Méthodes numériques

Delphine BRESCH-PIETRI et Pauline Bernard

Stage Liesse
Mai 2022

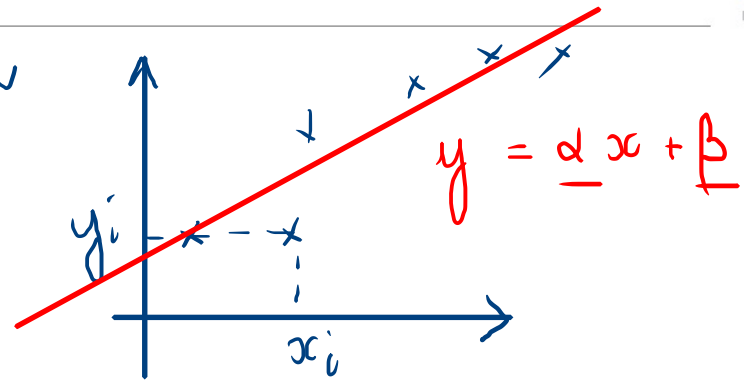
Programme

Résolution d'équations

- Equations linéaires
- Equations non linéaires

Résolution d'équations différentielles

$f(x) = 0$ $\quad (x_i, y_i),\ 1 \le i \le p$



- Régression linéaire

$$\begin{cases} \alpha x_1 + \beta = y_1 \\ \alpha x_2 + \beta = y_2 \\ \vdots \\ \alpha x_p + \beta = y_p \end{cases} \longrightarrow \sum_{i=1}^{p} (\alpha x_i + \beta - y_i)^2$$

$= dx/dt$

- Recherche de points d'équilibre

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

points d'équilibre : $0 = f(x)$

$n=2$, $\begin{cases} x_1 = \alpha \\ x_2 = \beta \end{cases}$

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

$b_j = y_j$

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

que l'on écrit sous forme matricielle

$$Ax = b$$

$A \in \mathcal{M}_{p\times n}(\mathbb{R})$, $x \in \mathbb{R}^n$, $b \in \mathbb{R}^p$

$f : \mathbb{R}^n \to \mathbb{R}$, $x \mapsto \|Ax-b\|^2$

Cas $p > n$ (sur-déterminé) :

$f(x)$

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

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

$$\nabla f(x^\star) = 0 = 2A^TAx^\star - 2A^Tb \Leftrightarrow \boxed{x^\star = (A^TA)^{-1}A^Tb}$$

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

$= x^TA^TAx - 2x^TA^Tb + b^Tb$

A de rang n $\Rightarrow$ A^TA inversible (toujours faisable)

si $n=p$, A de rang n $\Rightarrow$ A inversible

$x^\star = A^{-1}(A^T)^{-1}A^Tb$

$= A^{-1}b$

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

Cas $n > p$ (sous-déterminé) :

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

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

`lstsq(A,b,rcond)`

de la librairie `numpy.linalg`, où

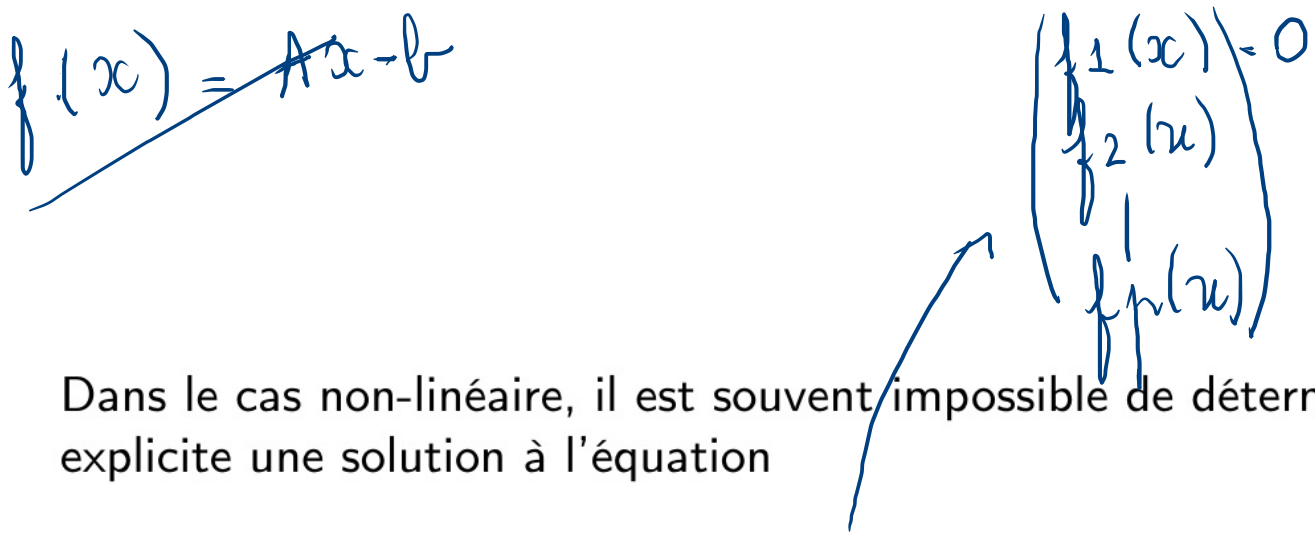
- `A` : array (p, n) des coefficients
- `b` : array $(p,)$
- `rcond` : optionnel, valeur de conditionnement de la matrice A. Mettre "rcond = None" pour éviter un warning.

La fonction renvoie :

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

$$= \|Ax^* - b\|^2$$

$$= \sum_i \left(\sum_j a_{ij} x_j - b_i \right)^2$$

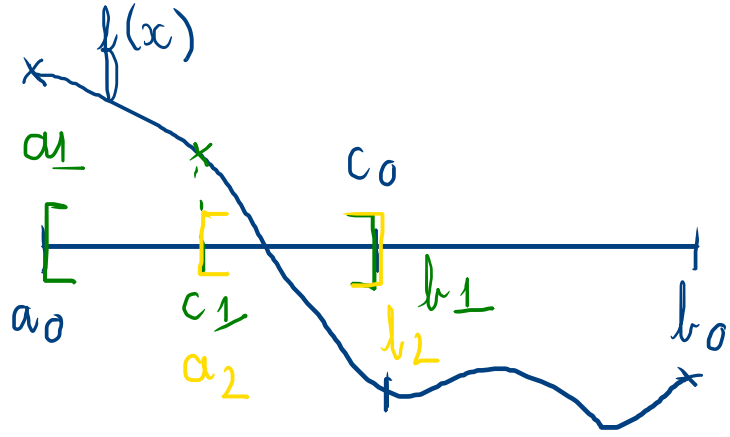


Dans le cas non-linéaire, il est souvent impossible de déterminer de façon explicite une solution à l'équation

$$f(x) = 0$$

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.

Cas $n = p = 1$:



Algorithme

A partir de $[a_0, b_0]$ *itérer* tant que $\|a_k - b_k\| \geq Tol$

- $c_k = \frac{a_k + b_k}{2}$
- *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* $b_{k+1} = b_k$

$\leq \frac{1}{2^k} \|a_0 - b_0\|$

f continue et $[a_0, b_0]$ tel tq $f(a_0)f(b_0) < 0$

convergence vers x^* solution de $f(x^*) = 0$ avec vitesse de convergence linéaire de rapport 1/2 : $\|a_{k+1} - b_{k+1}\| \leq \frac{1}{2}\|a_k - b_k\|$

$f(x) = 0 \iff x = g(x)$ $\quad g : \mathbb{R}^n \to \mathbb{R}^n$

Si la fonction g 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

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

Théorème du point fixe de Banach :

si g est contractante, il existe un unique x^* tq $x^* = g(x^*)$

et $\begin{cases} x_{k+1} = g(x_k) \\ x_0 \in \mathbb{R}^n \end{cases}$ est convergente et tq $\lim_{k \to +\infty} x_k = x^*$

$$\begin{cases} \dot{T} = -aT + b\,u(t) \\ T(0) = T_0 \end{cases}$$

on cherche t_1 le temps où $T(t_1) = T_1$

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

on veut trouver $e^{-at_1} T_0 + \int_0^{t_1} e^{-a(t_1-s)} b\,u(s)\,ds - T_1 = 0$

$\triangleq f(t_1)$

ceci est équivalent à $t_1 = -\frac{1}{a} \ln\left(\frac{T_1 - \int_0^{t_1} e^{-a(t_1-s)} b\,u(s)\,ds}{T_0} \right)$

$\triangleq g(t_1)$

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

Cas $n = p$:

$$\left(\frac{\partial f_i}{\partial x_j}\right) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} & \cdots \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots \\ | & | & \end{pmatrix}$$

Si f est différentiable,

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

$\in \mathcal{M}_{n \times n}(\mathbb{R})$

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

$$0 \simeq f(x) + Jf(x)(x^* - x) \Rightarrow x^* = x - Jf(x)^{-1} f(x)$$

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

$$\begin{cases} x_0 \in \mathbb{R}^n \\ x_{k+1} = x_k - J_f(x_k)^{-1} f(x_k) \end{cases}$$

on voudrait que cette relation converge vers x^*

si f C^2 avec des hypothèses de bonitude, on a CV quadratique de cette méthode vers x^*

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

la méthode de Newton-Raphson consiste à itérer

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

[par ailleurs, par Taylor avec reste intégral,

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

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

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

f'' : on suppose ces termes bornés

$$\Rightarrow \|x_{k+1} - x^*\| \leq M \|x^* - x_k\|^2$$

convergence quadratique

$$\leq (M \|x_0 - x^*\|)^{2^k}$$

$$< 1/M$$

Cas $p = n$:

On s'intéresse au problème

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

Si F est différentiable, on a

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

Méthode de gradient à pas fixe

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

dichotomie obligatoire

`bisect(f,a,b,args)`

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, ...

obligatoire

`fsolve(f,x0,args)`

de la librairie `scipy.optimize`, où

- `f` : fonction $\mathbb{R}^n \to \mathbb{R}^n$,
- `x0` : condition initiale de taille n , array à 1 dim
- `options` : jacobien, nombre maximal d'itérations, tolérance, ...

Programme

Résolution d'équations

- Equations linéaires
- Equations non linéaires

Résolution d'équations différentielles

Exemples

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

$$\begin{cases} \dot{c}_A = -k\, c_A c_B + k' c_C \\ \dot{c}_B = -k\, c_A c_B + k' c_C \\ \dot{c}_C = k\, c_A c_B - k' c_C \end{cases}$$

"ordre 1"

- mécanique Newtonienne ou Lagrangienne

$$M\ddot{q} = \sum_k F_k(t, q, \dot{q})$$

"ordre 2"

y de classe C^p vérifie

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

équation scalaire d'ordre p

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

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

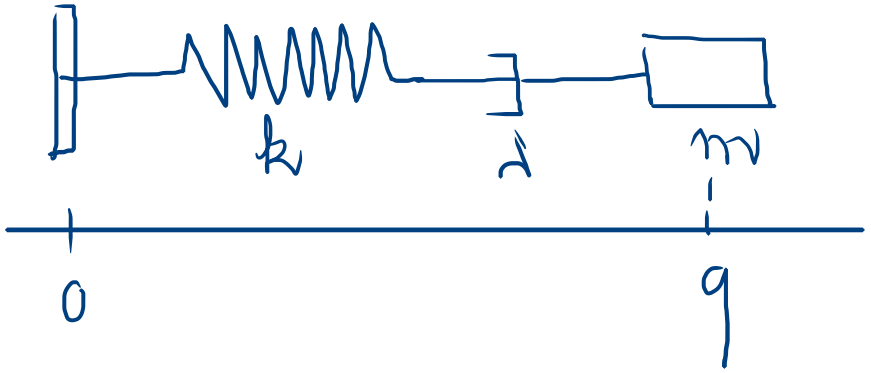
équation vectorielle ($\in \mathbb{R}^p$) d'ordre 1

où f est définie par

$$\dot{x} = \begin{pmatrix} \dot{y} \\ \ddot{y} \\ \vdots \\ y^{(p)} \end{pmatrix} = \begin{pmatrix} x_2 \\ x_3 \\ \vdots \\ x_p \\ \psi(t, x_1, x_2, \dots, x_p) \end{pmatrix} \overset{\Delta}{=} f(t, x_1, x_2, \dots, x_p)$$

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

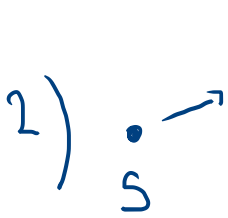
1)



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

$$x = \begin{pmatrix} q \\ \dot{q} \end{pmatrix} \quad \text{et alors} \quad \dot{x} = \begin{pmatrix} x_2 \\ -\frac{1}{m}(kx_1 + \lambda x_2) \end{pmatrix} = f(x)$$

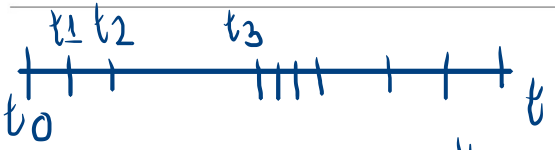
2)



$m_S \ddot{q}_S = \cdots$

$m_T \ddot{q}_T = \cdots$

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



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

forme intégrale

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

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

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

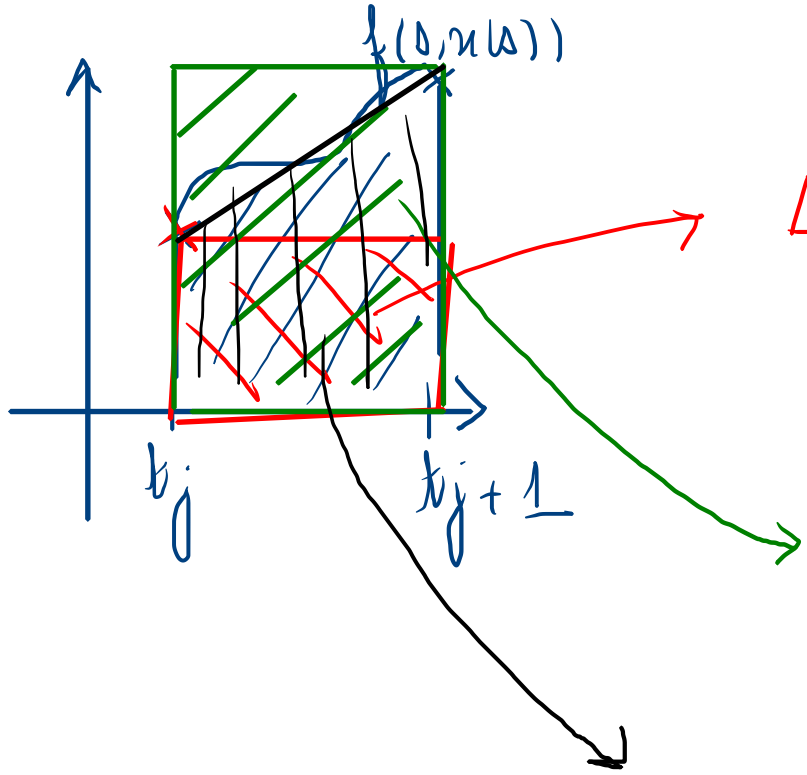
x^j : approximation de $x(t_j)$ $\qquad$ $\Delta t_j = t_{j+1} - t_j$

On implémente

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

tel que

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



$\Delta t_j \, f(t_j, u(t_j))$

Euler explicite

$\Delta t_j \, f(t_{j+1}, u(t_{j+1}))$

Euler implicite

Trapèze

$$\frac{1}{2} \Delta t_j \left[f(t_j, u(t_j)) + f(t_{j+1}, u(t_{j+1})) \right]$$

$$\begin{cases} \dot{x} = f(t,x) \\ x(t_0) = x_0 \end{cases} \longrightarrow \begin{cases} x_{j+1} = x_j + \Delta t_j \, f(t_j, x_j) \\ x_0 = x_0 \end{cases}$$

Taylor à l'ordre 1 pour x :

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

on a donc

$$\max_{1 \le j \le J} \| x(t_j) - x_j \| \le \underbrace{c}_{\in \mathbb{R}_+} \Delta t \quad \text{avec } \Delta t = \max_{1 \le i \le J} \Delta t_j$$

Limites
- ordre 1
- systèmes "raides" (avec des critères de temps $\neq$ tes)
- systèmes conservatifs / hamiltoniens

$$\begin{cases} x_{j+1} = x_j + \Delta t_j \, f(t_{j+1}, x_{j+1}) \triangleq F(x_{j+1}) \\ x_0 = x_0 \end{cases}$$

où

$$F : x \mapsto x_j + \Delta t_j \, f(t_{j+1}, x)$$

on souhaite résoudre cette équation par point-fixe et il faut donc que F soit contractante

supposons que f est Lipschitzienne de constante K par rapport à x :
cad $\forall (a,b) \in \mathbb{R}^n \times \mathbb{R}^n \; \forall t \in \mathbb{R}$

$$\| f(t,a) - f(t,b) \| \leq K \| a - b \|$$

alors

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

$$= \Delta t_j \, \| f(t_{j+1}, a) - f(t_{j+1}, b) \| \leq \underbrace{\Delta t_j K}_{<1 \text{ si } \Delta t_j < 1/K} \| a - b \|$$

on résout l'équation sur x_{j+1} par point fixe

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

arrêt lq $\| x_{j+1}^{k+1} - x_{j+1}^{k} \| \leq Tol$

$\Rightarrow$ méthode d'ordre 1

$x(t) = e^{-\lambda t}$ $\longleftarrow$ $\dot{x} = -\lambda x \qquad x(0) = 1$ avec $\lambda > 0$

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

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

$x_j \underset{j \to +\infty}{\longrightarrow} 0$ seulement si $|1 - \lambda \Delta t| < 1$

donc pour $\boxed{\Delta t < \frac{2}{\lambda}}$

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

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

$x_{j+1} = \frac{1}{1+\lambda \Delta t}\, x_j \underset{j \to +\infty}{\longrightarrow} 0$

convergence inconditionnelle sur Δt

$\Longrightarrow$ 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

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

Détermination numérique de l'ordre :

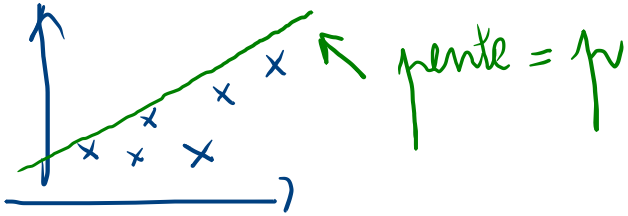
on réalise plusieurs simulations pour un Δt différent à chaque fois
on plotte $\log(\max \|x_j - x(t_j)\|)$ par rapport $\log \Delta t$

Détermination analytique de l'ordre :

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

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

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



- 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 Δt_j

`odeint(fun,y0,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.