SYSTÈMES DIFFÉRENTIELS LINÉAIRES À COEFFICIENTS CONSTANTS

PYTHON
COURS ET EXERCICES

1 – AFFICHER LES TRAJECTOIRES D'UN SYSTÈME DIFFÉRENTIEL, FAIRE VARIER LES CONDITIONS INITIALES :

On considère les trajectoires suivantes provenant de systèmes différentiels. Le but est de les représenter en faisant varier les conditions initiales. Pour cela nous allons faire varier les constantes.

$$(1)\ X(t)=\begin{pmatrix} C_1e^{-4t}-3C_2e^{2t} \\ 2C_1e^{-4t}+C_2e^{2t}\end{pmatrix} (2)\ X(t)=\begin{pmatrix} C_1e^{-0,5t}-C_2 \\ C_1e^{-0,5t}+2C_2\end{pmatrix} (3)\ X(t)=\begin{pmatrix} -7C_1e^{-0,5t}+C_2e^{-0,4t} \\ 2C_1e^{-0,5t}+C_2e^{-0,4t}\end{pmatrix}$$

$$(4)\begin{pmatrix} 4C_1e^{t}+5C_2e^{2t}+C_3 \\ C_1e^{t}-6C_2e^{2t}+2C_3 \\ 2C_1e^{t}+2C_2e^{2t}-4C_3\end{pmatrix}; (5)\begin{pmatrix} 4C_1e^{-t}+5C_2e^{-2t}-C_3 \\ C_1e^{-t}-6C_2e^{-2t}+5C_3 \\ 2C_1e^{-t}+2C_2e^{-2t}+2C_3\end{pmatrix}; (6)\begin{pmatrix} 4C_1e^{t}+5C_2e^{-2t}-2C_3 \\ C_1e^{t}-6C_2e^{-2t}-C_3 \\ 2C_1e^{t}+2C_2e^{-2t}+3C_3\end{pmatrix}; (7)\begin{pmatrix} 4C_1e^{-t}+5C_2e^{-2t}-2C_3e^{-3t} \\ C_1e^{-t}-6C_2e^{-2t}-C_3e^{-3t} \\ 2C_1e^{-t}+2C_2e^{-2t}+3C_3e^{-3t}\end{pmatrix}$$

1) Voici un script Python permettant d'afficher les trajectoires du premier système différentiel du (1) :

```
import numpy as np
import matplotlib.pyplot as plt

t=np.linspace(0,50,1000)  #la paramètre est t. On créé une liste de paramètres.

def trajectoire(a,b) : #la fonction trajectoire dépend des constantes, définies par les
                       # conditions initiales
      x= a*np.exp(-4*t)-3*b*np.exp(2*t)#x est l'abscisse de la trajectoire
      y=2*a*np.exp(-4*t)+b*np.exp(2*t) # y est l'ordonnée de la trajectoire
      return (x,y)              #renvoie les deux valeurs de x et y en fonction de t
for a in np.linspace (-5,5,20) : #on génère 20 valeurs de a entre -5 et 5
      for b in np.linspace (-5,5,20) :  #on génère 20 valeurs de b entre -5 et 5
            (x,y)=trajectoire(a,b) #on créé des points(x,y) définis par les équations du
                                   #système et en fonction de t
            plt.plot(x,y)       #on affiche ces points
plt.show()                      #on montre la courbe.
```

a) Pour chaque système du 1), déterminer quelles trajectoires amènent à un point d'équilibre, quels sont les points d'équilibre et déterminer si ces équilibres sont stables ou pas ?
b) Tester le script ci-dessus pour afficher les trajectoires.
c) Dans une nouvelle fenêtre Python, modifier si besoin le script pour afficher les trajectoires amenant aux points d'équilibre ? Comparer.

2) Voici un script Python permettant d'afficher les trajectoires du premier système différentiel du (2) (exercice 9 de la feuille d'exercices). Si l'on veut faire varier les conditions initiales, on insère des boucles for mais attention, Edupython n'affichera pas toutes les trajectoires dans la même fenêtre. Il va ouvrir autant de fenêtres graphiques que de conditions (12 fenêtres pour l'exemple ci-dessous). Donc, il ne faut pas générer trop de conditions.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D       #bibliothèque de tracé 3D de mathpoltlib

t=np.linspace(0,50,1000)         #la variable est t. On créé une liste de paramètres.

def trajectoire(a,b,c) : #la fonction trajectoire dépend des constantes, définies par les
                         # conditions initiales
      x= 4*a*np.exp(t)+5*b*np.exp(2*t)+c #x est l'abscisse de la trajectoire
      y=a*np.exp(t)-6*b*np.exp(2*t)+2*c      # y est l'ordonnée de la trajectoire
      z=2*a*np.exp(t)+2*b*np.exp(2*t)-4*c    # z est la côte de la trajectoire
      return (x,y,z)            #renvoie les trois valeurs de x, y et z en fonction de t
```

```
for a in np.linspace (-5,5,2) : #on génère 2 valeurs de a entre -5 et 5
      for b in np.linspace (-5,5,2) :  #on génère 2 valeurs de b entre -5 et 5
            for c in np.linspace (-5,5,3) :  #on génère 3 valeurs de c entre -5 et 5
            (x,y,z)=trajectoire(a,b,c) #on créé des points(x,y,z) définis par les équations
                                       #du système et en fonction de t
            fig=plt.figure()           #récupération de la figure dans une variable
            ax=fig.gca(projection='3d')      #affichage en 3D
            ax.plot(x,y,z)      #on affiche les points de Y=(x,y,z) en fonction de t
plt.show()                      #on montre la courbe.
```

a) Pour chaque système du 1), déterminer quelles trajectoires amènent à un point d'équilibre, quels sont les points d'équilibre et déterminer si ces équilibres sont stables ou pas ?
b) Tester le script ci-dessus pour afficher les trajectoires.
c) Dans une nouvelle fenêtre Python, modifier si besoin le script pour afficher les trajectoires amenant aux points d'équilibre ? Comparer.

2 – Résoudre une équation différentielle et afficher la trajectoire, faire varier les conditions initiales :

La résolution des équations différentielles ou des systèmes se fait avec la commande odeint de la bibliothèque scipy.integrate. Cette fonction n'est pas exigible au programme. La syntaxe est odeint (F, a, t) où F est une fonction qui représente l'équation ou le système différentiel, a représente les conditions initiales et t est la variable des fonctions solutions.

Si l'on résout l'équation différentielle $y' = 0,5y$ sur $\mathbb{R}$, on obtient la famille de solutions $y(t) = Ce^{-0,5t}$ où $C \in \mathbb{R}$. En particulier, avec la condition initiale $y(0) = 1$, la solution unique est donnée par la fonction définie sur $\mathbb{R}$ par $y(t) = e^{0,5t}$.

1) Voici un script Python permettant de résoudre l'équation différentielle avec la condition initiale et qui affiche la trajectoire.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
t=np.linspace(0,50,1000)  #la variable est t. On créé une liste d'abscisses. Plus il y a de
                           # points, plus la résolution sera précise.
def equation (Y, t) :     #l'équation différentielle est créée sous la forme d'une fonction
      return 0.5*Y
Y=odeint(equation, 1, t)  #résolution de l'équation différentielle
plt.plot(t,Y)       #on affiche le résultat de Y en fonction de t
plt.show()          #on montre la courbe.
```

2) Comment modifier ce script pour afficher une famille de trajectoires lorsqu'on fait varier les conditions initiales (20 conditions entre $y(0) = -5$ et $y(0) = 5$) .

Il faut faire varier les conditions initiales. Pour cela nous créons une liste de valeurs et nous utilisons une boucle pour afficher les courbes selon ces différentes conditions initiales.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
t=np.linspace(0,50,1000)
def equation (Y, t) :
      return 0.5*Y
for y_0 in np.linspace(-5,5,20) :       #on génère 20 valeurs y(0) entre -5 et 5
      Y=odeint(equation, y_0, t)        #les conditions initiales y(0) varient avec les valeurs
                                        #de y_0
      plt.plot(t,Y)
plt.show()
```

3 – Représenter une trajectoire de solution d'un système à deux équations ou à trois équations, les conditions initiales étant données :

1) On considère le système différentiel suivant défini sur $\mathbb{R}$ (S) : $\begin{cases} x' = -3x + y \\ y' = x - 3y \end{cases}$ avec les conditions initiales $x(0) = 1$ et $y(0) = 1$. Voici un script Python qui affichera la trajectoire de la solution de ce système.

(On rappelle que les solutions d'un tel système sont données par $X(t) = \begin{pmatrix} C_1 e^{-4t} + C_2 e^{-2t} \\ -C_1 e^{-4t} + C_2 e^{-2t} \end{pmatrix}$ où $(C_1, C_2) \in \mathbb{R}^2$ et $t \in \mathbb{R}$)

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
t=np.linspace(0,50,1000)  #la variable est t. On créé une liste de paramètres.
def systeme(Y, t) : #le système différentiel est créé sous la forme d'une fonction vecteur
       x=Y[0] ; y=Y[1]     #x est la première composante et y la deuxième composante de Y
       return ([-3*x+y,x-3*y])    #renvoie un vecteur formé par les équations du système
Y_0=[1,1]                         #Y_0 contient le vecteur [x(0),y(0)]des conditions initiales
Y=odeint(systeme, Y_0, t)         # résolution du système différentiel
x=Y[ :,0]    #on récupère les données, x récupére la valeur de la première composante de Y
y=Y[ :,1]                  #y récupère la valeur de la deuxième composante de Y.
plt.plot(x,y)              #on affiche le résultat de Y=(x,y) en fonction de t
plt.show()                 #on montre la courbe.
```

2) Comment modifier ce script pour afficher une famille de trajectoires lorsqu'on fait varier les conditions initiales ?

a) 20 valeurs -5 et 5 pour $x(0)$ et 20 valeurs entre -5 et 5 pour $y(0)$.

b) 20 valeurs -5 et 5 pour $x(0)$ et 20 valeurs entre -5 et 0 pour $y(0)$.

c) 20 valeurs -5 et 0 pour $x(0)$ et 20 valeurs entre -5 et 5 pour $y(0)$.

c) 20 valeurs 0 et 5 pour $x(0)$ et 20 valeurs entre 0 et 5 pour $y(0)$.

Le principe est identique à l'exercice précédent. Il créer deux boucles for et créer une initialisation variable.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
t=np.linspace(0,50,1000)
def systeme(Y, t) :
       x=Y[0] ; y=Y[1]
       return ([-3*x+y,x-3*y])

for r in np.linspace(-5,5,20) :         #on génère 20 valeurs x(0) entre -5 et 5
    for s in np.linspace(-5,5,20) :     #on génère 20 valeurs y(0) entre -5 et 5
       Y_0=[r,s]                    #le vecteur Y_0 est initialisé par[r,s]
       Y=odeint(systeme, Y_0, t) #les conditions initiales Y_0=[x(0),y(0)]varient avec les
                                    # valeurs de [r,s]
       x=Y[:,0]
       y=Y[:,1]
       plt.plot(x,y)
plt.show()
```

3) Voici un exemple de script précédent pour afficher la trajectoire du système différentiel suivant défini sur $\mathbb{R}$ (S) :

$\begin{cases} x' = -0,1x \\ y' = -0,5y \\ z' = -0,5z \end{cases}$ avec les conditions initiales $x(0) = 1,\ y(0) = 1$ et $z(0) = 1$.

(On rappelle que les solutions d'un tel système sont données par $X(t) = \begin{pmatrix} e^{-0,1t} \\ e^{-0,5t} \\ e^{-0,5t} \end{pmatrix}$ où $(C_1, C_2, C_3) \in \mathbb{R}^3$ et $t \in \mathbb{R}$)

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
from mpl_toolkits.mplot3d import Axes3D       #bibliothèque de tracé 3D de mathpoltlib
t=np.linspace(0,50,1000)         #la variable est t. On créé une liste de paramètres.
def systeme(Y, t) : #le système différentiel est créé sous la forme d'une fonction vecteur
       x=Y[0] ; y=Y[1] ; z=Y[2]  #x est la première composante, y la deuxième composante et z
                                        #la troisième composante de Y
       return ([-0.1*x,-0.5*y,-0.5*z]) #renvoie un vecteur formé par les équations du système
Y_0=[1,1,1]         #Y_0 contient le vecteur [x(0),y(0), z(0)]des conditions initiales
Y=odeint(systeme, Y_0, t) # résolution du système différentiel
x=Y[ :,0] # on récupère les données, x récupère la valeur de la première composante de Y
y=Y[ :,1] #y récupère la valeur de la deuxième composante de Y
z=Y[ :,2]    #z récupère la valeur de la troisième composante de Y.
fig=plt.figure()    #récupération de la figure dans une variable
ax=fig.gca(projection='3d')      #affichage en 3D
ax.plot(x,y,z)      #on affiche les points Y=(x,y,z) en fonction de t
plt.show()          #on montre la courbe.
```

Si l'on veut faire varier les conditions initiales, on insère des boucles for mais attention, Edupython n'affichera pas toutes les trajectoires dans la même fenêtre. Il va ouvrir autant de fenêtres graphiques que de conditions (12 fenêtres pour l'exemple ci-dessous). Donc, il ne faut pas générer trop de conditions.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
from mpl_toolkits.mplot3d import Axes3D

t=np.linspace(0,50,1000)
def systeme(Y, t) :
       x=Y[0] ; y=Y[1] ; z=Y[2]
       return ([-0.1*x,-0.5*y,-0.5*z])

for r in np.linspace(-5,5,2) :          #on génère 2 valeurs de a entre -5 et 5
    for s in np.linspace(-5,5,2) :      #on génère 2 valeurs de b entre -5 et 5
        for q in np.linspace(-5,5,3) : #on génère 3 valeurs de b entre -5 et 5
            Y_0=[r,s,q]
            Y=odeint(systeme, Y_0, t)
            x=Y[ :,0]
            y=Y[ :,1]
            z=Y[ :,2]
            fig=plt.figure()
            ax=fig.gca(projection='3d')
            ax.plot(x,y,z)
plt.show()
```