23

Systèmes d'équations différentielles

◇ On considère le problème de Cauchy

$$\begin{cases} \dfrac{\mathrm{d}x}{\mathrm{d}t} = -2x + y + 2t \\ \dfrac{\mathrm{d}y}{\mathrm{d}t} = 4x + y + 3\mathrm{e}^{-t} \\ x(0) = 1 \\ y(0) = 0 \end{cases}$$

Il s'écrit sous la forme $X' = F(X,t)$ en posant $X = \begin{bmatrix} x \\ y \end{bmatrix}$ et $F(X,t) = \begin{bmatrix} -2x+y+2t \\ 4x+y+3\mathrm{e}^{-t} \end{bmatrix}$.

◇ Dans les programmes PYTHON ou SCILAB, on écrit **xprim**, **yprim**, **Xprim** pour x', y', X'.

I. Avec Numpy/Scipy

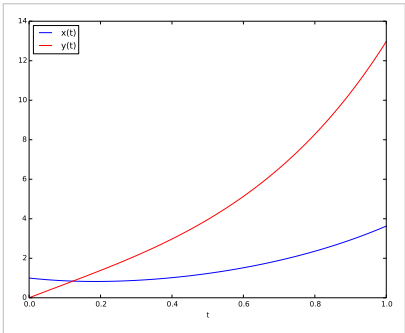
◇ On définit X comme un tableau (*array*) à une dimension dont les composantes sont x et y. De même, X' est le tableau dont les composantes sont x' et y'. C'est toujours la commande **odeint** qui est utilisée sous la forme **odeint(F,X0,t)** où **X0** est la condition initiale (ici **np.array([1,0])**).

```
from math import *
from scipy.integrate import odeint
import numpy as np
import matplotlib.pyplot as plt
def F(X,t):
    x=X[0]
    y=X[1]
    xprim=-2*x+y+2*t
    yprim=4*x+y+3*exp(-t)
    Xprim=np.array([xprim,yprim])
    return Xprim
t=np.linspace(0,1,100)
X=odeint(F,np.array([1,0]),t)
print t.shape, X.shape
```

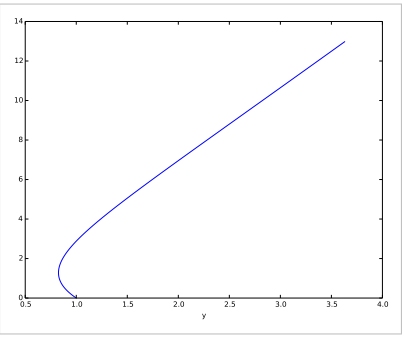
```
(100,) (100, 2)
```

Remarque. La première colonne de **X** donne les valeurs successives de x et la seconde colonne celles de y. On peut ainsi faire différents tracés. □

```
plt.plot(t,X[:,0],'b')
plt.plot(t,X[:,1],'r')
plt.show()
```



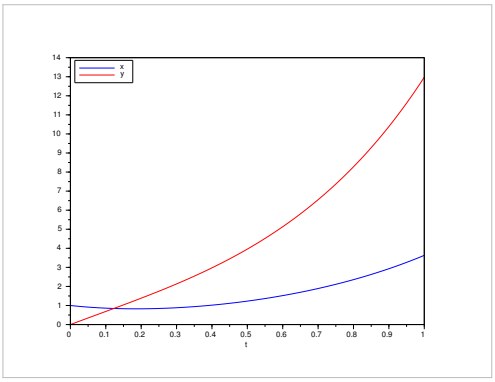

```
plt.plot(X[:,0],X[:,1])
plt.show()
```



II. Avec Scilab

◇ Le principe est le même en adaptant les notations.

```
function Xprim=F(t,X)
    x=X(1)
    y=X(2)
    xprim=-2*x+y+2*t
    yprim=4*x+y+3*exp(-t)
    Xprim=[xprim,yprim]
endfunction
t=linspace(0,1,100);
X=ode([1;0],0,t,F);
size(X)
 ans  =
    2.    100.
plot(t,X(1,:),'b');
plot(t,X(2,:),'r');
xlabel("t");
legend("x","y","in_upper_left");
```



⚠ *Remarques.*

- Avec NUMPY/SCIPY : la numérotation commence à 0 et on utilise **odeint(F,X0,t)** où **X0** est un *array* à 1 dimension de taille 2 et **F** est définie avec **def F(x,t)**. Le résultat **X** est un *array* de dimension $(n,2)$, n est la taille de **t**.
- Avec SCILAB : la numérotation commence à 1 et on utilise **ode(X0,t0,t,F)** où **X0** est une matrice de taille 2×1 et **F** est définie en écrivant **function Xprim=F(t,X)** etc. Le résultat **X** est une matrice de taille $(2,n)$ (la première ligne de **X** donne les valeurs de x et la deuxième les valeurs de y). □

Résolutions approchées de systèmes d'éq. diff.

```
from math import *
from scipy.integrate import odeint
import numpy as np
import matplotlib.pyplot as plt
```

1 Un exemple basique

Exemple. On considère le système d'équations différentielles :

$$\begin{cases} x'(t) = -3x + y + 1 + t \\ y'(t) = x - 3y + \mathrm{e}^{-2t} \end{cases}$$

Représenter graphiquement les solutions associées à la condition initiale $(0,1)$ à l'instant $t = 0$ sur l'intervalle $[0,2]$. Représenter ensuite la trajectoire de cette solution.

- La fonction F associée au système d'équations :

```
def F(X,t):
    x=X[0]
    y=X[1]
    xprim=-3*x+y+1+t
    yprim=x-3*y+exp(-2*t)
    return [xprim,yprim]
```

- Le vecteur t qui représente l'intervalle de temps considéré :

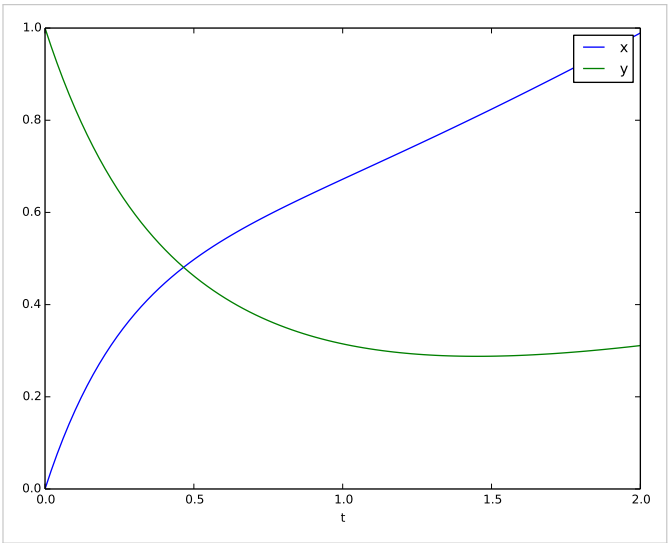
```
t=np.linspace(0,2,100)
```

- La résolution approchée avec **odeint** :

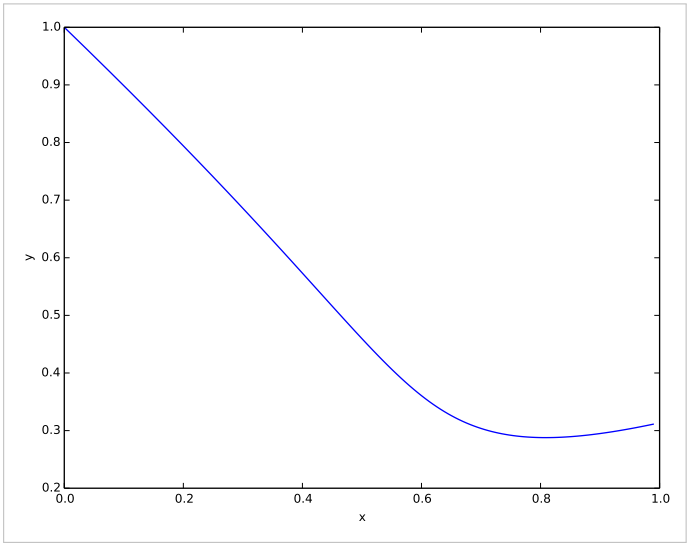
```
X=odeint(F,[0,1],t)
```

- Représentations graphiques :

```
plt.plot(t,X[:,0])
plt.plot(t,X[:,1])
plt.legend(["x","y"])
plt.xlabel("t")
```




```
plt.plot(X[:,0],X[:,1])
plt.xlabel("x")
plt.ylabel("y")
```



2 Le cas « autonome »

Exemple. On considère le système d'équations différentielles :

$$\begin{cases} x'(t) = x \times (1-y) \\ y'(t) = y \times (x-1) \end{cases}$$

Représenter graphiquement les trajectoires associées aux conditions initiales $(0.5, 0.5)$ et $(0.3, 0.3)$ sur l'intervalle $[0, 8]$.

- La fonction F associée au système d'équations :

```
def F(X,t):
    x=X[0]
    y=X[1]
    return [x*(1-y),y*(x-1)]
```

- Le vecteur t qui représente l'intervalle de temps considéré :

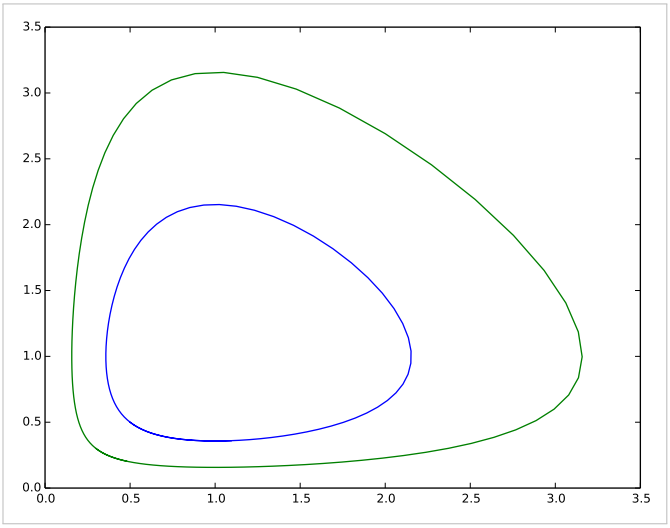
```
t=np.linspace(0,8,100)
```

- La résolution approchée avec **`odeint`** :

```
X1=odeint(F,[0.5,0.5],t)
X2=odeint(F,[0.3,0.3],t)
```

- Représentation graphique :

```
plt.clf()
plt.plot(X1[:,0],X1[:,1])
plt.plot(X2[:,0],X2[:,1])
```



3 Un système avec paramètres

Exemple. On considère un écosystème dans lequel cohabitent deux espèces : des proies et des prédateurs (pour l'exemple historique étudié par Volterra, des requins et des sardines). On note $x(t)$ la population de proies au temps t et $y(t)$ la population de prédateurs. On modélise l'évolution de ces populations par un système d'équations différentielles du type :

$$\begin{cases} x'(t) = ax(t) - px(t)y(t) \\ y'(t) = -by(t) + qx(t)y(t) \end{cases}$$

Les paramètres réels a, b, p, q sont à déterminer en fonction des populations étudiées (à l'aide, entre autres, d'études statistiques). Ces paramètres sont des réels strictement positifs.

(a) Représenter sur le même dessin les fonctions x et y pour la condition initiale $x(0) = 3$, $y(0) = 2$ et pour $(a, b, p, q) = (1, 0.1, 0.5, 0.2)$. Représenter également la trajectoire de cette solution. On travaillera sur l'intervalle $[0, 100]$.
(b) Faire de même avec $(a, b, p, q) = (0.8, 0.2, 0.5, 0.2)$.

- La fonction F associée au système d'équations :

```
def F(X,t,a,b,p,q):
    x=X[0]
    y=X[1]
    return [a*x-p*x*y,-b*y+q*x*y]
```

- Le vecteur t qui représente l'intervalle de temps considéré :

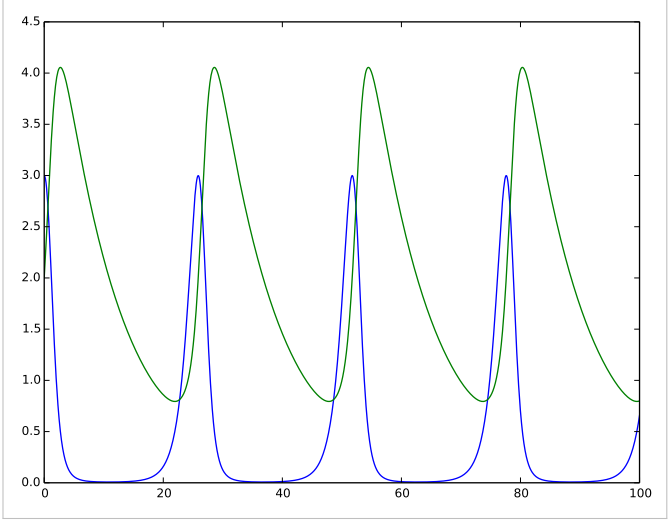
```
t=np.linspace(0,100,1000)
```

- La résolution approchée avec **odeint** :

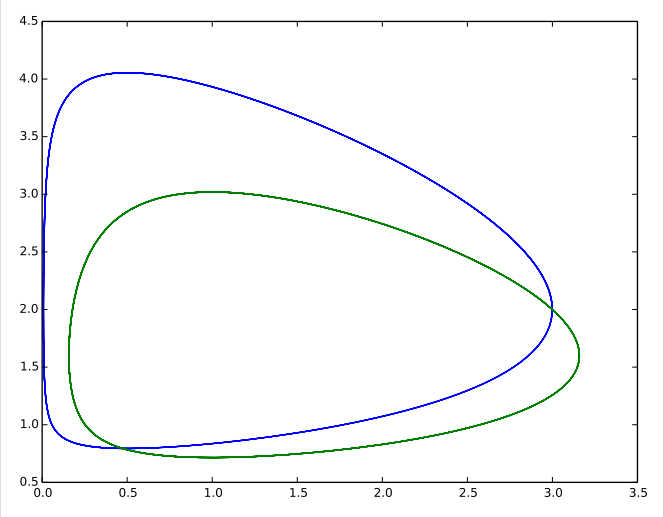
```
X1=odeint(F,[3,2],t,args=(1,0.1,0.5,0.2))
X2=odeint(F,[3,2],t,args=(0.8,0.2,0.5,0.2))
```

- Représentation graphique :

```
plt.clf()
plt.plot(t,X1[:,0])
plt.plot(t,X1[:,1])
```




```
plt.clf()
plt.plot(X1[:,0],X1[:,1])
plt.plot(X2[:,0],X2[:,1])
```



4 Application aux équations différentielles d'ordre 2

Exemple. On considère l'équation différentielle du second ordre :

$$x''(t) - (1 - x^2)x'(t) + x(t) = 0$$

On pose $y = x'$.

(a) Vérifier que (x, y) est solution d'un système de deux équations différentielles du premier ordre.

(b) Tracer les trajectoires pour les conditions initiales $(x(0), y(0)) = (1, 0)$ et $(x(0), y(0)) = (3, 0)$ sur l'intervalle $[0, 20]$.

- La fonction F associée au système d'équations :

```
def F(X,t):
    x=X[0]
    y=X[1]
    return [y,(1-x**2)*y-x]
```

- Le vecteur t qui représente l'intervalle de temps considéré :

```
t=np.linspace(0,20,200)
```

- La résolution approchée avec **`odeint`** :

```
X1=odeint(F,[1,0],t)
X2=odeint(F,[3,0],t)
```

- Représentation graphique :

```
plt.plot(X1[:,0],X1[:,1])
plt.plot(X2[:,0],X2[:,1])
```