Méthodes numériques

Delphine BRESCH-PIETRI et Pauline Bernard

Stage Liesse
Mai 2022

Programme

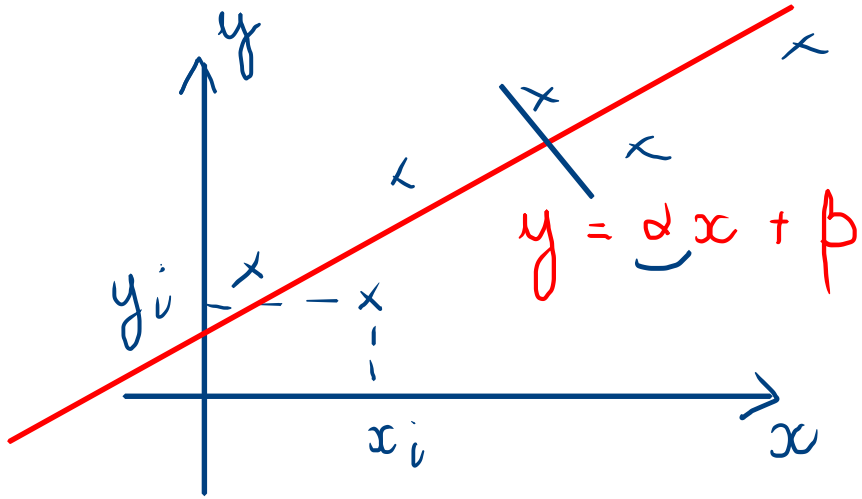
Résolution d'équations

- Equations linéaires
- Equations non linéaires

Résolution d'équations différentielles

Exemples

- Régression linéaire



(x_i, y_i), $1 \leq i \leq N$

$\alpha x_i + \beta = y_i \rightarrow \begin{cases} \alpha x_1 + \beta = y_1 \\ \alpha x_2 + \beta = y_2 \\ \vdots \end{cases}$

- Recherche de points d'équilibre

$\ddot{\theta} = -\frac{g}{l} \sin\theta$

$\dot{x} = f(x)$

$\hookrightarrow 0 = f(x)$

point d'équilibre

$n = 2$: $x_1 = \alpha$, $x_2 = \beta$

$$\begin{cases} a_{j1} = x_j \\ a_{j2} = 1 \\ b_j = y_j \end{cases}$$

inconnues

$$\begin{cases} a_{1,1}x_1 + \ldots + a_{1,n}x_n = b_1 \\ = x_1 \quad \vdots \quad = 1 \\ a_{p,1}x_1 + \ldots + a_{p,n}x_n = b_p \\ x_p \quad = 1 \end{cases} \begin{matrix} \} y_1 \\ \\ \\ \} y_p \end{matrix}$$

que l'on écrit sous forme matricielle

$$\underbrace{Ax}_{\in \mathcal{M}_{p\times n}(\mathbb{R})} = b \in \mathbb{R}^p \qquad = \begin{pmatrix} b_1 \\ \vdots \\ b_p \end{pmatrix} \in \mathbb{R}^p$$

$p \gg n$

$f : \mathbb{R}^n \to \mathbb{R}$, $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$

Cas $p > n$ (sur-déterminé) : on cherche à minimiser l'écart entre les mesures et la régression linéaire $f(x)$

$$\min_{x \in \mathbb{R}^n} \|Ax - b\|^2 = \sum_{i=1}^{?} \left(\sum_{j=1}^{n} a_{ij} x_j - b_j \right)^2$$

où $\|\cdot\|$ est la norme euclidienne dans $\mathbb{R}^p$. On peut montrer que la solution de ce problème est unique et telle que

$$\nabla f(x^\star) = 0 = 2A^T A x^\star - 2A^T b \quad \Leftrightarrow \quad \boxed{x^\star = (A^T A)^{-1} A^T b}$$

$$f(x) = \|Ax - b\|^2 = (Ax - b)^T (Ax - b)$$

$$= x^T A^T A x - 2 x^T A^T b + b^T b$$

A de rang plein (de rang n) $\Rightarrow$ $A^T A$ inversible

(on peut toujours le faire)

si $p = n$, A de rang n $\Leftrightarrow$ A inversible
(et A^T aussi)

$$x^* = (A^T A)^{-1} A^T b$$

$$= A^{-1} (A^T)^{-1} A^T b$$

$$= A^{-1} b$$

$$\min \|x\|_1 = \sum_{i=1}^{n} |x_i|$$

Cas $n > p$ (sous-déterminé) :

$$\min_{Ax=b} \|x\|^2$$

Il existe une unique solution qui est $x^\star = A^T(AA^T)^{-1}b$.

`lstsq(A,b,rcond)[0]`

de la librairie `numpy.linalg`, où

- `A` : array (p, n) des coefficients
- `b` : array $(p,)$
- `rcond` : optionnel, valeur de conditionnement de la matrice A. Mettre "rcond = None" pour éviter un warning.

La fonction renvoie :

- `x` : array à une dimension n contenant la solution $\longrightarrow x^*$
- `residuals` : scalaire, somme des résidus au carré (fonction coût) $\longrightarrow f(x^*)$
- `rank` : rang de la matrice A
- `s` : valeurs singulières de la matrice A

$$\left(\sum a_{ij} x_j - b_i\right)^2$$

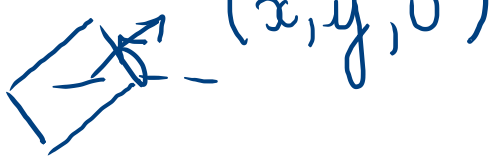
$f(x) = \begin{pmatrix} f_1(x) \\ f_2(x) \\ f_3(x) \\ \vdots \\ f_p(x) \end{pmatrix} = 0$ $\quad x \in \mathbb{R}^n$

$f : \mathbb{R}^2 \to \mathbb{R}^3$, $(x_1, x_2) \mapsto \begin{pmatrix} e^{x_1} x_2 - 3 \\ \cos x_1 \\ \sin x_2 \end{pmatrix}$

Dans le cas non-linéaire, il est souvent impossible de déterminer de façon explicite une solution à l'équation

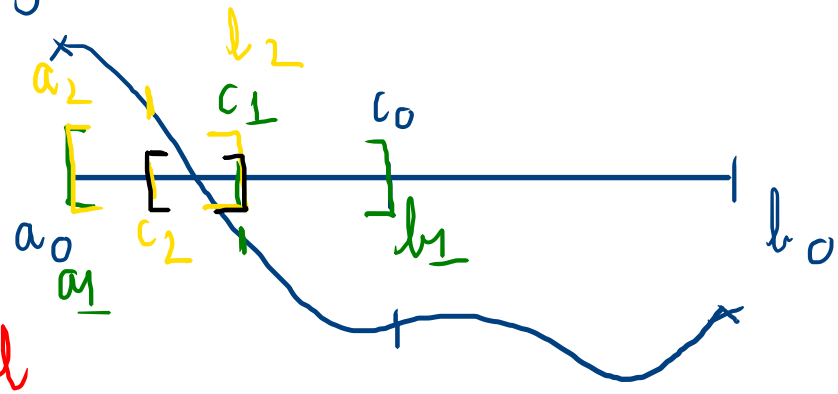
$$f(x) = 0$$

où $x \in \mathbb{R}^n$ et $f : \mathbb{R}^n \to \mathbb{R}^p$. On dispose néanmoins de méthodes numériques permettant de réaliser cela.



$$\begin{cases} \dot{x} = v\cos\theta \\ \dot{y} = v\sin\theta \\ \dot{\theta} = w \end{cases} \longrightarrow \begin{cases} v\cos\theta = 0 \\ v\sin\theta = 0 \\ w = 0 \end{cases}$$

f continue et tq $f(a_0)f(b_0) \leq 0$



Cas $n = p = 1$:

jusqu'à ce que $l([a_k, b_k]) < \text{Tol}$

Algorithme

A partir de $[a_0, b_0]$, itérer

- $c_k = \frac{a_k + b_k}{2}$
- *si* $f(a_k)f(c_k) < 0$, *choisir* $a_{k+1} = a_k$ *et* $b_{k+1} = c_k$
- *sinon, choisir* $a_{k+1} = c_k$ *et* $b_{k+1} = b_k$

→ permet de trouver une solution x^*

à chaque itération, on améliore l'estimation "linéairement"

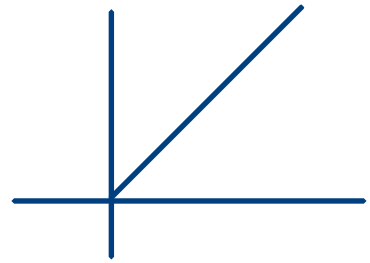
$$\|x_{k+1} - x^*\| \leq \frac{1}{2}\|x_k - x^*\| \leq \frac{1}{2^k} l([a_0, b_0])$$

⇒ convergence linéaire du rapport 1/2

$n = 1$:

$f(x) = 0 \iff x = g(x)$

$\iff x - f(x) = x$



Si la fonction $g = Id - f$ est contractante, cad s'il existe une constante $\kappa \in [0, 1[$ telle que

$$\forall (x_1, x_2) \in \mathbb{R}^n \times \mathbb{R}^n \quad \|g(x_1) - g(x_2)\| \leq \kappa \|x_1 - x_2\|$$

on peut alors résoudre l'équation en itérant la relation

$$x_{k+1} = g(x_k) = x_k - f(x_k)$$

Théorème du point fixe de Banach :

si g est contractante, $\exists !\ x^*$ tq $x^* = g(x^*)$

et la suite $\begin{cases} x_{n+1} = g(x_n) \\ x_0 \in \mathbb{R}^n \end{cases}$ est convergente de limite x^*

$$\dot{T} = -a\,T + b\,u(t)$$

$$\longrightarrow \quad T(t) = e^{-at}\,T_0 + \int_0^t e^{-a(t-s)}\,b\,u(s)\,ds$$

on veut trouver t_1 tq $T(t_1) = T_1$

soit tq $f(t_1) = T(t_1) - T_1 = 0$ ||

$$<=> \quad t_1 = -\frac{1}{a}\,\ln\left(\frac{T_1 - \int_0^{t_1} e^{-a(t_1-s)}\,b\,u(s)\,ds}{T_0}\right)$$

$$\boxed{t_1 \overset{\Delta}{=} g(t_1)}$$

Cas $n = p$: $f : \mathbb{R}^n \to \mathbb{R}^n$

$$\begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & & & \\ \vdots & & & \\ & & & \end{pmatrix} = \left(\frac{\partial f_i}{\partial x_j} \right)_{1 \le i,j \le n}$$

Si f est différentiable, Taylor à l'ordre 1

$$f(x^*) = 0 = f(x) + J_f(x)(x^* - x) + o(\|x - x^*\|)$$

$\simeq f(x) + Jf(x)(x^* - x)$

où $J_f(x^*)$ est la matrice jacobienne de f au point x^*.

$$\Leftrightarrow \quad x^* = x - Jf(x)^{-1} f(x)$$

La méthode de Newton-Raphson consiste à utiliser cette approximation et à choisir

$$x_{k+1} = x_k - J_f(x_k)^{-1} f(x_k)$$

on voudrait que cette méthode converge vers une (ou la) solution x^* du problème

$\Rightarrow \|x_{k+1} - x^*\| \le M \|x_k - x^*\|^2 \le (M \|x_0 - x^*\|)^{2^k}$

convergence quadratique

$f \in C^2$ et $f : \mathbb{R} \to \mathbb{R}$

$$x_{k+1} = x_k - \frac{1}{f'(x_k)} f(x_k) \quad (**)$$

et Taylor-Young avec reste intégral donne

$$\forall x \in \mathbb{R} \quad f(x) = f(x_k) + f'(x_k)(x - x_k) + \int_{x_k}^{x} f''(t)(x-t)\,dt \quad (*)$$

donc $x_{k+1} = x_k - \frac{1}{f'(x_k)} \Big[\underbrace{f(x^*)}_{=0} - f'(x_k)(x^* - x_k) - \int_{x_k}^{x^*} f''(t)(x^* - t)\,dt \Big]$

par $(*)$ dans $(**)$ pour $x = x^*$

$$x_{k+1} = x_k - \frac{1}{f'(x_k)} \left[f(x^*) - f'(x_k)(x^* - x_k) - \int_{x_k}^{x^*} f''(t)(x^* - t)\,dt \right]$$

donc

$$x_{k+1} - x^* = x_k - x^* - \frac{1}{f'(x_k)} \left[- f'(x_k)(x^* - x_k) \right] + \frac{1}{f'(x_k)} \int_{x_k}^{x^*} f''(t)(x^* - t)\,dt$$

f', f'' bornées au $v(x^*)$

$$\Rightarrow \|x_{k+1} - x^*\| \leq M \left| \int_{x_k}^{x^*} (x^* - t)\,dt \right|$$

$$\leq M_0 \|x_k - x^*\|^2$$

$f(x) = 0 \iff \|f(x)\|^2 = 0$

$\iff \min_x \|f(x)\|^2$

Cas $p \geq n$:
On s'intéresse au problème

$$\min_{x \in \mathbb{R}^n} \|f(x)\|^2 = F(x) \tag{1}$$

$\min_{x \in \mathbb{R}^n} F$ $\quad F(x) = \|f(x)\|^2$ avec $F : \mathbb{R}^n \to \mathbb{R}$

$= \sum_{i=1}^{p} f_i(x)^2$

Si F est différentiable, on a

$$F(x) = F(x_k) + \nabla F(x_k)^T (x - x_k) + o(\|x - x_k\|) \tag{2}$$

$\simeq F(x_k) + \nabla F(x)^T (x - x_k) = F(x_{k+1})$

on prend $x_{k+1} = x_k - l \nabla F(x_k)$

Méthode de gradient à pas fixe

$$x_{k+1} = x_k - l \nabla F(x_k) \tag{3}$$

$- l \|\nabla F(x_k)\|^2 \leq 0$

$l \geq 0$

obligatoire

`bisect(f,a,b,args)` optionnel → dichotomie

de la librairie `scipy.optimize`, où

- `f` : fonction $\mathbb{R} \to \mathbb{R}$,
- `a` : scalaire de début d'intervalle
- `b` : scalaire de fin d'intervalle
- `options` : nombre maximal d'itérations, tolérance, ...

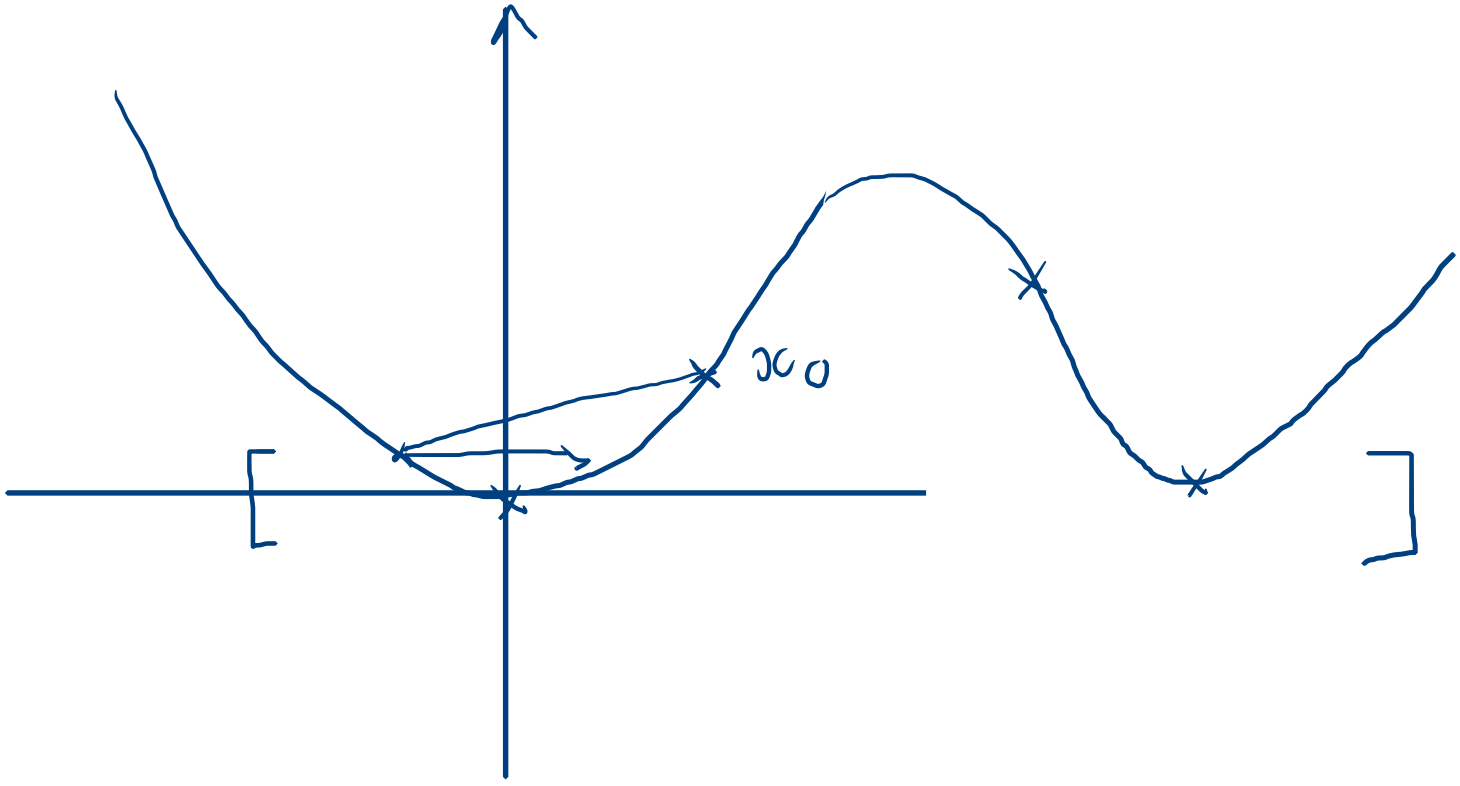
obligatoires

`fsolve(f,x0,args)` optionnel → Newton-Raphson / descente de gradient (Gauss-Newton)

de la librairie `scipy.optimize`, où

- `f` : fonction $\mathbb{R}^n \to \mathbb{R}^n$,
- `x0` : condition initiale de taille n
- `options` : jacobien, nombre maximal d'itérations, tolérance, ...

scipy.optimize.minimize ←


x_0

Programme

Résolution d'équations
- Equations linéaires
- Equations non linéaires

$f(x) = 0$, $x \in \mathbb{R}^n$

Résolution d'équations différentielles

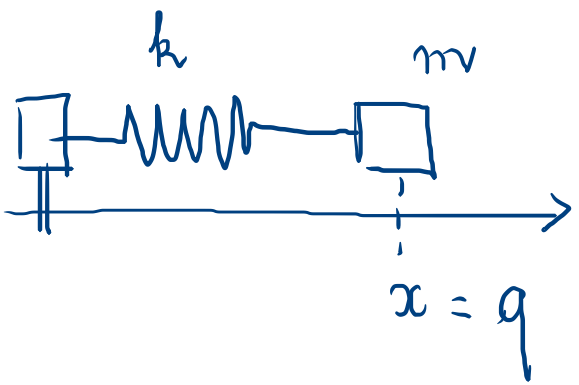
$\dot{x} = f(t, x)$, $x(t) \in \mathbb{R}^n$

Exemples

- concentrations des espèces chimiques dans une réaction $A + B \underset{k'}{\overset{k}{\rightleftharpoons}} C$

$$\frac{dc_A}{dt} = \begin{aligned} \dot{c}_A &= -k\, c_A c_B + k' c_C \\ \dot{c}_B &= -k\, c_A c_B + k' c_C \\ \dot{c}_C &= k\, c_A c_B - k' c_C \end{aligned} \quad \rightarrow \text{3 EDO d'ordre 1}$$

- mécanique Newtonienne ou Lagrangienne



EDO ordre 2 $\leftarrow$ $M\ddot{q} = \sum_k F_k(t, q, \dot{q})$

y de classe C^p vérifie

$$y^{(p)}(t) = \psi\left(t, y(t), \dot{y}(t), \dots, y^{(p-1)}(t)\right)$$

si et seulement si $x = (y, \dot{y}, \dots, y^{(p-1)})$ de classe C^1 vérifie

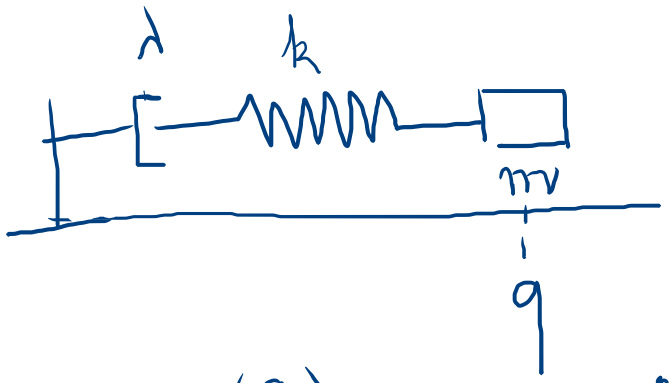
$$\dot{x} = f(t, x) \ ,$$

où f est définie par

$$f(t, x_1, x_2, \dots, x_p) = \begin{pmatrix} x_2 \\ x_3 \\ \vdots \\ x_p \\ \psi(t, x_1, \dots, x_p) \end{pmatrix}$$

$\Longrightarrow$ toujours se mettre sous la forme $\dot{x} = f(t, x)$: crucial pour étude mathématique et numérique !

équation scalaire d'ordre p $\Leftrightarrow$ équation vectorielle (dans $\mathbb{R}^p$) d'ordre 1

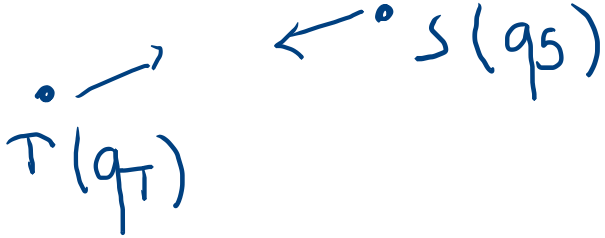


$$m\ddot{q} = -kq - \lambda\dot{q}$$

$$x = \begin{pmatrix} q \\ \dot{q} \end{pmatrix} \qquad \dot{x} = \begin{cases} \dot{q} = x_2 \\ \ddot{q} = -\frac{1}{m}\left(kq + \lambda\dot{q}\right) = -\frac{1}{m}\left(kx_1 + \lambda x_2\right) \end{cases}$$

$$= f(x)$$

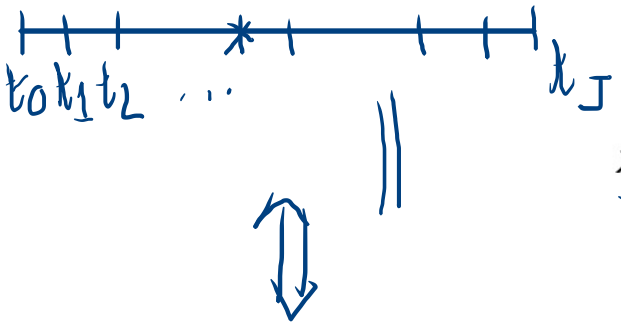
$$f : x \in \mathbb{R}^2 \longmapsto \begin{pmatrix} x_2 \\ -\frac{1}{m}\left(kx_1 + \lambda x_2\right) \end{pmatrix}$$



$$\begin{cases} m_T \ddot{q}_T = \dots \\ m_S \ddot{q}_S = \dots \end{cases}$$

$$x = (q_T, \dot{q}_T, q_S, \dot{q}_S)$$

$$\dot{x} = \begin{pmatrix} \dot{q}_T \\ \frac{1}{m_T}(\ ) \\ \dot{q}_S \\ \frac{1}{m_S}(\ ) \end{pmatrix}$$



$$\dot{x} = f(t,x) \quad , \quad x(t_0) = x_0$$

$$x(t) = x_0 + \int_{t_0}^{t} f(s,x(s))ds = x_0 + \sum_{j=0}^{J-1} \int_{t_j}^{t_{j+1}} f(s,x(s))\,ds$$

$$t_0 < t_1 < \ldots < t_J = t$$

Idée : approximer l'intégrale sur des intervalles $[t_j, t_{j+1}]$ suffisamment petits.

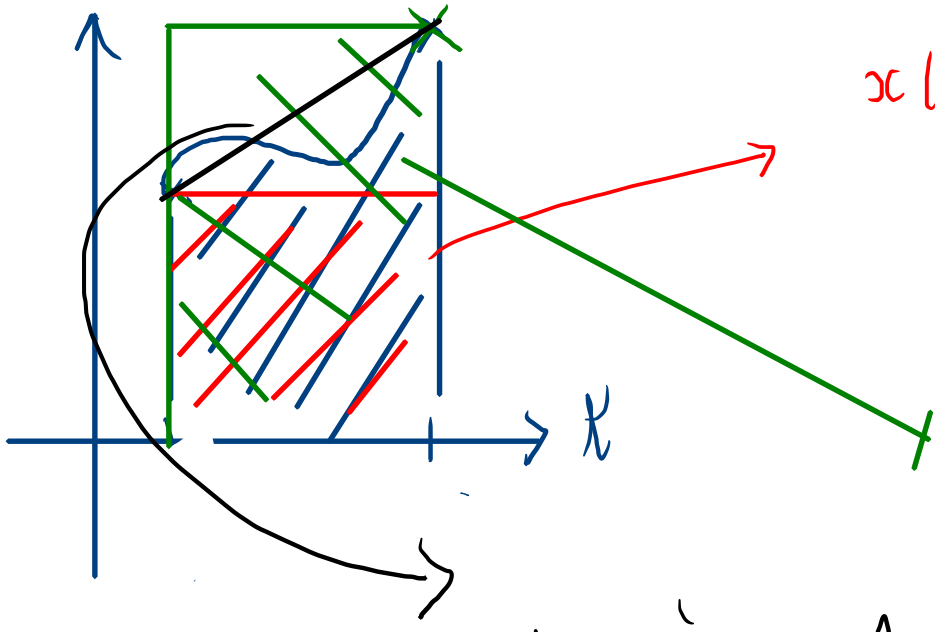
x^j : approximation de $x(t_j)$ $\qquad \Delta t_j = t_{j+1} - t_j$

On implémente

$$x^{j+1} = x^j + \Delta t_j \, \Phi(t_j, x^j, \Delta t_j)$$

tel que

$$\Phi(t_j, x^j, \Delta t_j) \approx \frac{1}{\Delta t_j} \int_{t_j}^{t_{j+1}} f(s,x(s))\,ds$$


x(t_j)
Euler implicito
x(t_{j+1})
Euler implicito
Trapèzo
1/2 (x(t_j) + x(t_{j+1}))

$$\begin{cases} x_{j+1} = x_j + \Delta t_j f(t_j, x_j) \\ x_0 = x_0 \end{cases} \qquad \dot{x} = f(t,x) \quad x(0) = x_0$$

Taylor ordre 1 :

$$x(t_{j+1}) = x(t_j) + \Delta t_j \dot{x}(t_j) + o(\Delta t_j)$$
$$= x(t_j) + \Delta t_j f(t_j, x(t_j)) + o(\Delta t_j)$$

donc $\max_j \| x_j - x(t_j) \| \leq c \Delta t$ avec $\Delta t = \max_j \Delta t_j$

$\Rightarrow$ schéma d'ordre 1

Limites $\rightleftarrows$ ordre 1 "

$\rightarrow$ systèmes "raides" (constantes de temps ≠tes)

$\searrow$ systèmes conservatifs / hamiltoniens

$$\begin{cases} x_{j+1} = x_j + \Delta t_j f(t_{j+1}, x_{j+1}) \\ x_0 = x_0 \end{cases} \qquad \begin{cases} \dot{x} = f(t,x) \\ x(0) = x_0 \end{cases}$$

on cherche à résoudre $x_{j+1} = F(x_{j+1})$ ||

où $F : x \in \mathbb{R}^n \mapsto x_j + \Delta t_j f(t_{j+1}, x)$

si $f : (t,x) \mapsto f(t,x)$ est K-lipschitzienne par rapport à x :

$$\exists K \geq 0 \quad \forall (x_1, x_2) \in \mathbb{R}^n \times \mathbb{R}^n$$

$$\forall t \in \mathbb{R} \quad \|f(t,x_1) - f(t,x_2)\| \leq K \|x_1 - x_2\|$$

$$\|F(a) - F(b)\| = \|x_j + \Delta t_j f(t_{j+1}, a) - [x_j + \Delta t_j f(t_{j+1}, b)]\|$$

$$= \Delta t_j \|f(t_{j+1}, a) - f(t_{j+1}, b)\| \leq \Delta t_j K \|a - b\|$$

$\Rightarrow$ contractante si $\Delta t_j K < 1$: on résout par point fixe

$$\begin{cases} x_{j+1}^{k+1} = F(x_{j+1}^{k}) \\ x_{j+1}^{0} = x_j + \Delta t_j f(t_j, x_j) \end{cases}$$ Euler implicito

$x(t) = \exp(-\lambda t)$ ← $\dot{x} = -\lambda x \qquad x(0) = 1$ avec $\lambda > 0$

Euler explicite : $x_{j+1} = x_j - \Delta t \lambda x_j$

$= (1 - \Delta t \lambda) x_j$

si $\Delta t \lambda < 2$, alors $x_j \xrightarrow[j \to +\infty]{} 0$

Euler implicite : $x_{j+1} = x_j - \Delta t \lambda x_{j+1}$

donc $(1 + \lambda \Delta t) x_{j+1} = x_j$

stabilité inconditionnelle, $x_{j+1} = \frac{1}{1 + \lambda \Delta t} x_j \xrightarrow[j \to +\infty]{} 0$

$\Longrightarrow$ Explicite ou implicite pour $\dot{x} = \begin{pmatrix} -1 & 0 \\ 0 & -\mu \end{pmatrix} x$ avec $\mu >> 1$?

Schéma convergent à l'ordre p si $\exists c_v > 0$ (indépendent de Δt) tel que

$$\max_{1 \leq j \leq J} \|x^j - x(t_j)\| \leq c_v (\Delta t)^p$$

Détermination numérique de l'ordre :

plusieurs (x_j) correspondant à plusieurs valeurs de Δt

$\log(\max \|x_j - x(t_j)\|) \% \log(\Delta t)$

Détermination analytique de l'ordre :

$$x(t_{j+1}) = x(t_j) + \Delta t_j \dot{x}(t_j) + \frac{\Delta t_j^2}{2} \ddot{x}(t_j) + o(\Delta t_j^2)$$

$= f(t_j, x(t_j))$

$= \partial_t f(t_j, x(t_j)) + \partial_x f(t_j, x(t_j)) \dot{x}(t_j)$

$= f(t_j, x(t_j))$

- méthode de Heun (ordre 2) :

$$x^{j+1} = x^j + \frac{\Delta t_j}{2}\left(f(t_j, x^j) + f\left(t_{j+1}, x^j + \Delta t_j f(t_j, x^j)\right)\right),$$

ou sa version implicite méthode des trapèzes (ou Crank–Nicolson) :

$$x^{j+1} = x^j + \frac{\Delta t_j}{2}\left(f(t_j, x^j) + f(t_{j+1}, x^{j+1})\right).$$

- schéma de Runge–Kutta d'ordre 4 (RK4) largement utilisée:

$$x^{j+1} = x^j + \Delta t_j \frac{F_1 + 2F_2 + 2F_3 + F_4}{6} \quad \text{avec} \begin{cases} F_1 = f(t_j, x^j) \\ F_2 = f\left(t_j + \frac{\Delta t_j}{2}, x^j + \frac{\Delta t_j}{2} F_1\right) \\ F_3 = f\left(t_j + \frac{\Delta t_j}{2}, x^j + \frac{\Delta t_j}{2} F_2\right) \\ F_4 = f(t_j + \Delta t_j, x^j + \Delta t_j F_3), \end{cases}$$

- adaptation du pas

`odeint(fun,x0,t,args)`

de la librairie `scipy.integrate`, où

- `fun` : fonction prenant en entrée (y, t) et renvoyant un array $f(y, t)$,
- `y0` : la condition initiale
- `t` : suite des temps auxquels on souhaite résoudre y
- `options` : jacobien de la fonction, tolérance, points critiques, pas de temps min ou max, ...

La fonction renvoie :

- `y` : array à deux dimensions de taille $(len(t), len(y_0))$

`solve_ivp(fun,t_span,x0,method,t_eval)`

de la librairie `scipy.integrate`, où

- `fun` : fonction prenant en entrée (t, x) et renvoyant un array $f(t, x)$,
- `t_span` : intervalle d'intégration $[t_0, t_f]$
- `x0` : la condition initiale
- `method` : méthode d'intégration : par défaut, en mettant [ ] ou en nommant les entrées suivantes, c'est "RK45", le Runge–Kutta explicite d'ordre 5 (avec contrôle du pas à l'ordre 4). Lorsque cela échoue ou demande trop d'itérations, le système est peut-être raide : utiliser alors les méthodes implicites "Radau" ou "BDF".
- `t_eval` : array donnant les temps auxquels vous voulez qu'il vous renvoie la solution (doit être dans l'intervalle `t_span`)
- `options` :

La fonction renvoie un objet `sol` de type bunch :

- `sol.t` : array à une dimension contenant les temps
- `sol.y` : array à deux dimensions `(dim_x,dim_t)`, de façon à ce que `sol.y[k]` donne l'array du k-ième état au cours du temps.