Informatique - CN3

Résolution numérique d'une équation différentielle ordinaire

D.Malka

MPSI 2018-2019

Sommaire

Sommaire

Equations différentielles

Equations différentielles ordinaires (EOD)

$$y' = ay$$

$$y' = \frac{y}{a.t + y}$$

$$\begin{cases} x' = ax - bxy \\ y' = cy + dyx \end{cases}$$

Forme générale d'une équation d'ordre 1 pour une fonction dépendant d'une seule variable t :

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

Existence de solutions

Existence de solutions

La majorité des équations différentielles n'admet pas de solution analytique.

Mais des preuves non constructives peuvent assurer qu'elles admettent une solution.
Exemple : le théorème de Cauchy-Lipschitz.

Sommaire

Résolution approchée

Résolution approchée

On détermine de manière algorithmique une solution approchée de l'équation différentielle en discrétisant l'équation.

La fonction solution de l'équation différentielle est alors une liste d'approximations $(\tilde{y}_i)_{i\in\mathbb{N}}$ des valeurs $(y(t_i))_{i\in\mathbb{N}}$ prises par la fonction pour différents antécédents $(t_i)_{i\in\mathbb{N}}$.

Il faut s'assurer de la convergence et de la stabilité numérique de l'algorithme. Il faut contrôler et estimer l'erreur commise par rapport à la solution exacte.

Schéma numérique

Soit le problème de Cauchy :

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

On appelle *schéma numérique* la relation de récurrence permettant de calculer successivement les termes de la suite $(\tilde{y}_i)$ en discrétisant les abscisses (t_i).

Méthode d'Euler explicite

- On cherche à résoudre sur $[t_0, T]$ l'équation de la forme :

$$y' = f(y(t), t) \quad \text{avec} \quad y(t_0) = y_0$$

- Si $y(t)$ est de classe $\mathcal{C}_1$ alors, connaissant $y(t_i)$, on peut évaluer de façon approchée, $y(t_{i+1})$ par son développement de Taylor à l'ordre 1.

$$y(t_{i+1}) = y(t_i) + y'(t_i)(t_{i+1} - t_i)$$

- Avec $y' = f(y(t), t)$, il vient $y(t_{i+1}) = y(t_i) + f(y(t_i), t_i)(t_{i+1} - t_i)$.
- Sauf qu'on ne connaît pas les images $y(t_i)$ sauf pour $t_0 = a$.
- On calcule donc une valeur approchée $\tilde{y}_1$ de $y(t_1)$ ainsi :

$$y(t_1) \approx \tilde{y}_1 = y_0 + f(y_0, t_0)(t_1 - t_0)$$

- Par récurrence, on détermine tous les termes de la suite $(\tilde{y}_i)$ via le *schéma numérique* d'Euler explicite :

$$\tilde{y}_{i+1} = \tilde{y}_i + f(\tilde{y}_i, t_i)(t_{i+1} - t_i)$$

- Si en plus, le pas de discrétisation $h = t_{i+1} - t_i$ est constant :

Méthode d'Euler explicite

- Par récurrence, on détermine tous les termes de la suite $(\tilde{y}_i)$ via le *schéma numérique* d'Euler explicite :

$$\tilde{y}_{i+1} = \tilde{y}_i + f(\tilde{y}_i, t_i)(t_{i+1} - t_i)$$

- Si en plus, le pas de discrétisation $h = t_{i+1} - t_i$ est constant :

$$\boxed{\tilde{y}_{i+1} = \tilde{y}_i + f(\tilde{y}_i, t_i)h} \quad \text{avec} \quad \boxed{t_i = t_0 + i.h}$$

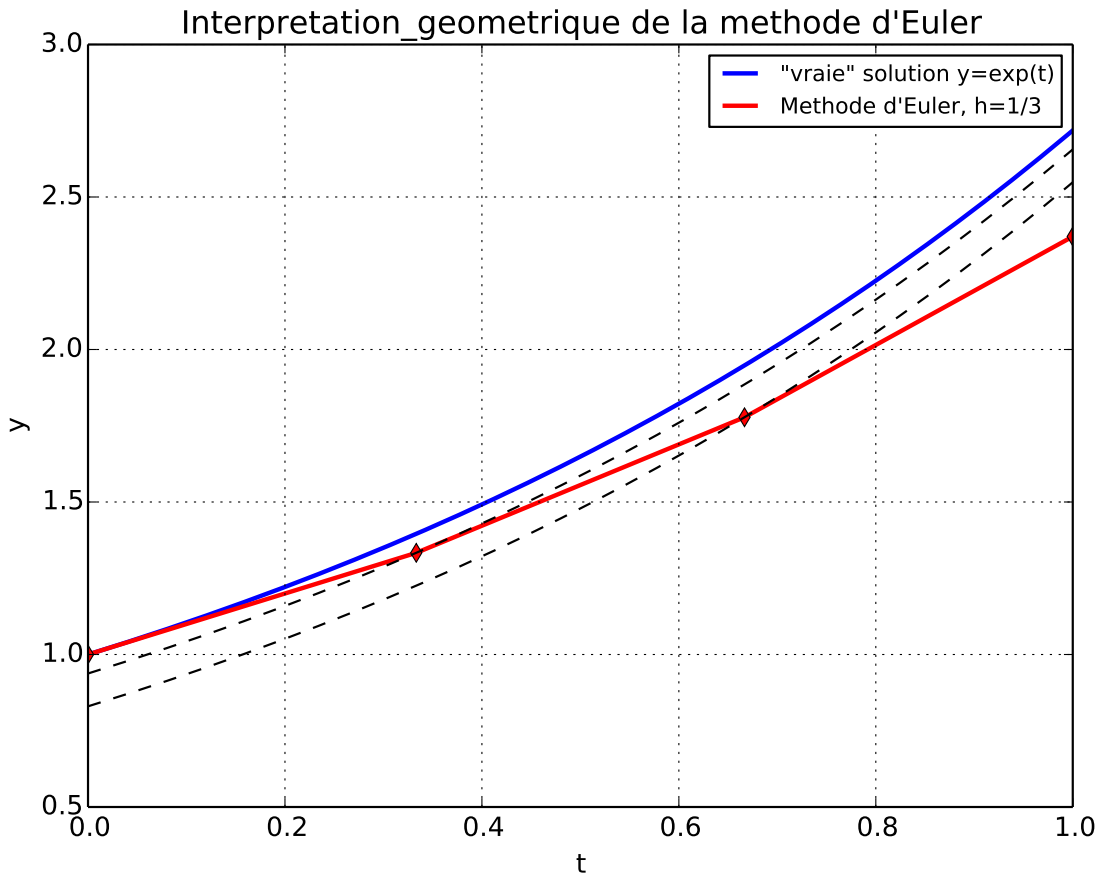
Attention $\tilde{y}_i \neq y(t_i)$!

Il est important de comprendre que les termes suite $(\tilde{y}_i)$ ne sont pas les images $y(t_i)$ de la fonction mais des valeurs approchées.

On espère qu'en réduisant le pas h de discrétisation, cette suite converge vers la suite $y(t_i)$ i.e. que :

$$\forall i \lim_{h \to 0} \tilde{y}_i = y(t_i)$$

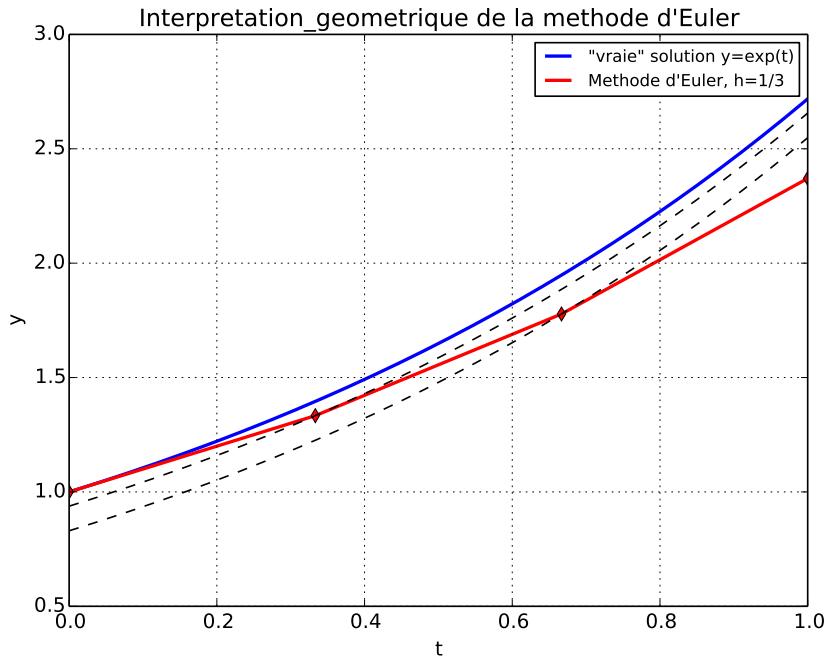
Interprétation géométrique



Implémentation en Python

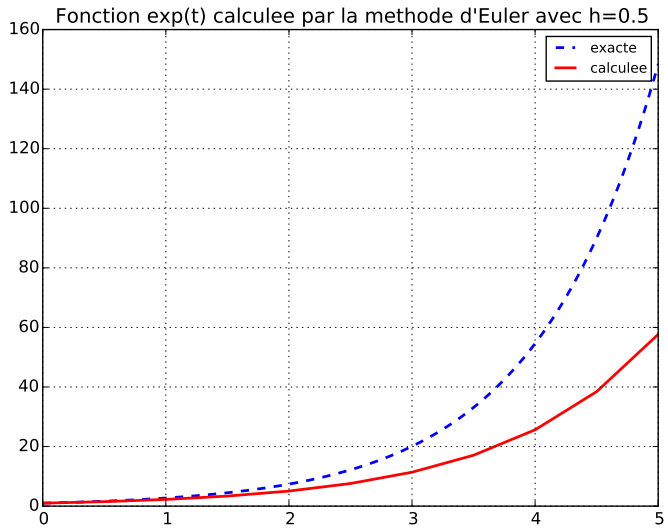
```
1
2  def Euler(f,y0,a,b,n):
3      '''
4      Integre l'equation y'(t)=f(y(t),t) avec y(a)=y0 selon le schema
           explicite d'Euler
5      f : function
6      y0,a,b :floats. [a,b] : intervalle de resolution
7      n : int, nombre de points. Pas de discretisation : h=(b-a)/(n-1)
8      t,y : list of floats
9      '''
10     h=(b-a)/(n-1)
11     y=np.zeros(n);y[0]=y0
12     t=np.linspace(a,b,n,endpoint=True)
13     for i in range(0,n-1):
14         y[i+1]=y[i]+f(y[i],t[i])*h
15     return t,y
```

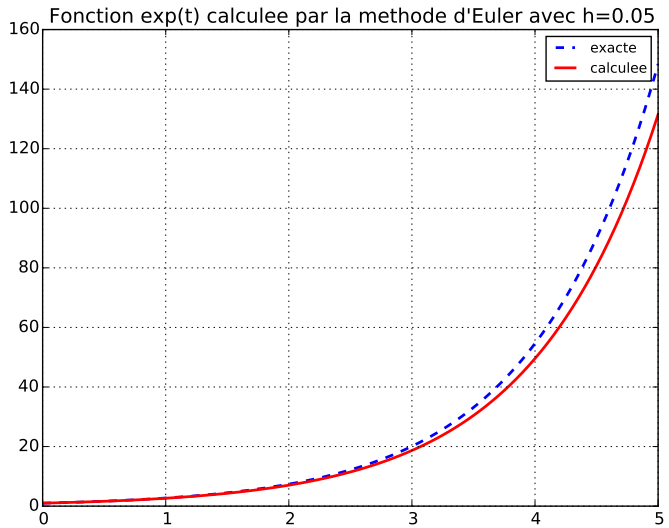
Erreur



- Erreur locale $e_i \sim \frac{1}{2} f''(x_i) h^2$.
- Erreur globale E qui provient de l'accumulation des erreurs des itérations précédentes.

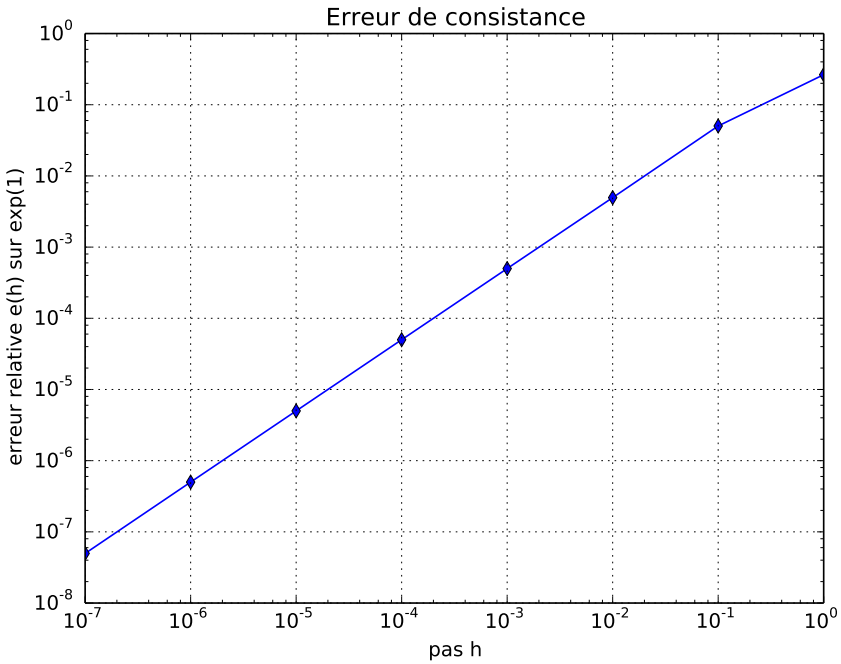
Influence du pas discrétisation





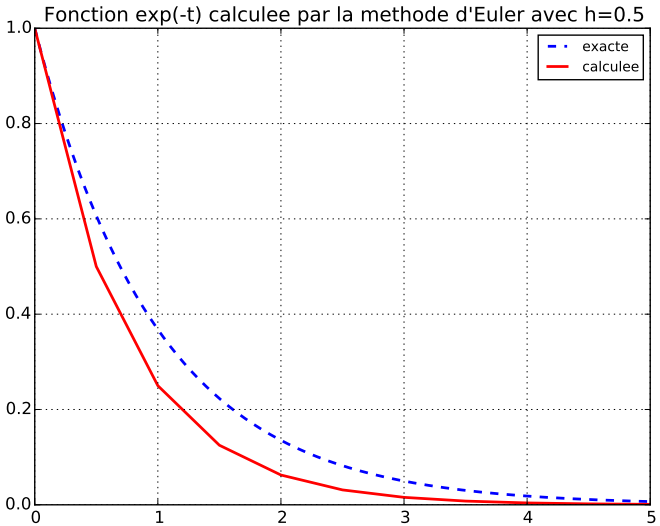
- Plus h est petit, plus la solution approchée sera consistante . . .
- . . . jusqu'à un certain point. En deçà les erreurs numériques explosent !
- **Complexité de la méthode d'Euler** : $O(n)$ avec $n = \dfrac{b-a}{h}$. Plus h est petit, plus le temps de calcul est élevé.
- Il faut trouver un compromis.

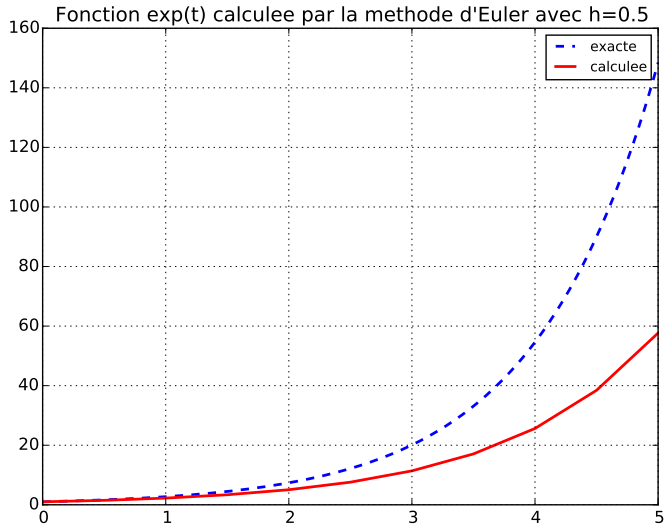
Consistance de la méthode d'Euler



Erreur relative e par calcul itératif pour différents pas h avec la méthode d'Euler : pour h petit, $\log(e) = \alpha \log(h) + b$ soit $e = k.h^{\alpha}$.

Influence du pas discrétisation





- toujours instable si l'EOD est instable elle-même ($y' - y = 0$) i.e l'erreur globale diverge.
- stable si l'EOD stable et si h suffisamment petit ($y' + y = 0$) i.e l'erreur globale est bornée.

Autres schémas numérique en bref

La méthode d'Euler n'est pas efficace et donc jamais utilisée en pratique. On peut définir d'autre schémas numérique dont certains seront abordés en TD. Par exemple

- Idée : pousser le développement en série de la fonction y à un ordre supérieur et/ou composer différents schémas numérique entre eux. Exemple :
  - Méthode d'Euler implicite : $\tilde{y}_{i+1} = \tilde{y}_i + f(\tilde{y}_{i+1}, t_{i+1})h$
  - Méthode d'Heun : ordre 2
  - Méthode de Runge-Kutta : ordre 4
- Méthode à pas adaptatif : le pas h_i est choisi d'autant plus petit que les variations locales de la fonction y sont importante i.e. que $\left|y'\right|$ est élevée.

Sommaire

{Equation différentielle d'ordre supérieur à 1

Exemple : chute verticale avec frottement : $\ddot{y} = -g - \lambda \dot{y}$.

Cette équation différentielle d'ordre 2 est équivalente au système d'équations différentielles d'ordre 1 :

$$\begin{cases} y' = y_p \\ y_p' = -g - \lambda y_p \end{cases}$$

On applique alors un schéma numérique, par exemple d'Euler, à chacune des équations du système pour évaluer $y_p(t) = y'(t)$ et $y(t)$.

Sommaire

Fonction prédéfinies en Python

Module `integrate` de la bibliotheque `scipy` :

```
import scipy.integrate as integ
```

Fonction `odeint`.

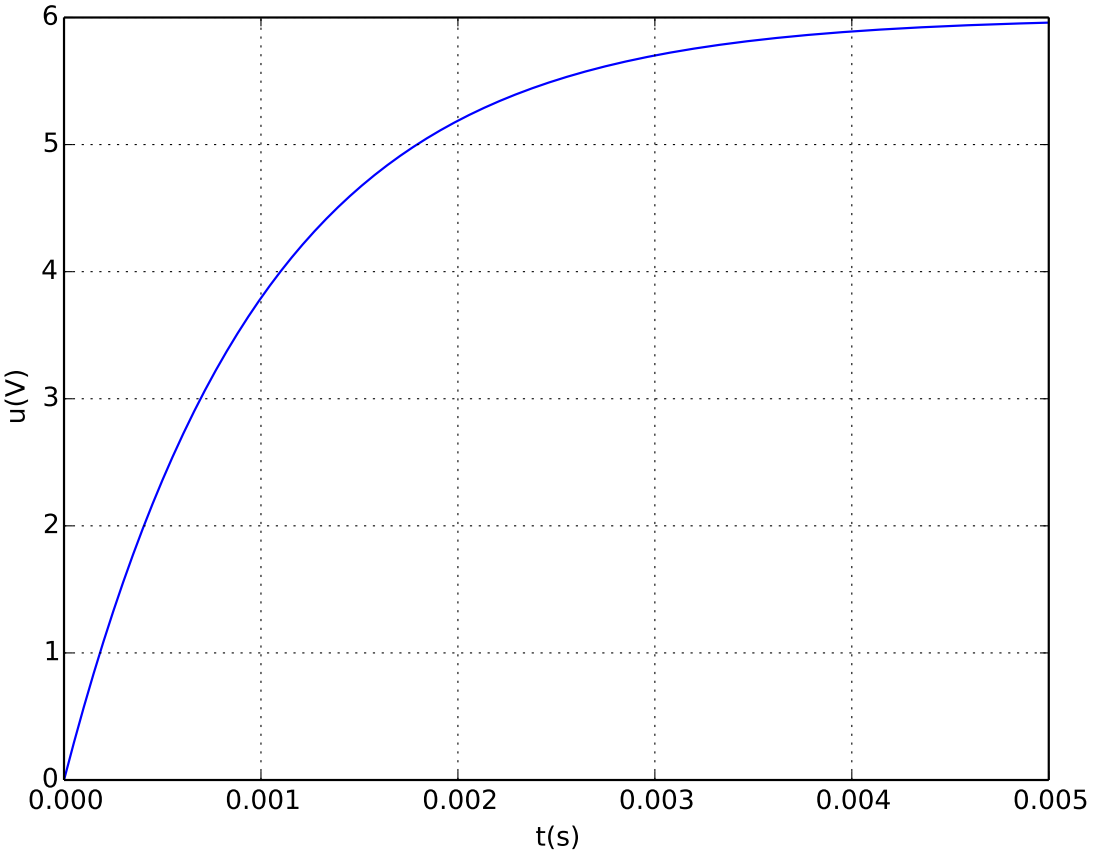
```
1 import scipy.integrate as integ
2 integ.odeint(f,init,t)
3 #f fonction telle que y'=f(y,t)
4 # t=[a...b], instants t auxquels y(t) est calculee
5 #y(a)=init
```

Charge d'un condensateur

$$\frac{dU}{dt} + \frac{u}{\tau} = \frac{E}{\tau}$$

```
1  C=1e-6
2  R=1e3
3  tau=R*C
4  E=6
5
6  def f(u,t):
7      rhs=-u/tau+E/tau
8      return rhs
9
10 u0=0
11 t=np.linspace(0,5*tau,1000)
12 u=integ.odeint(f,u0,t)
```

Charge d'un condensateur

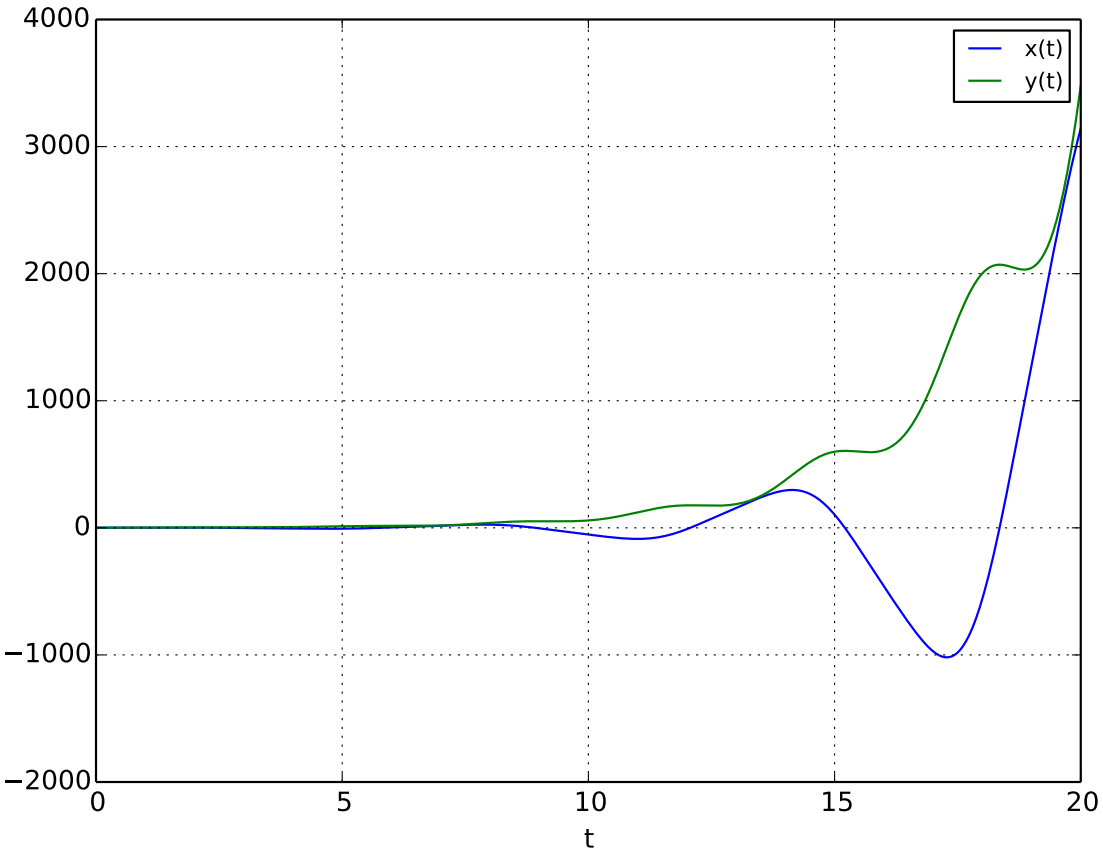


Système non linéaire

$$\begin{cases} x' = \cos(t)y \\ y' = \sin(t)x \end{cases}$$

```
1  def f(sol,t):
2      x=sol[0]
3      y=sol[1]
4
5      rhs_x=np.cos(t)*y
6      rhs_y=np.sin(t)*x
7      return [rhs_x,rhs_y]
8
9  init=[1,1]
10
11 t=np.linspace(0,20,1000000)
12 sol=integ.odeint(f,init,t)
13
14 print(sol)
```

Système non linéaire



Oscillateur de Van der Pol

$$\ddot{x} = \mu(1 - x^2)\dot{x} - x \Leftrightarrow \begin{cases} x' = x_p \\ x'_p = \mu(1 - x^2)x_p - x \end{cases}$$

```
1  def f(sol,t):
2      x=sol[0]
3      xp=sol[1]
4      rhs_x=xp
5      rhs_xp=mu*(1-x**2)*xp-x
6      return [rhs_x,rhs_xp]
7  #RESOLUTION
8  init=[0,0.1]
9  t=np.linspace(0,100,10000)
10 sol=integ.odeint(f,init,t)
11 x=sol[:,0]
12 xp=sol[:,1]
```

Oscillateur de Van der Pol