Résolution numérique d'équations différentielles : méthode d'Euler

Nous allons nous intéresser à la résolution d'équations différentielles de manière numérique, ce qui est particulièrement intéressant lorsque les équations ne possèdent pas de solutions analytiques. Sous couvert de vérifier certaines conditions (par exemple le théorème de Cauchy-Lipschitz), on peut montrer qu'elles peuvent admettre une unique solution. Les enjeux majeurs seront de développer des algorithmes permettant de s'approcher numériquement de la « vraie » solution et minimiser l'erreur commise.

I. Présentation de la méthode d'Euler

I.1 Principe

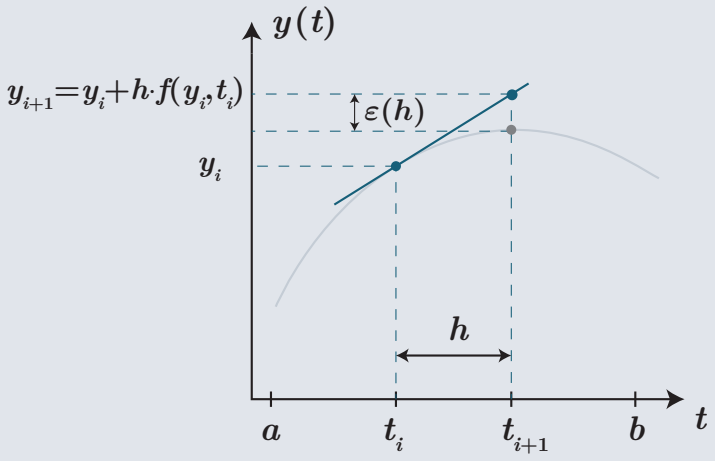
La méthode d'Euler est la première méthode inventée pour résoudre numériquement des ED en les discrétisant. On va illustrer dans un premier temps avec une équation d'ordre 1 que l'on résout pour $t \in [a;b]$:

$$\begin{cases} y' = f(y(t),t) & (1) \\ y(a) = y_0 & (2) \end{cases}$$

Par exemple pour l'équation différentielle $y' + 3ty^2(t) = 7$, on pose $f(y,t) = -3ty^2 + 7$.

Principe de la méthode d'Euler

On part d'un point d'abscisse t_i, dont on connaît l'ordonnée y_i on cherche à calculer l'ordonnée du point d'abscisse $t_{i+1} = t_i + h$ (on se décale d'un pas h). On trace la tangente à la courbe en t_i, connaissant la dérivée (donc la pente), et on approxime la position de y_{i+1} en prenant l'ordonnée de la tangente en t_{i+1} (la courbe grisée correspondant à la courbe "réelle", que l'on ne connaît pas) :



On peut exprimer explicitement l'ordonnée en t_{i+1} :

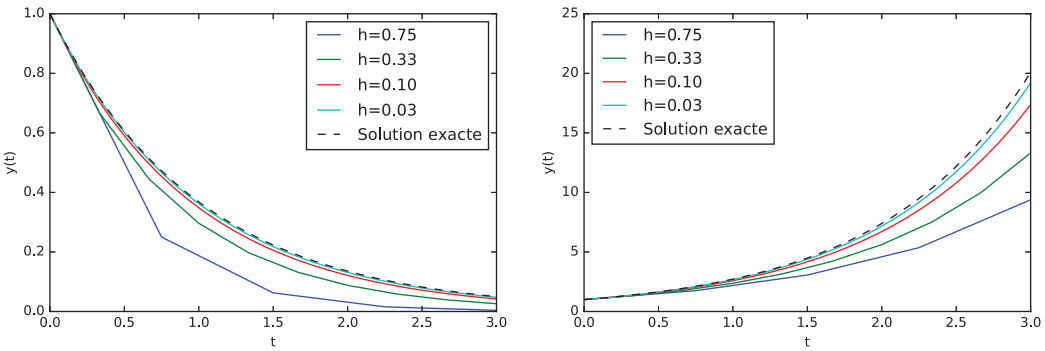
- on peut écrire un développement limité à l'ordre 1

$$y(t_i + h) = y(t_i) + hy'(t_i)$$

- Comme $y' = f(y,t)$, il vient : $y_{i+1} = y_i + hf(y_i,t_i)$.
- Par propagation on peut ensuite connaître y sur tout l'intervalle d'étude.

I.2 Influence du pas

Sur l'exemple de la solution à $y'(t)+y(t) = 0$ avec $y(0) = 1$ sur $[0;3]$ (à gauche), et $y'(t)-y(t) = 0$ avec $y(0) = 1$ sur $[0;3]$ (à droite) :



- on constate que plus le pas est petit, plus l'erreur commise est faible, mais avec un temps de calcul plus élevé ;
- on observe aussi que si l'équation différentielle est 'stable' (c'est-à-dire que la solution exacte est bornée), l'erreur globale qui provient des erreurs des itérations successives est bornée (gauche) ; mais elle diverge sinon (droite).

II. Algorithmes de la méthode d'Euler

II.1 Équation différentielle d'ordre 1

On dispose d'une fonction $f(y,t)$ déjà programmée qui est telle que $y' = f(y,t)$.

On peut définir une fonction **`euler(f,a,b,y0,n)`** qui renvoie deux tableaux **`numpy liste_t`** et **`liste_y`** dans l'intervalle $[a;b]$ avec $y_0 = y(a)$ et n correspond au nombre de points de la solution approchée.

```
def euler(f,a,b,y0,n):
    t=np.linspace(a,b,n)
    h=(b-a)/(n-1) #on veut n points, donc il faut n-1 intervalles
    y=np.zeros(n)
    y[0]=y0
    for i in range(n-1):
        y[i+1]=y[i]+h*f(y[i], t[i])
    return (t,y)
```

Plusieurs variantes sont possibles : on peut partir du pas h plutôt que n, construire des listes plutôt que des tableaux `numpy`,...

II.2 Équation différentielle d'ordre 2

a) Présentation

Dans le cas où l'équation différentielle à résoudre est d'ordre 2, on utilise une astuce à bien maîtriser : on se ramène à une équation différentielle d'ordre 1 sur une **fonction vectorielle**. Partons de l'équation d'un oscillateur amorti :

$$\ddot{y} + \frac{\omega_0}{Q}\dot{y} + \omega_0^2 y = 0 \tag{3}$$

On pose $Y = \begin{pmatrix} y \\ \dot{y} \end{pmatrix}$. L'équation à résoudre devient $\dfrac{\mathrm{d}Y}{\mathrm{d}t} = f(Y)$ où :

$$f(Y) = \begin{pmatrix} Y[1] \\ -\dfrac{\omega_0}{Q}Y[1] - \omega_0^2 Y[0] \end{pmatrix} \tag{4}$$

car $\dfrac{\mathrm{d}Y}{\mathrm{d}t} = \begin{pmatrix} \dot{y} \\ \ddot{y} \end{pmatrix} = \begin{pmatrix} \dot{y} \\ -\dfrac{\omega_0}{Q}\dot{y} - \omega_0^2 y \end{pmatrix}$.

On peut aussi définir une matrice A telle que $\dot{Y} = A.Y$ avec, pour cet exemple,

$$A = \begin{bmatrix} 0 & 1 \\ -\omega_0^2 & -\dfrac{\omega_0}{Q} \end{bmatrix}$$

L'idée principale à retenir est que résoudre une équation d'ordre 2 (ou même plus) par la méthode d'Euler n'est pas plus difficile que pour une équation différentielle d'ordre 1 : il suffit uniquement d'avoir conscience que l'on utilise une fonction f qui doit renvoyer un vecteur (à deux dimensions pour l'ordre 2), et non plus un nombre.

b) Algorithme

Voici ce que l'on pourrait écrire de manière très concrète dans Python pour résoudre et afficher la solution de l'équation de l'oscillateur amorti :

```
omega0=10
Q=20

def f(Y):
    return np.array( [ Y[1] , -omega0/Q*Y[1] - omega0**2 *Y[0] ] )

def euler(f,a,b,y0,dy0,n):
    t=np.linspace(a,b,n) #base de temps
    Y0=np.array([y0,dy0]) #conditions initiales
    Y=[Y0]#crée une liste de vect. et initialise
    h=(b-a)/n
    for i in range(n-1):
        Y.append( Y[i] + h*f(Y[i],t[i]) )#array nécessaire
    return t, np.array(Y) # on convertit le résultat en un
                          #tableau numpy de dim (n,2)

solution=euler(f,0,5,1,0,4000) #on le calcule à part
plt.plot(solution[0],solution[1][:,0]) #et ensuite on affiche
plt.show()
```

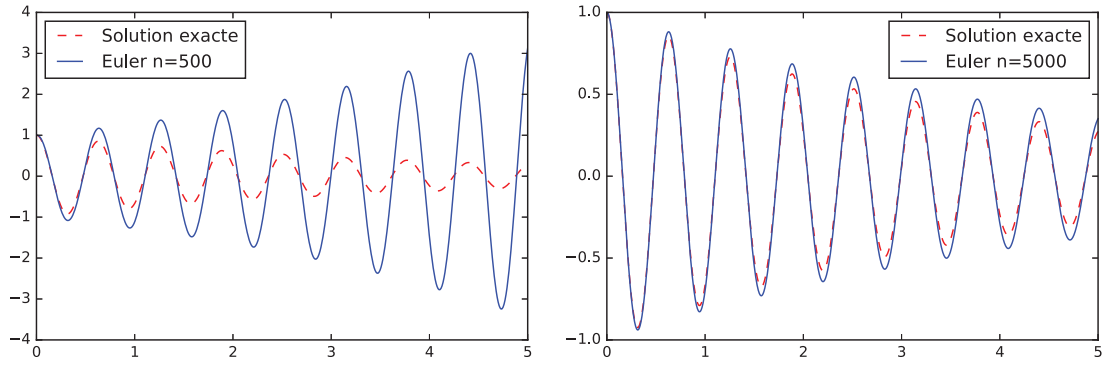
Notez qu'on introduit une variable `solution` stockée une fois pour toute, que l'on manipule ensuite autant de fois que l'on veut : cela permet de ne pas refaire le calcul un grand nombre de fois.

La structure de `solution[1]` est du type : `[[y0,dy0],[y1,dy1],[y2,dy2],...]`. La syntaxe `solution[1][:,0]` permet alors de prendre uniquement pour tous les vecteurs (le " :") le premier terme (lié à "0") : `solution[1][:,0] = [y0,y1,y2,...]`.

> On pourrait très bien faire plus simple (en plus de lignes) et ne pas utiliser des tableaux numpy, néanmoins cela a l'avantage de permettre d'avoir la dérivée au cours du temps, et donc tracer un portrait de phase, par exemple.

c) Résultats

La méthode d'Euler n'est pas sans défaut, et il s'avère que les erreurs peuvent vite s'accumuler et conduire à une solution divergente alors même que l'on étudie un système avec frottements (avec ici $n = 500$) ou convergente mais pas assez rapidement :



On peut alors soit développer d'autres techniques (par exemple la méthode de Runge-Kutta d'ordre 4) ou utiliser le module `scipy.integrate` contenant la fonction `odeint` (se référer à la notice).

On souhaite résoudre l'équation différentielle $\ddot{y} + 3y\dot{y}^2 = \cos(t)$ avec $y(0) = 0$ et $\dot{y}(0) = 1$. On peut écrire les lignes de programme correspondantes pour résoudre cette équation pour $t \in [0, 9]$ avec 10000 points (on utilisera obligatoirement la fonction `euler`, et tracer le portrait de phase associé :

```
def f(Y,t):
    return np.array([Y[1] , - 3*Y[0]*Y[1]**2 + np.cos(t)])
```

```
3
4 solution = euler(f,0,9,0,1,10000)
5 x=solution[1][:,0]
6 y=solution[1][:,1]
7 plt.plot(x,y)
8 plt.show()
```