Introduction à Scilab
Modélisation et systèmes différentiels

Cyrille Bertelle

UFRST
Université du Havre

M1 Math-Info - MD2PMS

Scilab permet de résoudre *numériquement* des équations différentielles. On ne cherche donc pas ici une solution analytique de la solution d'une équation différentielle mais on calcule une approximation de la solution à une succession de pas de temps, à partir d'une condition initiale.

Soit une équation avec une condition initiale (problème de Cauchy) :

$$\begin{cases} \dfrac{du(t)}{dt} &= f(t, u(t)) \\ u(t_0) &= u_0 \end{cases} \tag{1}$$

où $u : \boldsymbol{R} \to \boldsymbol{R}^n$, $u_0 \in \boldsymbol{R}^n$ et $f : \boldsymbol{R} \times \boldsymbol{R}^n \to \boldsymbol{R}^n$.

Le théorème de Cauchy-Lipschitz nous dit que si f est lipschitzienne par rapport à sa deuxième variable, sur un intervalle fermé contenant u_0, alors il existe une solution unique pour la fonction u, donnant ainsi une valeur unique de cette fonction en une valeur t donnée, à condition de ne pas sortir de l'intervalle dans lequel la condition de Cauchy-Lipschitz est énoncée. Dans la suite on supposera que cette condition de régularité de la fonction f est vérifiée.

La fonction `ode` est la fonction Scilab qui va permettre de calculer une solution numérique, c'est à dire de l'"intégrer" sur un intervalle démarrant en t_0, en partant du vecteur colonne u_0.

Pour cela, il faut commencer par définir la fonction f du problème de Cauchy décrit ci-dessus (1). On la définit comme une fonction Scilab classique :

```
function [f] = SecondMembreEDO(t,u)
  // on décrit ici les composantes de f
endfunction
```

Remarque

Même si la fonction f est autonome (c'est à dire indépendante de t), il faudra quand même mettre t comme premier argument de ce second membre, afin qu'ultérieurement, il soit identifié comme la variable de la solution cherchée.

Exemple

On considère l'équation de Van der Pol :

$$y'' = c(1 - y^2)y' - y$$

que l'on reformule classiquement comme un système de deux équations différentielles du premier ordre, en posant

$$\begin{aligned} u_1(t) &= y(t) \\ u_2(t) &= y'(t) \end{aligned}$$

Soit le système équivalent

$$\frac{d}{dt}\begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix} = \begin{bmatrix} u_2(t) \\ c(1 - u_1^2(t))u_2(t) - u_1(t) \end{bmatrix}$$

Exemple

Ce que l'on décrit par la fonction Scilab suivante, en fixant $c = 0.4$ *:*

```
function [f] = vanDerPol(t,u)
  f(1)=u(2)
  f(2)=0.4*(1-u(1)^2)*u(2) - u(1)
endfunction
```

Exemple

Pour résoudre numériquement l'équation, il faut donc définir une distribution uniforme de valeurs de t sur un intervalle, à partir de t_0, avec par exemple, la fonction linspace. On appelle ensuite la fonction `ode` *dont les arguments sont les suivants :*

- *les deux premiers paramètres correspondent à la condition initiale du problème de Cauchy, ce qui correspond ici au vecteur $[u_1(t_0), u_2(t_0)]$ et t_0 ;*
- *le troisième est le vecteur des valeurs successives de t ;*
- *le quatrième est la fonction f du problème de Cauchy (1).*

Exemple

Ceci correspond ainsi à deux lignes de code Scilab prenant la forme suivante :

```
t = linspace(t0,T,m);
[U] = ode(u0, t0, t, vanDerPol)
```

On récupère alors une matrice U telle que $U(i,j)$ est la solution approchée de $u_i(t(j))$.

Exemple

On donne ci-après un traitement complet de l'équation de Van Der Pol.

On commence par tracer le champ de vecteur correspondant de la fonction f du problème de Cauchy pour une valeur quelquonque de t, ici prise à 0, sachant que cette équation est autonome, ce champ de vecteur sera le même quelque soit la valeur de t.

Selon la construction du problème de Cauchy (1), ce champ vectoriel va décrire les tangentes aux trajectoires correspondant à des portraits de phase, c'est à dire aux courbes $(u_1(t), u_2(t))$, paramétrées par le temps.

Exemple

On effectue ensuite une résolution numérique de l'équation différentielle (avec `ode`*).*

On trace alors par-dessus le champ vectoriel, la courbe de portrait de phase démarrant à la condition initiale.

On trace ensuite dans deux autres fenêtres graphiques, l'évolution de chacune des composantes de la solution par rapport au temps.

```
function [f] = vanderpol(t,u)
  f(1)=u(2)
  f(2)=0.4*(1-u(1)^2)*u(2)-u(1)
endfunction

n=30;
delta=5;
x=linspace(-delta, delta,n); // on prend y=x
clf()
fchamp(vanderpol,0,x,x)
//tracé du champs de vecteur à l'instant t=0
```

```
m=500; T=30;
t=linspace(0,T,m);
u0 = [-2.5; 2.5];
plot2d(u0(1),u0(2),style=-9)
   //marquage de la condition initiale
[u] = ode(u0,0,t,vanderpol);
plot2d(u(1,:)', u(2,:)', style=2)
xtitle("Equation de Van Der Pol :
          plan de phase (y(t), dy(t)/dt)",
       "y", "dy/dt")
```

Modélisation et systèmes différentiels XII

```
xset("window",1);
plot2d(t', u(1,:)')
xtitle("Equation de Van Der Pol : y(t)", "t", "y")

xset("window",2);
plot2d(t', u(2,:)')
xtitle("Equation de Van Der Pol : dy(t)/dt",
          "t", "dy/dt")
```

Remarque

Derrière la résolution numérique de la fonction `ode`*, il y a plusieurs algorithmes possibles que l'on peut contrôler soi-même en indiquant celui que l'on souhaite utiliser (non décrit ici ... voir l'aide si nécessaire). Par défaut, un processus adaptatif est utilisé : il est capable de passer d'une méthode explicite standard (ici méthode d'Adams prédicteur/correcteur) à une méthode spécifique à une équation "raide" si l'équation est diagnostiquée en tant que telle, c'est à dire qu'elle s'intégre difficilement avec une méthode explicite (une méthode de Gear est alors utilisée).*