TD Informatique TSI2

Differential equations and systems

Paragraph I

Resolution differentiels equations order 1

On commence par écrire l'équation différentielle sous la forme $y'(t) = f(y(t), t)$ et la condition initiale $y(t_0) = y_0$, ce qui permet de généraliser à une équation non linéaire, c'est-à-dire dans le cas où $f(y(t), t)$ n'est pas de la forme $a(t)y(t) + b(t)$.

1. Utilisation de `odeint`

On peut 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 selon le modèle suivant :

```
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.

Exemple : *déterminons sur* $[0, 1]$ *la solution de* $y'(t) = t\,y(t) + t$ *qui vérifie* $y(0) = 1$ *en utilisant* ***`odeint`***

```
In [1]: import matplotlib.pyplot as plt
In [2]: def f(x,t) :
            return(x*t+t)
In [3]: import scipy.integrate as integr; import numpy as np
In [4]: T = np.linspace(0,1,100);Y = integr.odeint(f,1,T);plt.plot(T,Y)
```

2. Utilisation 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 $\forall k \in [\![0, n-1]\!]$, $y_{k+1} = hf(y_k, t_k) + y_k$.

Explication : 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.

On tape alors 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.

```
In [1]: 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
```

T.S.V.P →

Exemple : *déterminons sur* $[0,1]$ *la solution de* $y'(t) = t\,y(t) + t$ *qui vérifie* $y(0) = 1$ *en utilisant* ***Euler*** *avec* ***n=100***

```
In [1]: def f(x,t) :
            return(x*t+t)
In [2]: Euler(f,0,1,100,1)
```

On va écrire maintenant une fonction `Euler_Affich` de mêmes arguments, qui doit renvoyer l'affichage de la courbe reliant les points calculés. En notant Δt et Δy les amplitudes des valeurs manipulées, on créera une fenêtre graphique qui déborde de 10% de chaque côté du tracé.

```
In [1]: import matplotlib.pyplot as plt
In [2]: def Euler_Affich(f,t0,tn,n,y0) :
            h = (tn-t0)/float(n) ; ymin = y0; ymax = y0
            t = t0 ; y = y0;  T = [t0]; Y = [y0]
            for k in range(n) :
               y = y + h*f(y, t); t = t + h; ymin = min(ymin,y)
               ymax = max(ymax,y); T.append(t); Y.append(y)
            deltaT = (tn-t0)/10; deltaY = (ymax-ymin)/10
            plt.plot(T, Y, color = 'b'); plt.grid()
            plt.axis([t0-deltaT,tn+deltaT,ymin-deltaY,ymax+deltaY])
            plt.axhline(color=' 0 ');plt.axvline(color = ' 0 ')
            plt.show()
```

Exemple : *Afficher sur* $[0,1]$ *la courbe intégrale de la solution de* $y'(t) = t\,y(t) + t$ *qui vérifie* $y(0) = 1$ *en utilisant* ***Euler_Affich*** *avec* ***n=100*** *et comparer avec la courbe obtenue avec* ***odeint***

Paragraph II

Resolution differentiels systems order 1

1. Utilisation de `odeint`

Nous allons traiter le cas du système différentiel $\left\{ \begin{array}{rcl} x'(t) & = & a\,x(t) + b\,y(t) \\ y'(t) & = & c\,x(t) + d\,y(t) \end{array} \right.$ avec la condition initiale $x(t_0) = x_0$ et $y(t_0) = y_0$. Ici a, b, c et d sont bien entendu constantes. On cherche une solution $t \mapsto (x(t), y(t))$ pour $t \in [t_0, t_n]$ de pas h.

On tape :

```
import numpy as np
import scipy.integrate as integr
def F(x,t) : return np.array([a*x[0]+b*x[1], c*x[0]+ d*x[1]])
T = np.arange(t0,tn,h)
X = integr.odeint(F,np.array([x0,y0]),T)
```

`X` est un tableau de deux colonnes. Chaque $k^{\tilde{a}¨{\rm me}}$ ligne correspond à $x(tk)$ et $y(tk)$.
Ainsi, `X[:,0]` (*resp.* `X[:,1]`) donne la fonction $t \mapsto x(t)$ (*resp.* $t \mapsto y(t)$).
Pour afficher ces deux fonctions, on appliquera respectivement les commandes `plt.plot(T,X[:,0])` et `plt.plot(T,X[:,1])`.
Pour afficher $t \mapsto (x(t), y(t))$, on appliquera `plt.plot(X[:,0],X[:,1])`

Exemple. *Résolvons avec* ***odeint*** *le système différentiel* $\left\{ \begin{array}{rcl} x'(t) & = & -x(t) - y(t) \\ y'(t) & = & x(t) - y(t) \end{array} \right.$ *sur* $[0, 10]$ *de pas* 0.01 *avec* $x(0) = 1$ *et* $y(0) = -1$. *On affichera ensemble* $t \mapsto x(t)$ *et* $t \mapsto y(t)$.

```
In [1]: import numpy as np ; import scipy.integrate as integr
In [2]: import matplotlib.pyplot as plt
In [3]: def F(x,t) : return(np.array([-x[0]-x[1], x[0] - x[1]]))
In [4]: T = np.arange(0,10,0.01) ; X = integr.odeint(F,np.array([1,-1]),T)
In [5]: plt.plot(T,X[:,0]) ; plt.plot(T,X[:,1]) ; plt.show()
```

2. Utilisation de la méthode d'Euler

Considérons le système différentiel du premier ordre $\left\{ \begin{array}{rcl} x'(t) & = & f(x(t),y(t),t) \\ y'(t) & = & g(x(t),y(t),t) \end{array} \right.$, où f et g sont des fonctions de $\mathbb{R}^3$ dans $\mathbb{R}$. Les conditions initiales sont $x(t0) = x0$ et $y(t0) = y0$.

Adaptons la fonction `Euler_Affich` du paragraphe I en une fonction `EulerSyst_Affich` qui a pour arguments f, g, $t0$, tn, n et $x0$, $y0$ et qui affiche la courbe intégrale $t \mapsto (x(t),y(t))$ pour $t \in [t0,tn]$ avec un pas $h = (tn - t0)/n$.

```
In [1]: import matplotlib.pyplot as plt
In [2]: def EulerSyst_Affich(f,g, t0,tn,n,x0,y0) :
            t=t0; x=x0; y=y0; h=(tn-t0)/float(n); T=[t0]; X=[x0]; Y=[y0]
            for k in range(n) :
                x,y = x + h*f(x, y,t), y + h *g(x,y,t); t=t+h
                T.append(t); X.append(x); Y.append(y)
            plt.plot(X,Y)
```

Exemple : *Appliquons à* $\left\{ \begin{array}{rcl} x'(t) & = & x(t)(1-y(t)) \\ y'(t) & = & y(t)(x(t)-1) \end{array} \right.$ *avec* $[t0,tn] = [0,10]$, $x(0) = 2$, $y(0) = 1$ *et* $n = 500$.

```
In [1]: import matplotlib.pyplot as plt
In [2]: def f(x,y,t) : return(x*(1-y))
In [3]: def g(x,y,t) : return(y*(x-1))
In [4]: EulerSyst_Affich(f,g,0,10,500,2,1)
```

Paragraph III

Resolution linears differentiels equations order 2

Soit $y''(t) + a(t)y'(t) + b(t)y(t) = g(t)$ avec la donnée de $y(t_0)$ et de $y'(t_0)$.

- ☞ **Étape 1.** On transforme notre équation différentielle en un système différentiel d'ordre 1 car nos procédures ne peuvent résoudre directement une équation différentielle d'ordre 2.
  On pose $X(t) = \begin{pmatrix} y(t) \\ y'(t) \end{pmatrix}$, $A(t) = \begin{pmatrix} 0 & 1 \\ -b(t) & -a(t) \end{pmatrix}$ et $B(t) = \begin{pmatrix} 0 \\ g(t) \end{pmatrix}$.
  Alors $X'(t) = \begin{pmatrix} y'(t) \\ y''(t) \end{pmatrix}$ et on a le système différentiel linéaire d'ordre 1 :
  $$X'(t) = A(t)\,X(t) + B(t).$$
- ☞ **Étape 2.** À partir de là, on rentre `integr.odeint` de `scipy.integrate` :
  `import numpy as np ; import scipy.integrate as integr`
- ☞ **Étape 3.** On rentre éventuellement les fonctions a, b et g (sauf si ce sont des constantes ou des fonctions simples que l'on incorpore directement dans `np.array`).
- ☞ **Étape 4.** On tape le système différentiel et on l'exécute.
  On cherche les solutions sur un intervalle `T` de borne gauche t_0 et les conditions initiales $y(t_0)$ et $y'(t_0)$ sont traduites par le tableau Python `np.array([x0,y0])`. On tape donc la syntaxe suivante :

  ```
  def F(x,t) : return(np.array([x[1],-b(t)*x[0]-a(t)*x[1]+g(t)]))
  T = np.arange(t0,tn,h)
  X = integr.odeint(F,np.array([x0,y0]),T)
  ```

  `X` est un tableau de deux colonnes, `X[:,0]` renvoie $t \mapsto y(t)$ et `X[:,1]` renvoie $t \mapsto y'(t)$.
- ☞ **Étape 5.** Pour tracer ces deux fonctions, on tape pour commencer :
  `import matplotlib.pyplot as plt`
  Puis on appliquera la commande `plt.plot(T,X[:,0])`

T.S.V.P →

Exemple : *Résoudre et afficher pour* $t \in [0,4]$ *l'unique solution de*

$$(E)\ y''(t) + 4ty(t) = 0,$$

de condition initiale $y(0) = 1$ *et* $y'(0) = -2$ *en utilisant* **`odeint`** *et on prendra un pas de* $h = 0.01$.

Posons $X(t) = \begin{pmatrix} y(t) \\ y'(t) \end{pmatrix}$, l'équation (E) correspond au système $X'(t) = A(t)X(t)$ en posant

$$A(t) = \begin{pmatrix} 0 & 1 \\ -4t & 0 \end{pmatrix}$$

avec ici $B(t) = 0$. On a :

$$F(X(t), t) = A(t)X(t) = \begin{pmatrix} y'(t) \\ -4ty(t) \end{pmatrix}.$$

Passons à la partie Python.

```
In [1]: import numpy as np; import scipy.integrate as integr
In [2]: def b(t):
            return 4*t
In [3]: def F(x,t):
          return(np.array( [x[1] , -b(t)*x[0]]))
In [4]: T = np.arange(0,4,0.01); X0 = [1,-2]; X = integr.odeint(F,X0,T)
In [5]: import matplotlib.pyplot as plt
In [6]: plt.plot(T, X[:,0]);plt.show()
```