**Option Calcul**

**L2 Mass** **2015-2016**

# Formulaire 3 : Intégration d'équations différentielles par les méthodes de Runge-Kutta

**version du 11 avril 2016**

## 1) Rappel : les équations différentielles et le problème de Cauchy

**1.1) Equation différentielle scalaire d'ordre 1.** On considère l'équation différentielle scalaire d'ordre 1 avec une condition initiale en $t = 0$ :

$$\begin{cases} y'(t) = f\big(t, y(t)\big) \\ y(0) = y_0 \end{cases} \quad \text{avec} \quad y : [0,T] \to \mathbb{R} \quad \text{et} \quad f : [0,T] \times \mathbb{R} \to \mathbb{R} \tag{1}$$

Dans l'équation (1) la fonction $t \mapsto y(t)$ est une fonction à valeurs réelles. L'existence et l'unicité de la solution est un problème complexe en mathématiques et ce n'est pas l'objet de ce cours. Comme il est en général impossible de calculer une solution analytique on construit numériquement une solution approchée. Les méthodes de Runge-Kutta sont une famille de méthode à pas constant dont la programmation est simple. Le problème modèle suivant sera utilisé comme test ( $f\big(t, y(t)\big) = y$ et $y_0 = 1$ ) :

$$\begin{cases} y'(t) = y \\ y(0) = 1 \end{cases} \quad \Longrightarrow \quad y(t) = e^t$$

**1.2) Système différentiel d'ordre 1.** Un système différentiel (ou équation différentielle à valeurs vectorielles) est une équation de la forme :

$$\begin{cases} Y'(t) = F\big(t, Y(t)\big) \\ Y(0) = Y_0 \end{cases} \quad \text{avec} \quad Y : [0,T] \to \mathbb{R}^n \quad \text{et} \quad F : [0,T] \times \mathbb{R}^n \to \mathbb{R}^n \tag{2}$$

Dans l'équation (2) la fonction $t \mapsto Y(t) \in \mathbb{R}^n$ est une fonction à valeurs vectorielles . Le problème modèle suivant sera utilisé comme test :

$$Y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \quad ; \quad A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad ; \quad \begin{cases} Y'(t) = A\,Y \\ Y(0) = Y_0 \end{cases} \quad \Longrightarrow \quad Y(t) = e^{tA}\,Y_0$$

dans cette expression la matrice $e^{tA}$ est l'exponentielle de la matrice $A$. On a par exemple pour

$$A = \begin{bmatrix} a & -b \\ b & a \end{bmatrix} \quad \text{et} \quad Y_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \Longrightarrow \quad Y(t) = e^{at} \begin{bmatrix} \cos(bt) \\ \sin(bt) \end{bmatrix}$$

Les algorithmes présentés dans la suite pour les équations différentielles scalaires d'ordre 1 seront aussi utilisés pour les systèmes différentiels d'ordre 1. Notons qu'une équation différentielle

d'ordre 2 avec 2 conditions initiales s'écrit classiquement sous la forme du système différentiel d'ordre 1 avec donnée initiale :

$$\begin{cases} y''(t) = f\big(t, y(t), y'(t)\big) \\ y(0) = y_0 \ ; \ y'(0) = y'_0 \end{cases} \quad \Leftrightarrow \quad \begin{cases} Y'(t) = F\big(t, Y(t)\big) \\ Y(0) = Y_0 \end{cases}$$

avec

$$Y(t) = \begin{bmatrix} y(t) \\ y'(t) \end{bmatrix} \quad ; \quad F\left(t, \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}\right) = \begin{bmatrix} y_2 \\ f\big(t, y_1, y_2\big) \end{bmatrix} \quad ; \quad Y_0 = \begin{bmatrix} y_0 \\ y'_0 \end{bmatrix}$$

Cette méthode se généralise à des équations différentielles d'ordre quelconque. Les équations différentielles d'ordre supérieur sont résolues numériquement avec les mêmes schémas en les transformant en un système différentiel d'ordre 1.

## 2) Les méthodes RK1, RK2 et RK4.

Le principe général pour approcher la solution sur un intervalle $[0, T]$ est de construire des suites $t_k = k\ h$ et $y_k$ tel que $y_k \sim y(t_k)$. Le nombre $h$ représente le pas d'intégration qui est constant dans ce cas. La différence entre ces méthodes réside principalement dans la précision de l'approximation. Dans cette partie la précision du schéma est testée sur l'équation différentielle (1) avec $f\big(t, y\big) = y$ et $y_0 = 1$. La solution analytique est dans ce cas $y(t) = \exp(t)$).

**2.1) La méthode d'ordre 1 (RK1).** C'est la méthode la plus simple mais elle est peu précise (l'erreur est en $O(h)$).

a) L'algorithme. La suite $y_k$ est construite par récurrence :

$$t_{k+1} = t_k + h \quad ; \quad y_{k+1} = y_k + h\ f\big(t_k, y_k\big)$$

b) La programmation.

```
// test RK2
clear
function z=f(t,y)
z=y;
endfunction

y0=1;T=1;
N=100;h=T/N;

// Calcul de la solution approchee
t=0;y=y0;

for j=1:N
t1=t;y1=y;ay1=f(t1,y1);
t=t+h;y=y+h*ay1;
end

// Calcul de l'erreur et affichage
erreur=exp(1)-y;
printf("y(%f)=%f \n",t,y);
printf("N=%d  erreur= %e",N,erreur);
```

Les résultats sont les suivants pour `N=100` et `N=200`.

```
-->exec('/home/demay/Ens:15-16/M1/MN/A-RK/RK1.sci', -1)
y(1.000000)=2.704814
N=100 erreur= 1.346800e-02

-->exec('/home/demay/Ens:15-16/M1/MN/A-RK/RK1.sci', -1)
y(1.000000)=2.711517
N=200 erreur= 6.764706e-03

-->1.346800e-02/6.764706e-03
 ans  =
    1.9909217
-->
```

On constate que l'erreur est divisée par $\sim 2$ : l'erreur est donc bien d'ordre $h$.

**2.2) Une méthode d'ordre 2 (RK2).** C'est une méthode plus précise (l'erreur est en $O(h^2)$).

a) L'algorithme. On utilise à chaque pas deux calculs de la valeur de la fonction $f$ :

$$t_{k+1} = t_k + h \quad ; \quad y_{k+1} = y_k + h\, f\Big(t_k + \frac{h}{2}\,,\, y_k + \frac{h}{2}\, f\big(t_k, y_k\big)\Big)$$

Cet algorithme s'écrit aussi :

$$t_{k+1} = t_k + h \quad ; \quad y_{k+1} = y_k + h a_2 \quad \text{avec} \quad \left\{ \begin{array}{l} a_1 = f(t_k, y_k) \\ a_2 = f\big(t_k + \frac{h}{2}\,,\, y_k + \frac{h}{2}\, a_1\big) \end{array} \right.$$

b) La programmation.

```
// test RK2
clear
function z=f(t,y)
z=y;
endfunction

y0=1;T=1;
N=200;h=T/N;

// Calcul de la solution approchee
t=0;y=y0;

for j=1:N
t1=t;y1=y;ay1=f(t1,y1);
t2=t+0.5*h;y2=y+0.5*h*ay1;ay2=f(t2,y2);
t=t+h;y=y+h*ay2;
end

// Calcul de l'erreur et affichage
erreur=exp(1)-y;
printf("y(%f)=%f \n",t,y);
printf("N=%d  erreur= %e",N,erreur);
```

Les résultats sont les suivants pour `N=100` et `N=200`.

```
-->exec('/home/demay/Ens:15-16/M1/MN/A-RK/RK2.sci', -1)
y(1.000000)=2.718237
N=100  erreur= 4.496590e-05

-->exec('/home/demay/Ens:15-16/M1/MN/A-RK/RK2.sci', -1)
y(1.000000)=2.718271
N=200  erreur= 1.128376e-05

-->4.496590e-05/1.128376e-05
 ans  =
    3.9850103
-->
```

On constate que l'erreur est divisée par $\sim 4$ : l'erreur est donc bien d'ordre $h^2$.

**2.3) Une méthode d'ordre 4 (RK4).**

C'est une méthode encore plus précise (l'erreur est en $O(h^4)$).

a) L'algorithme. On utilise à chaque pas quatre calculs de la valeur de la fonction $f$ :

$$t_{k+1} = t_k + h \quad ; \quad y_{k+1} = y_k + \frac{h}{6}\left(a_1 + 2\,a_2 + 2\,a_3 + a_4\right) \quad \text{avec} \quad \left\{ \begin{array}{l} a_1 = f(t_k, y_k) \\ a_2 = f\left(t_k + \frac{h}{2}\,,\, y_k + \frac{h}{2}\,a_1\right) \\ a_3 = f\left(t_k + \frac{h}{2}\,,\, y_k + \frac{h}{2}\,a_2\right) \\ a_4 = f\left(t_k + h\,,\, y_k + h\,a_3\right) \end{array} \right.$$

b) La programmation.

```
// test RK1
clear
function z=f(t,y)
z=y;
endfunction;

y0=1;T=1;
N=100;h=T/N;

// Calcul de la solution approchee
t=0;y=y0;
for i=1:N
t1=t;       y1=y;           ay1=f(t1,y1);
t2=t+0.5*h; y2=y+0.5*h*ay1; ay2=f(t2,y2);
t3=t+0.5*h; y3=y+0.5*h*ay2; ay3=f(t3,y3);
t4=t+h;     y4=y+h*ay3;     ay4=f(t4,y4);
t=t+h;y=y+h*(ay1+2*ay2+2*ay3+ay4)/6;
end

// Calcul de l'erreur et affichage
```

```
erreur=exp(1)-y;
printf("y(%f)=%f \n",t,y);
printf("N=%d  erreur= %e",N,erreur);
```

Les résultats sont les suivants pour `N=100` et `N=200`.

```
-->exec('/home/demay/Ens:15-16/M1/MN/A-RK/RK4.sci', -1)
y(1.000000)=2.718282
N=100  erreur= 2.246412e-10

-->exec('/home/demay/Ens:15-16/M1/MN/A-RK/RK4.sci', -1)
y(1.000000)=2.718282
N=200  erreur= 1.409894e-11

-->2.246412e-10/1.409894e-11
 ans  =
    15.933198
-->
```

On constate que l'erreur est divisée par $\sim 16$ : l'erreur est donc bien d'ordre 4 $\left(\epsilon = 0\left(h^4\right)\right)$.

**3) Le cas d'un système différentiel.**

On utilise les mêmes schémas. Considérons le système différentiel : On considère l'équation différentielle :

$$\begin{cases} Y'(t) = A\,Y \\ Y(0) = Y_0 \end{cases} \quad \text{avec} \quad A = \begin{bmatrix} a & -b \\ b & a \end{bmatrix} \quad \text{et} \quad Y_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{3}$$

Le programme suivant calcule la solution approchée de l'équation (3) par un schéma de Runge-Kutta d'ordre 1, trace la solution exacte puis donne l'erreur en $t = T$ :

```
clear
function Z=F(t,Y)
Z=(A*Y')';
endfunction

a=0.1;b=1;A=[a,-b;b,a];
Y0=[1,0];T=10;N=2000;h=T/N;

// Calcul de la solution approchee par RK1
t(1)=0;Y(1,:)=Y0;
for i=1:N
t1=t(i);       Y1=Y(i,:);           aY1=F(t1,Y1);
t(i+1)=t(i)+h;Y(i+1,:)=Y(i,:)+h*aY1;
end

// Tracé de courbe
plot2d(Y(:,1),Y(:,2),5);
for i=1:N+1 Y_ex(i,:)=exp(a*t(i))*[cos(b*t(i)),sin(b*t(i))]; end
plot2d(Y_ex(:,1),Y_ex(:,2),1);
printf("RK1: N=%d ; erreur en t=T:%e \n",N,norm(Y(N+1,:)-Y_ex(N+1,:)));
```

## Exercices

• **Exercice 1 :** On considère l'équation différentielle modèle :

$$\begin{cases} y'(t) = y(t) \\ y(0) = 1 \end{cases} \tag{4}$$

On utilise un schéma de Runge-Kutta d'ordre 1 (RK1) à pas constant $h = \dfrac{1}{N}$ en $t = 0$ et $t = 1$.

1.1) Calculer $y_{k+1}$ en fonction de $y_k$ et $h$.

1.2) Calculer $y_N$ et l'erreur en $t = 1$ en fonction du pas $h = \dfrac{1}{N}$ . Montrer que cette erreur est de la forme $\epsilon = O\big(h\big)$.

1.3) Même questions pour un schéma de Runge-Kutta d'ordre 2 (RK2). On montrera que l'erreur en $t = 1$ est de la forme $\epsilon = O\big(h^2\big)$.

• **Exercice 2 :**

2.1) Calculer $y_{k+1}$ en fonction de $y_k$ et $h$ pour la méthode RK4 dans le cas de l'équation différentielle (4).

2.2) Quelle est l'ordre de grandeur de l'erreur faite à chaque pas ? On admettra que cette erreur est dans ce cas particulier de la forme :

$$\epsilon_k = \big|y_{k+1} - e^h\, y_k\big|$$

• **Exercice 3 :**

On considère l'équation différentielle (3).

3.1) Tester le programme RK1 de la page 5.

3.2) Adapter ce programme pour RK2 et RK4.

3.3) Mettre la partie calculant la solution approchée par RK4 en `function`.

• **Exercice 4 : Modèle prédateurs-proies.**

On considère le système différentiel suivant (equations de Lotka–Volterra, $x > 0$ et $y > 0$) :

$$\begin{cases} \dfrac{dx}{dt} = ax - bxy \\ \dfrac{dy}{dt} = -cy + dxy \end{cases} \quad \text{avec les données initiales} \quad \begin{cases} x(0) = x_0 \\ y(0) = y_0 \end{cases} \tag{5}$$

4.1) Montrer que ce système différentiel admet un point fixe $\begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$ que l'on calculera.

4.2) Calculer numériquement les solutions de ce système différentiel pour des valeurs initiales de plus en plus éloignées du point fixe. Tracer ces solutions ainsi que le point fixe. On vérifiera que ces trajectoires sont périodiques.

4.3) Calculer numériquement la période d'une de ces solutions en utilisant une méthode de Newton.

4.4) Linéariser le système différentiel (5) autour du point fixe $\begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$. On introduira dans (5) des petites perturbations $\widehat{x}$ et $\widehat{y}$ de ce point fixe :

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \begin{bmatrix} \widehat{x} \\ \widehat{y} \end{bmatrix}$$

puis on négligera les termes d'ordres 2 en $\widehat{x}$ et $\widehat{y}$. Montrer que ce système différentiel linéarisé admet des solutions périodiques dont on calculera la période.

## Solutions

• **Exercice 3 :**

3.1)

```
clear
function Z=F(t,Y)
Z=(A*Y')';
endfunction

a=0.1;b=1;A=[a,-b;b,a];
Y0=[1,0];T=10;N=2000;h=T/N;

// Calcul de la solution approchee par RK1
t(1)=0;Y(1,:)=Y0;
for i=1:N
t1=t(i);       Y1=Y(i,:);           aY1=F(t1,Y1);
t(i+1)=t(i)+h;Y(i+1,:)=Y(i,:)+h*aY1;
end

// Tracé de courbe
plot2d(Y(:,1),Y(:,2),5);
for i=1:N+1 Y_ex(i,:)=exp(a*t(i))*[cos(b*t(i)),sin(b*t(i))]; end
plot2d(Y_ex(:,1),Y_ex(:,2),1);
printf("RK1: N=%d ; erreur en t=T:%e \n",N,norm(Y(N+1,:)-Y_ex(N+1,:)));
```

3.2)

```
clear
function Z=F(t,Y)
Z=(A*Y')';
endfunction

a=0.1;b=1;A=[a,-b;b,a];
Y0=[1,0];T=10;N=400;h=T/N;

// Calcul de la solution approchee par RK2
t(1)=0;Y(1,:)=Y0;
for i=1:N
t1=t(i);       Y1=Y(i,:);           aY1=F(t1,Y1);
t2=t(i)+0.5*h; Y2=Y(i,:)+0.5*h*aY1; aY2=F(t2,Y2);
t(i+1)=t(i)+h;Y(i+1,:)=Y(i,:)+h*aY2;
end

// Tracé de courbe
plot2d(Y(:,1),Y(:,2),5);
for i=1:N+1 Y_ex(i,:)=exp(a*t(i))*[cos(b*t(i)),sin(b*t(i))]; end
plot2d(Y_ex(:,1),Y_ex(:,2),1);
printf("RK2: N=%d ; erreur en t=T:%e \n",N,norm(Y(N+1,:)-Y_ex(N+1,:)));
```

3.3)

```
clear

function Z=F(t,Y)
Z=(A*Y')';
endfunction

function [t,Y]=RK4(N,T,Y0)
h=T/N;
// Initialisation
t(1)=0;Y(1,:)=Y0;
// Iterations
for i=1:N
t1=t(i);       Y1=Y(i,:);           aY1=F(t1,Y1);
t2=t(i)+0.5*h; Y2=Y(i,:)+0.5*h*aY1; aY2=F(t2,Y2);
t3=t(i)+0.5*h; Y3=Y(i,:)+0.5*h*aY2; aY3=F(t3,Y3);
t4=t(i)+h;     Y4=Y(i,:)+h*aY3;     aY4=F(t4,Y4);
t(i+1)=t(i)+h; Y(i+1,:)=Y(i,:)+h*(aY1+2*aY2+2*aY3+aY4)/6;
end
endfunction

// Programme principal
a=0.1;b=1;A=[a,-b;b,a];
Y0=[1,0];T=10;N=400;
[t,Y]=RK4(N,T,Y0);

// Trace de graphe
xset('window',1);xset('thickness', 2);
plot2d(Y(:,1),Y(:,2),style=6);
for i=1:N+1 Y_ex(i,:)=exp(a*t(i))*[cos(b*t(i)),sin(b*t(i))]; end
plot2d(Y_ex(:,1),Y_ex(:,2),1);
printf("RK4: N=%d ; erreur en t=T:%e \n",N,norm(Y(N+1,:)-Y_ex(N+1,:)));
```

• **Exercice 4**

4.2)

```
clear

function dY=F(t,Y)
dY=[a*Y(1)-b*Y(1)*Y(2),-c*Y(2)+d*Y(1)*Y(2)];
endfunction

function [t,Y]=RK4(N,T,Y0)
h=T/N;
// Initialisation
t(1)=0;Y(1,:)=Y0;
// Iterations
```

```
for i=1:N
t1=t(i);       Y1=Y(i,:);           aY1=F(t1,Y1);
t2=t(i)+0.5*h; Y2=Y(i,:)+0.5*h*aY1; aY2=F(t2,Y2);
t3=t(i)+0.5*h; Y3=Y(i,:)+0.5*h*aY2; aY3=F(t3,Y3);
t4=t(i)+h;     Y4=Y(i,:)+h*aY3;     aY4=F(t4,Y4);
t(i+1)=t(i)+h; Y(i+1,:)=Y(i,:)+h*(aY1+2*aY2+2*aY3+aY4)/6;
end
endfunction

// Programme principal
a=1;b=1;c=1;d=1;
// Calcul du point fixe
Y_f=[c/d,a/b];

T=10;Y0=[c/d,1.5*a/b];
N=500;[t,Y]=RK4(N,T,Y0);

// Trace de graphe
//clf();
xset('window',1);xset('thickness', 2);
plot2d(Y(:,1),Y(:,2),style=5);
plot2d(Y_f(1),Y_f(2),style=-2);
```

4.3)

```
clear

function dY=F(t,Y)
dY=[a*Y(1)-b*Y(1)*Y(2),-c*Y(2)+d*Y(1)*Y(2)];
endfunction

function [t,Y]=RK4(N,T,Y0)
h=T/N;
// Initialisation
t(1)=0;Y(1,:)=Y0;
// Iterations
for i=1:N
t1=t(i);       Y1=Y(i,:);           aY1=F(t1,Y1);
t2=t(i)+0.5*h; Y2=Y(i,:)+0.5*h*aY1; aY2=F(t2,Y2);
t3=t(i)+0.5*h; Y3=Y(i,:)+0.5*h*aY2; aY3=F(t3,Y3);
t4=t(i)+h;     Y4=Y(i,:)+h*aY3;     aY4=F(t4,Y4);
t(i+1)=t(i)+h; Y(i+1,:)=Y(i,:)+h*(aY1+2*aY2+2*aY3+aY4)/6;
end
endfunction

// Programme principal
a=1;b=1;c=1;d=1;
// Calcul du point fixe
```

```
Y_f=[c/d,a/b];

T=7;Y0=Y_f+[1,0];
N=500;[t,Y_t]=RK4(N,T,Y0);

// Calcul de la periode
Per_init=2*%pi;Per=Per_init;
test=1;eps=1e-8;
while test>eps
[t,Y]=RK4(N,Per,Y0);test=abs(Y(N+1,2)-Y_f(2));Der=F(t(N+1),Y(N+1,:))
Per=Per-(Y(N+1,2)-Y_f(2))/Der(2)
printf("Per=%f ; test=%e \n",Per,test);
end
[t,Y]=RK4(N,Per,Y0);

xset('window',1);xset('thickness', 2);
plot2d(Y(:,1),Y(:,2),style=1);
plot2d(Y_f(1),Y_f(2),style=-2);

xset('window',2);xset('thickness', 2);
plot2d(t,Y(:,1),style=1);
plot2d(t,Y(:,2),style=5);
```