Université Pierre et Marie Curie
Licences de Mathématiques B
Année universitaire 2007-2008
Thomas Capricelli et Driss Yakoubi

Séance *Scilab* : Equations différentielles : correction

1 Mise en oeuvre de schémas de résolution numérique

1. Tant que y ne s'annule pas, il est possible de transformer le problème sous la forme

$$\frac{y'(t)}{y} = t \Leftrightarrow \frac{\mathrm{d}}{\mathrm{d}t} \ln(y(t)) = t$$

qui devient en intégrant entre t_0 et T

$$\int_{t_0}^{T} \frac{\mathrm{d}}{\mathrm{d}t} \ln(y(t)) \, \mathrm{d}t = \int_{t_0}^{T} t \, \mathrm{d}t$$

soit

$$\ln(y(T)) = \frac{T^2 - t_0^2}{2} + \ln(y(t_0))$$

d'où

$$y(T) = y(t_0) \, e^{\frac{T^2 - t_0^2}{2}}$$

2. Pour EulerExplicite on propose la fonction suivante

```
function u=EulerExplicite(fun,u0,t,n)
// fun est la fonction de l'equation differentielle
// u0 est la condition initiale en t0=t(1)
// t contient les temps auxquels on veut calculer la solution (temps initial en t(1))
// n est le nombre de pas entre deux temps consecutifs
np=length(t);   // nombre de pas de temps
dim=length(u0); // dimension (>1 si systeme)
u=ones(np,dim); // np valeurs dont condition initiale
u(1,:)=u0.'; //on transpose la condition initiale en colonne
            //dans la ligne 1 (temps initial) du vecteur solution
            //(ne sert que dans le cas d'un systeme differentiel)
for ipp=2:np
t1=t(ipp);
u0=u(ipp-1,:).'; //transposition dans le cas d'un systeme differentiel
t0=t(ipp-1);
h=(t1-t0)/n;
for i=1:n
 u0=u0+h*fun(u0,t0);
 t0=t0+h;
end
u(ipp,:)=u0.';  // transposition dans le cas d'un systeme differentiel
end
endfunction
```

Ce qui nous permet d'écrire

```
y0=1/10;
deff('y=fun0(x,t)','y=sin(t*x)')
deff('y=fun1(x,t)','y=t*x')
deff('y=fun2(t)','y=y0*exp(t.^2/2)')

N=20;
T=[0:0.5/N:0.5];
sol0=EulerExplicite(fun0,y0,T,1);
sol1=EulerExplicite(fun1,y0,T,1);
xbasc()
plot2d(T,sol0,[2,2],leg='y''=sin(ty)');
plot2d(T,sol1,[3,3],leg='y''=ty');
plot2d(T,fun2(T),[4,4],leg='solution exacte');
```

(les solutions discrètes sont très proches)

3. En prenant `N=11000` il est possible de vérifier que

```
E=norm(sol0'-fun2(T),%inf)
 E  =
    0.0000049
```

4.
```
function u=RungeKutta4(fun,u0,t,n)
// Memes parametres d'entree que EulerExplicite
np=length(t);
dim=length(u0);
u=ones(np,dim);
u(1,:)=u0.';    //on transpose la condition initiale en colonne
//               dans la ligne 1 (temps initial) du vecteur solution
//               (ne sert que dans le cas d'un systeme differentiel)
for ipp=2:np
t1=t(ipp);
u0=u(ipp-1,:).'; //transposition dans le cas d'un systeme differentiel
t0=t(ipp-1);
h=(t1-t0)/n;
for i=1:n
  u1=u0;
  u2=u0+h/2*fun(u1,t0);
  u3=u0+h/2*fun(u2,t0);
  u4=u0+h*fun(u3,t0);
  u0=u0+h/6*(fun(u1,t0)+ 2*(fun(u2,t0)+fun(u3,t0))+fun(u4,t0));
  t0=t0+h;
end
u(ipp,:)=u0.'; // transposition dans le cas d'un systeme differentiel
end
endfunction
```

2 Vérification de l'ordre de convergence

En faisant tourner le script on obtient la figure 2

1. L'équation différentielle sous-jacente est

$$\begin{cases} y' = 2t\sqrt{1-y^2}, \\ y(0) = 0. \end{cases}$$

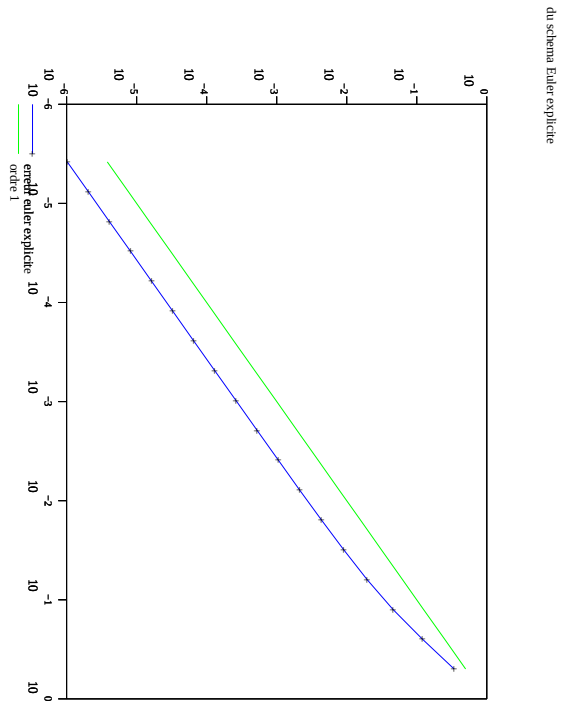

Figure 1: Ordre de convergence du schéma Euler explicite

la solution est calculée sur $[0:1]$ suivant `nt` intervalles mais les seules valeurs retournées sont l'approximation $(y(0), y(1))$.

2. D'après `help plot2d`

```
logflag     :  a string formed by to characters h (for horizontal axis)
            and v (for vertical axis) each of these characters can take
            the values "n" or "l". "l" stands for logarithmic graduation
            and "n" for normal graduation. For example "ll"stands for a
            log-log plot. Default value is "nn".
```

   ce qui indique que `ll` signifie une échelle logarithmique suivant chaque axe.

3. La qualité de l'approximation se dégrade significativement pour un pas h donné, du fait que la constante de Lipschitz associée à $\sqrt{1-y^2(t)}$ pour $y(t) = \sin(t^2)$ croît, jusqu'à perdre le caractère lipschiztien en $\sqrt{\pi/2}$. En ce point, on ne peut plus garantir la coonvergence de la méthode d'Euler. Par ailleurs, la croissance de la constante de Lipschitz nécessite l'utilisation de plus en plus de points pour obtenir une même qualité d'approximation (voir théorème 6.1.8 p 162 du livre de cours).

4. Adapter le script pour vérifier l'ordre numérique du schéma de Runge Kutta. Commenter la courbe obtenue.

3 Application à un problème vectoriel linéaire

Si on pose $p = \begin{pmatrix} m \\ l \end{pmatrix}$, le système précédent s'écrit $p' = Ap$ avec la matrice A à coefficients constants. On peut facilement la diagonaliser:

$$A \begin{pmatrix} 4 & -2 \\ 1 & 1 \end{pmatrix}, \quad AM = MD, \quad \text{avec} \quad M = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix} \quad \text{et} \quad D = \begin{pmatrix} 3 & 0 \\ 0 & 2 \end{pmatrix}$$

Si on multiplie le système différentiel de départà droite par la matrice M, et à gauche par la matrice M^{-1}, comme elle est à coefficient constants, on obtient:

$$\begin{aligned} (M^{-1}p)' &= M^{-1}Ap = M^{-1}AMM^{-1}p = M^{-1}MDM^{-1}p = DM^{-1}p \\ M^{-1}p(0) &= M^{-1}\begin{pmatrix} m_0 \\ l_0 \end{pmatrix} \end{aligned}$$

On pose maintenant le vecteur inconnu auxilliaire: $u = M^{-1}p$ qui est solution d'un système différentiel diagonal:

$$\begin{aligned} u' &= Du \\ u(0) &= M^{-1}\begin{pmatrix} m_0 \\ l_0 \end{pmatrix} = \begin{pmatrix} 1 & -1 \\ -1 & 2 \end{pmatrix}\begin{pmatrix} m_0 \\ l_0 \end{pmatrix} = \begin{pmatrix} m_0 - l_0 \\ -m_0 + 2l_0 \end{pmatrix} \end{aligned}$$

Soit, en intégrant chaque composante entre 0 et t,

$$\begin{aligned} u_1(t) &= u_1(0)e^{3t} = (m_0 - l_0)e^{3t} \\ u_2(t) &= u_2(0)e^{2t} = (-m_0 + 2l_0)e^{2t} \end{aligned}$$

d'où on tire la solution $p(t)$ en multipliant u par M

$$\begin{aligned} m(t) &= 2u_1(t) + u_2(t) = 2(m_0 - l_0)e^{3t} + (-m_0 + 2l_0)e^{2t} \\ &= m_0(2e^{3t} - e^{2t}) - 2l_0(e^{3t} - e^{2t}) \\ l(t) &= u_1(t) + u_2(t) = (m_0 - l_0)e^{3t} + (-m_0 + 2l_0)e^{2t} \\ &= m_0(e^{3t} - e^{2t}) - l_0(e^{3t} - 2e^{2t}) \end{aligned}$$

Tout ceci peut s'obtenir beaucoup plus rapidement si on a la définition des exponentielles de matrices vues en cours: Soit A une matrice diagonalisable sous la forme $A = MDM^{-1}$ on a $\forall n > 0$, $A^n = MD^nM^{-1}$. Et en utilisant les propriétés de la série exponentielle, on définit l'exponentielle de la matrice A

$$e^A = \sum_{n=0}^{\infty} \frac{1}{n!} A^n = \sum_{n=0}^{\infty} \frac{1}{n!} MD^nM^{-1} = M\left(\sum_{n=0}^{\infty} \frac{1}{n!} D^n\right) M^{-1} = Me^DM^{-1}$$

avec $(e^D)_{ij} = \delta_{ij}e^{\lambda_i}$ la matrice diagonale des exponantielles des valeurs propres de A. (soit, $D_{ij} = \delta_{ij}\lambda_i$). Avec cette définition, le système différentiel précédent s'intègre directement:

$$\left\{ \begin{array}{l} p' = Ap \\ p(0) = p_0 \end{array} \right. \quad \longrightarrow \quad p(t) = e^{tA}p_0$$

Le script suivant calcule la solution exacte et la représente graphiquement

```
A=[4,-2;1,1]; // Matrice associ\'e au syst\‘eme
Y0=[2;1];     // Condition initiale
T=[0:0.1:1]'; // Temps d'observation
Y= LinSysODE(A,Y0,T);
xbasc();
cadre=[min(T),min(-1,min(Y)),max(T),max(Y)];
plot2d (T,Y(:,1),[2,3],leg="moutons exact",rect=cadre);
plot2d (T,Y(:,2),[3,2],leg="loups exact",rect=cadre);
```

où la fonction LinSysODE est la fonction suivante

```
function Y=LinSysODE(A,X,T)
N=size(T,1);
Y=zeros(N,2);
for i=1:N
  Y(i,:)=(expm(T(i)*A)*X)';
end
endfunction
```

Pour intégrer numériquement le système linéaire en utilisant le schéma d'Euler explicite, on propose le script suivant:

```
deff('y=ml(x,t)','y=A*x')
sol=EulerExplicite(ml,y0,T,20);
// xbasc() // Sans xbasc pour l'avoir sur le m\^eme graphique
plot2d(T,sol(:,1),[4,4],leg="loups euler",rect=cadre);
plot2d(T,sol(:,2),[5,5],leg="moutons euler",rect=cadre);
```

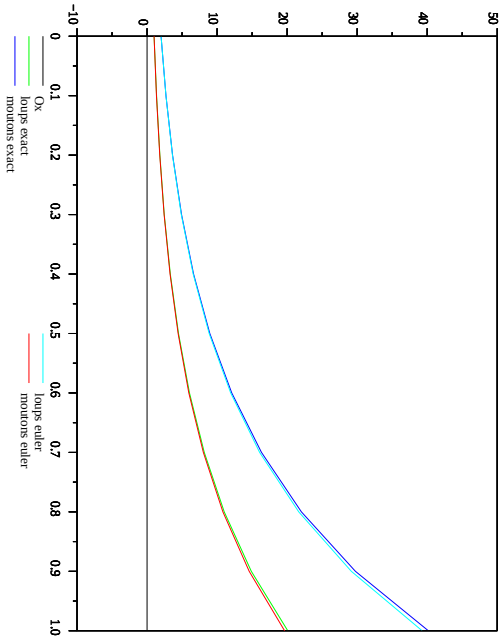

Figure 2: Comparaison de la méthode d'Euler explicite avec la solution analytique

4 Le modèle prédateur-proie

1. En l'absence de prédateurs, les proies ont un taux de croissance constant de a (en supposant une nourriture suffisamment abondante et l'absence de compétition)

$$\frac{m'(t)}{m(t)} = k_1, \text{ avec } k_1 > 0$$

De même, en l'absence de proies, la population de prédateurs décroît régulièrement

$$\frac{l'(t)}{l(t)} = -k_2, \text{ avec } k_2 > 0$$

Dès que les deux espèces sont mises en interaction, les prédateurs prélèvent une proportion relative à leur population dans l'espèce des proies

$$\frac{m'(t)}{m(t)} = k_1 - C\,l(t), \text{ avec } k_1 > 0,\, C > 0$$

et la quantité de proies favorise la croissance de la population des prédateurs

$$\frac{l'(t)}{l(t)} = -k_2 + D\,m(t), \text{ avec } k_2 > 0,\, D > 0$$

2. Dans sa forme non linéaire il suffit de modifier la fonction `ml` en la fonction `volterra` qui suit:

```
function y=volterra(x,t)
y=zeros(x);
y(1)=x(1)*(k1-C*x(2));
y(2)=x(2)*(D*x(1)-k2);
endfunction
```

dès lors le script suivant donne la figure

```
k1=2; k2=10; C=0.001; D=0.002;
y0=[5000;100];
T=0:0.05:10;

sol=EulerExplicite(volterra,y0,T,50);
cadre=[min(T),min(-1,min(sol)),max(T),max(sol)];
xbasc()
plot2d(T,sol(:,1),[4,4],leg="loups euler",rect=cadre);
plot2d(T,sol(:,2),[5,5],leg="moutons euler",rect=cadre);
```

3.

4. À l'aide d'une représentation de la forme $(x(t), y(t)) = (m(t), l(t))$ nous pouvons observer que le modèle engendre des cycles (figure 4) d'autant plus stable que la discrétisation du temps est fine.

```
xset("window",1);
xbasc()
cadre=[min(sol(:,1)),min(sol(:,2)),max(sol(:,1)),max(sol(:,2))];
plot2d(sol(:,1),sol(:,2),[2,2],leg="loups euler",rect=cadre);
xset("window",0);
```

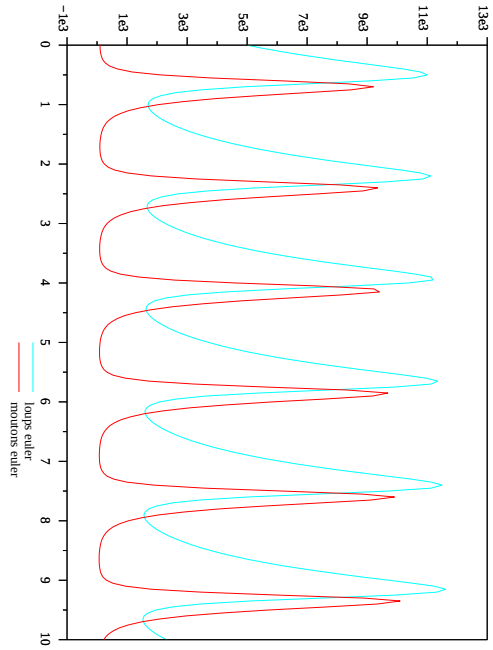

Figure 3: Solution du système prédateur-proie par une méthode d'Euler explicite

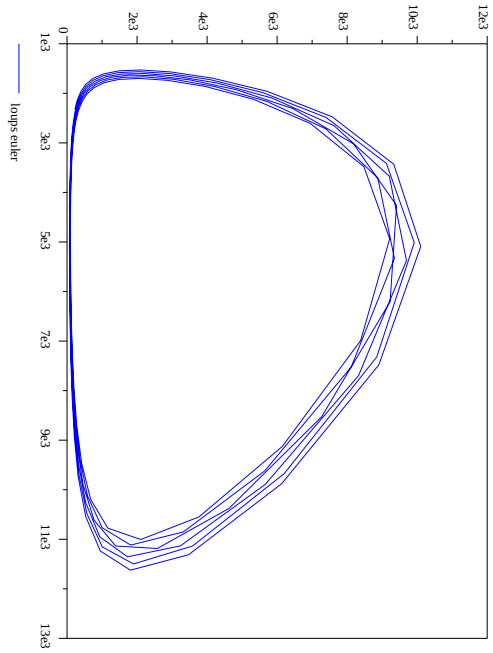

Figure 4: Évolution de la population des loups en fonction de celle des moutons

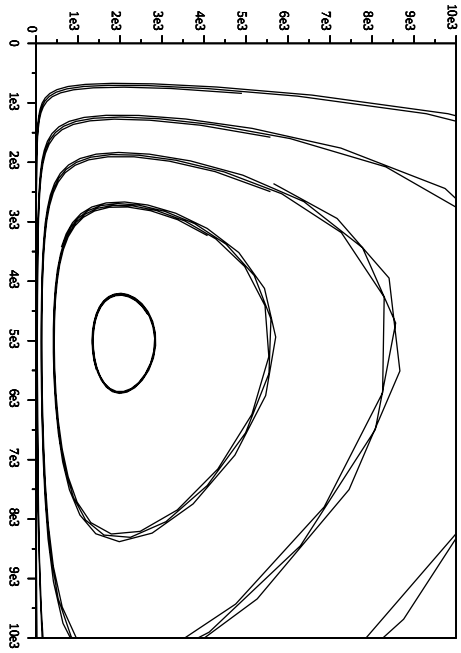

Figure 5: Évolution de la population des loups en fonction de celle des moutons suivant plusieurs conditions initiales

Pour un changement aisé de la condition initiale dans le cadre $[0, 10000] \times [0, 10000]$, le script suivant propose une utilisation de la fonction `xclick` permettant de cliquer le point initial (figuure 5).

```
xbasc();
xmin=0; xmax=10000; ymin=0; ymax=10000;
xset("thickness",2)
t0=0; tmax=5; dt=0.05; T=t0:dt:tmax;
while(%t)
  [c_i,x0,y0]=xclick()
  if c_i==2 then break end;
  if x0>=xmin & x0<=xmax & y0>=ymin & y0<=ymax then
    sol=EulerExplicite(volterra,[x0;y0],T,50);
    plot2d(sol(:,1),sol(:,2),rect=[xmin,ymin,xmax,ymax])
  end
end
```