Devoir libre 06

CORRECTION EXERCICE 01

Exercice 01 (avec Python)

Inspiré des oraux de CCS Maths II

Partie A

On s'intéresse à l'équation différentielle : $y'(t) = t^2 - y^2(t)$, avec $y(1.5) = a$.

1) *Rappel de la méthode d'Euler*

On cherche à déterminer une solution approchée de $y'(t) = f(y(t), t)$, valable sur un intervalle donné, c'est-à-dire $t \in [t_0, t_n]$. On suppose la condition initiale $y(t_0) = y_0$. Pour cela, on choisit une subdivision $(t_0, t_1, ..., t_n)$ de $[t_0, t_n]$ à pas constant $h = (t_n - t_0)/n$. Donc :

$$\forall k \in [\![0, n-1]\!],\ t_{k+1} = t_k + h.$$

On définit alors la liste $(y_1, ..., y_n)$ telle que $\forall k \in [\![0, n-1]\!],\ y_{k+1} = hf(y_k, t_k) + y_k$.
La méthode d'Euler consiste à considérer que si h est petit alors $y(t_k + h)$ est proche de $y(t_k) + hy'(t_k)$, c'est-à-dire de $y(t_k) + hf(y(t_k), t_k)$. Ainsi, pour connaître $y(t_k + h) = y(t_{k+1})$, on doit partir de t_k et de $y(t_k)$ et on suppose qu'entre les points $M_k(t_k, y(t_k))$ et $M_{k+1}(t_{k+1}, y(t_{k+1}))$, la courbe représentative de l'unique solution y reste « proche » de sa tangente au point t_k.

Élaboration d'une procédure Python utilisant l'algorithme d'Euler

Écrivons une fonction *Euler* qui prend en argument f, $t0$, tn, n et $y0$. On créera dans une boucle la liste T des abscisses et celle Y des valeurs prises par la solution y aux points considérés.
On renverra Y.

```
>>> def Euler(f,t0,tn,n,y0) :
        h = (tn-t0)/float(n) ; t = t0 ; y = y0
        T = [t0] ; Y = [y0]
        for k in range(n) :
            y = y + h*f(y,t) ; t = t + h ; T.append(t) ; Y.append(y)
        return Y
```

Terminons par un exemple.
Appliquons à $(t+1)y'(t) + y(t) = \cos t$ pour $t \in [0, 10]$ avec $y(0) = 1$ et $n = 10$. On tape alors (*numpy* est importé en prévision de l'apparition de cos dans l'exemple) :

```
>>> import numpy as np
>>> def f(x,t) : return (np.cos(t) - x)/(t+1)
>>> Euler(f,0,10,10,1)
```

```
array([1. , 1. , 0.77015115 , 0.37471849 , 0.03354074 ,
-0.10389613 , -0.03930308 , 0.10347883 , 0.1847176 , 0.1480808378, 0.04216238])
```

Procédure affichant les courbes intégrales demandées par la méthode d'Euler

L'idée est d'écrire une fonction *Euler_Affich* de mêmes arguments qu'Euler et qui doit renvoyer l'affichage de la courbe reliant les points calculés.

On commence par taper les packages utiles. Puis la fonction *EulerAffich* et enfin on tape f (issue de l'équation différentielle de l'énoncé).

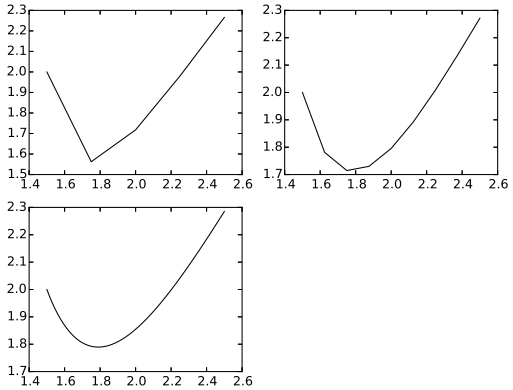
```
>>> import numpy as np ; import matplotlib.pyplot as plt
>>> import scipy.integrate as integr
>>> def EulerAffich(f,t0,tn,n,y0) :
        h = (tn-t0)/float(n) ; t = t0 ; y = y0
        T = [t0] ; Y = [y0]
        for k in range(n) :
            y = y + h*f(y,t) ; t = t + h
            T.append(t) ; Y.append(y)
        plt.plot(T, Y, color =' 0' )
```

$>>>$ *def* $f(x,t)$: *return*$(t**2 - x**2)$
$>>>$ TT $=$ *np.arange*$(1.5, 2.5, 0.00001)$; YY $=$ *integr.odeint*$(f, 2, TT)$

Pour l'utilisation de *odeint*, voir la remarque plus loin.
Par commodité, pour gagner de la place, on utilise *plt.subplot*. Ce n'est pas obligatoire pour vous. Et même, vous n'y avez pas pensé !

$>>>$ *plt.subplot*(221) ; *EulerAffich*$(f, 1.5, 2.5, 4, 2)$; *plt.subplot*(222) ;
EulerAffich$(f, 1.5, 2.5, 8, 2)$; *plt.subplot*(223) ; *plt.plot*(TT, YY)

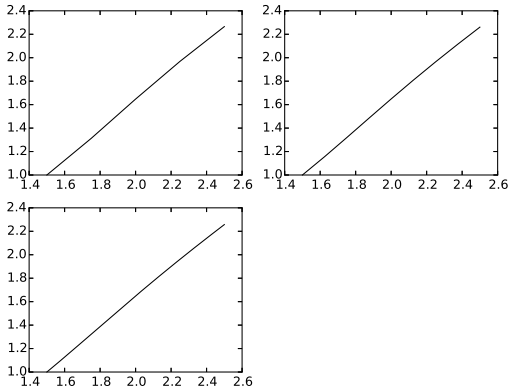
Le premier graphe en haut à gauche est *Euler* avec $h = 1/4$, puis le second en haut à droite est *Euler* avec $h = 1/8$ et celui du bas à gauche est *odeint* avec $h = 0.00001$.



2) Le cas $a = 1$ se fait sans problème car on a déjà fait le code. On remarquera que la courbe semble ici linéaire. Attention, on remplace YY par YYY (pour changer a dans *odeint*). On tape :

$>>>$ TT $=$ *np.arange*$(1.5, 2.5, 0.00001)$; YYY $=$ *integr.odeint*$(f, 1, TT)$
$>>>$ *plt.subplot*(221) ; *EulerAffich*$(f, 1.5, 2.5, 4, 1)$; *plt.subplot*(222) ;
EulerAffich$(f, 1.5, 2.5, 8, 1)$; *plt.subplot*(223) ; *plt.plot*(TT, YYY)

Le premier graphe en haut à gauche est *Euler* avec $h = 1/4$, puis le second en haut à droite est *Euler* avec $h = 1/8$ et celui du bas à gauche est *odeint* avec $h = 0.00001$.



• *Remarque*

Il faut se souvenir de la manière d'utiliser la fonction *odeint* du sous-module *scipy.integrate*.
Cette fonction nécessite une liste de valeurs de t, commençant en t_0 et une condition initiale y_0. La fonction *odeint* renvoie des valeurs approchées (aux points contenus dans la liste des valeurs de t) de la solution y de l'équation différentielle qui vérifie $y(t_0) = y_0$. Plus précisément, si l'on désire les solutions pour $t \in [t_0, t_n]$ avec un pas h, on tape :

$>>>$ *import scipy.integrate as integr* ; *def* $f(y,t)$: *return expression*
$>>>$ T $=$ *np.arange*$(t0, tn, h)$; Y $=$ *integr.odeint*$(f, y0, T)$

Y renvoyé est un tableau dont chaque ligne correspond à la valeur d'une fonction à un instant donné. Ainsi $Y[0]$ renvoie y_0 et $Y[-1]$ renvoie $y(t_1)$.

On tape alors :
$>>>$ $plt.plot(T, Y)\,;\ plt.show(\,)$
On a le tracé de la courbe intégrale.

Partie B

On considère maintenant pour $n \in \mathbb{N}$, (E_n) : $y''(t) - 11\,y'(t) + 28\,y(t) = \sin(nt)$, dont les conditions initiales sont $y(0) = 0$ et $y'(0) = 1$.

1)a) On résout l'équation caractéristique $x^2 - 11x + 28 = 0$. On trouve $\{4, 7\}$.
Donc les solutions de (E_0) sont les fonctions de la forme $t \mapsto ae^{4t} + be^{7t}$.

1)b) On pose : $X(t) = \begin{pmatrix} y(t) \\ y'(t) \end{pmatrix}$ et $A = \begin{pmatrix} 0 & 1 \\ -28 & 11 \end{pmatrix}$.
(Si vous etes un afficionado de Python, on peut utiliser aussi *alg.eig* de *numpy.linalg* ou mieux ici $A.eigenvects(\,)$ de *numpy* pour trouver les valeurs propres et des vecteurs propres de A.)
L'équation (E_0) est équivalente au système différentiel $X' = AX(t)$.
Le polynôme caractéristique de A est :
$\chi_A(t) = \mathsf{Det}\,(tI_2 - A) = t^2 - 11t + 28 = (t-4)(t-7)$.
L'espace propre $E_4(A)$ associé à la valeur propre 4 a pour base $(1, 4)$ et l'espace propre $E_7(A)$ associé à la valeur propre 7 a pour base $(1, 7)$.

La matrice de passage de la base canonique à la base de vecteurs propres est $P = \begin{pmatrix} 1 & 1 \\ 4 & 7 \end{pmatrix}$.

On a alors : $\begin{pmatrix} y(t) \\ y'(t) \end{pmatrix} = P \begin{pmatrix} a\,e^{4t} \\ b\,e^{7t} \end{pmatrix}$. On retrouve le cas précédent.

2) On a déjà les solutions de l'équation homogène. Comme $i\,n$ n'est pas solution de l'équation caractéristique, on cherche d'après le cours une solution particulière de la forme $y_p(t) = \lambda\,\cos(nt) + \mu\,\sin(nt)$.
On a : $y_p'(t) = -\lambda\,n\,\sin(nt) + \mu\,n\,\cos(nt)$ et $y_p''(t) = -\lambda\,n^2\,\cos(nt) - \mu\,n^2\,\sin(nt)$.
Donc :

$$\begin{gathered} y_p''(t) - 11y_p'(t) + 28y_p(t) \\ = (-\lambda n^2 - 11n\mu + 28\lambda)\cos nt + (-n^2\mu - 11n\lambda + 28\mu)\sin nt. \end{gathered}$$

Cela donne le système et sa résolution :

$$\left\{ \begin{array}{rcl} (28-n^2)\lambda - 11n\mu & = & 0 \\ -11n\lambda + (28-n^2)\mu & = & 1 \end{array} \right. \Rightarrow \lambda = \frac{11n}{n^4+65n^2+784} \text{ et } \mu = \frac{28-n^2}{n^4+65n^2+784}.$$

Puis : $\left\{ \begin{array}{rcl} a+b+\lambda & = & 0 \\ 4a+7b+n\mu & = & 1 \end{array} \right. \Rightarrow a = -\frac{1}{3} - \frac{7}{3}\lambda + \frac{n}{3}\mu$ et $b = \frac{1}{3} + \frac{4}{3}\lambda - \frac{n}{3}\mu$.
Et, toujours après calculs :

$$a = \frac{-n^4 - n^3 - 65n^2 - 49n - 784}{3(n^4+65n^2+784)} \text{ et } b = \frac{n^4 + n^3 + 65n^2 + 16n + 784}{3(n^4+65n^2+784)}.$$

On peut laisser comme cela mais la version Python montre que ces coefficients se simplifient. En effet, $n^4 + 65n^2 + 784 = (n^2+16)(n^2+49)$ et le numérateur de a est $-(n^2+n+16)(n^2+49)$ et celui de b est $(n^2+n+49)(n^2+16)$.
Donc, après simplifications : $a = \frac{n^2+n+16}{-3(n^2+16)}$ et $b = \frac{n^2+n+49}{3(n^2+49)}$.
Il reste à sommer cette solution particulière et la solution générale de (E_0) puis à déterminer les constantes a et b avec les conditions initiales.

3) On tape :

$>>>$ $from\ sympy\ import\ *$
$>>>$ $y = Function('y')\,;\ t = symbols('t')\,;\ n = symbols('n')$
$>>>$ $eq = dsolve(Derivative(y(t),t,t) - 11*Derivative(y(t),t) + 28*y(t) - sin(n*t), y(t))$
$>>>$ eq
$y(t) == C1*exp(4*t) + C2*exp(7*t) + 11*n*cos(n*t)/(n**4 + 65*n**2 + 784)$
$+(-n**2/(n**4 + 65*n**2 + 784) + 28/(n**4 + 65*n**2 + 784))*sin(n*t)$
$>>>$ $t0 = 0\,;\ y0 = 0\,;\ y1 = 1$
$>>>$ $eq1 = (eq.rhs).subs(t, t0) - y0\,;\ eq2 = diff(eq.rhs, t).subs(t, t0) - y1$
$>>>$ $C1 = symbols('C1')\,;\ C2 = symbols('C2')$

```
>>> Sol = solve([eq1, eq2], [C1, C2]) ; eq.subs(Sol)
y(t) == 11 * n * cos(n * t)/(n * *4 + 65 * n * *2 + 784)
+(−n * *2/(n * *4 + 65 * n * *2 + 784) + 28/(n * *4 + 65 * n * *2 + 784)) * sin(n * t)
+(n * *2 + n + 49) * exp(7 * t)/(3 * (n * *2 + 49)) − (n * *2 + n + 16)
*exp(4 * t)/(3 * n * *2 + 48)
```

4) On tape successivement :

```
>>> import numpy as np ; import matplotlib.pyplot as plt
>>> import scipy.integrate as integr
>>> def F(x, t, n) :
        return(np.array([x[1], −28 * x[0] + 11 * x[1] + np.sin(n * t)]))

>>> T = np.linspace(0, 0.5, 200)
>>> for n in range(11) :
        X = integr.odeint(F, np.array([0, 1]), T, (n, ))
        plt.plot(T , X[:, 0] , color = '0' )

>>> plt.show( )
```