Reprenons les notations précédentes et considérons par exemple le système de Lotke-Volterra

$$\forall\, t \in \mathrm{I} \qquad \begin{cases} x'(t) = x(t)\big(3 - 2y(t)\big) \\ y'(t) = y(t)\big(-4 + x(t)\big) \end{cases}$$

On définit deux suites de valeurs approchées des nombres $x(t_i)$ et $y(t_i)$ grâce aux relations

$$\begin{cases} x_{i+1} = x_i + \Delta t \times x_i\,(3 - 2y_i) \\ y_{i+1} = y_i + \Delta t \times y_i\,(-4 + x_i) \end{cases}$$

et aux conditions initiales $x_0 = x(t_{\min})$ et $y_0 = y(t_{\min})$.

Voici comment procéder en Python :

```
def volterra(tmin,tmax,x0,y0,n):
    dt=(tmax-tmin)/n
    T=[tmin+i*dt for i in range(n+1)]
    X=[x0]*(n+1)
    Y=[y0]*(n+1)
    for i in range(n):
        X[i+1]=X[i]+dt*X[i]*(3-2*Y[i])
        Y[i+1]=Y[i]+dt*Y[i]*(-4+X[i])
    return T,X,Y
```

```
def volterra(tmin,tmax,x0,y0,n):
    dt=(tmax-tmin)/n
    t,x,y = tmin,x0,y0
    T,X,Y =[t],[x],[y]
    for i in range(n):
        dx=dt*x*(3-2*y)
        dy=dt*y*(-4+x)
        t,x,y = t+dt,x+dx,y+dy
        T.append(t)
        X.append(x)
        Y.append(y)
    return T,X,Y
```