Université de Versailles Saint-Quentin-en-Yvelines **Année Universitaire 2020-2021**
Master 1 CHPS **Cours de Calcul Numérique**
Tahar Z. Boulmezaoud

Schémas de résolution d'équations différentielles ordinaires

1 Généralités.

On considère l'équation différentielle sur l'intervalle $I = [t_0,\ T]$:

$$\left\{ \begin{array}{rcl} y'(t) & = & f(t, y(t)), \\ y(t_0) & = & a, \end{array} \right. \tag{1}$$

où a est un nombre réel donné et f une fonction de $[t_0,\ T] \times \mathbb{R}$ dans $\mathbb{R}$, donnée aussi, et vérifiant au moins les hypothèses du théorème de Cauchy-Peano assurant l'existence d'une solution y. Soit $t_0 < t_1 < ... < t_N = T$ une subdivision de l'intervalle $[t_0,\ T]$. On pose :

$$h_i = t_{i+1} - t_i \text{ pour } i = 0, ..., N-1, \text{ et } h = \max\{h_i;\ 0 \leqslant i < N\}$$

On cherche à approcher chacune des valeurs $y(t_i)$ par une valeur numérique y_i qu'on sait calculer sur machine. Le problème continu (1) est alors approché par un problème *discret* dans lequel la donnée initiale $y(t_0)$ est remplacée par une donnée y_0 qui lui est proche ou égale. On définit l'erreur commise à l'instant t_i par $e_i = |y(t_i) - y_i|$ et l'erreur globale sur $[0, T]$ par $e = \max_{1 \leqslant i \leqslant N} |y(t_i) - y_i|$. On souhaite donc que la méthode utilisée pour obtenir le problème discret *converge*, c'est-à-dire que $e \longrightarrow 0$ quand $h \longrightarrow 0$ et $y_0 \longrightarrow y(0)$. Pour assurer la convergence de la méthode *a priori*, c'est-à-dire avant de connaître la solution, il est nécessaire d'estimer l'erreur globale afin de montrer la convergene. La connaissance de la valeur exacte de l'erreur est généralement hors d'atteinte car cela demande de connaître la solution exacte du système, ce qui n'est possible que dans des cas particuliers, simples en général (de toute manière la but de la discrétisation est justement d'approcher cette solution exacte qu'on ne connaît pas analytiquement). Soulignons quand-même que les cas où la solution exacte est connue (p. ex. quand $f(t, x) = \lambda x$) sont utilisés pour tester justement les performances des méthodes employées et vérifier les prédictions théoriques concernant leur convergence.

L'analyse numérique de l'erreur conduit naturellement à l'introduction des deux notions de *stablité* et de *consistance* de la méthode. La stabilité signifie sommairement qu'une petite déviation dans la valeur initiale y_0 (par exemple une erreur d'arrondi), ne doit pas provoquer un *grande* déviation inattendue de l'une des valeurs y_i (en d'autres termes, la déviation de cette dernière valeur reste *contrôlable* par celle de la première). On renvoie par exemple à [1] et à [2] pour les détails de cette notion.
Quant à la consistance, elle repose sur l'étude de l'*erreur de consistance* définie par :

$$\epsilon_i = y_i - y_i^*, \text{ pour tout } i = 1, ..., N, \tag{2}$$

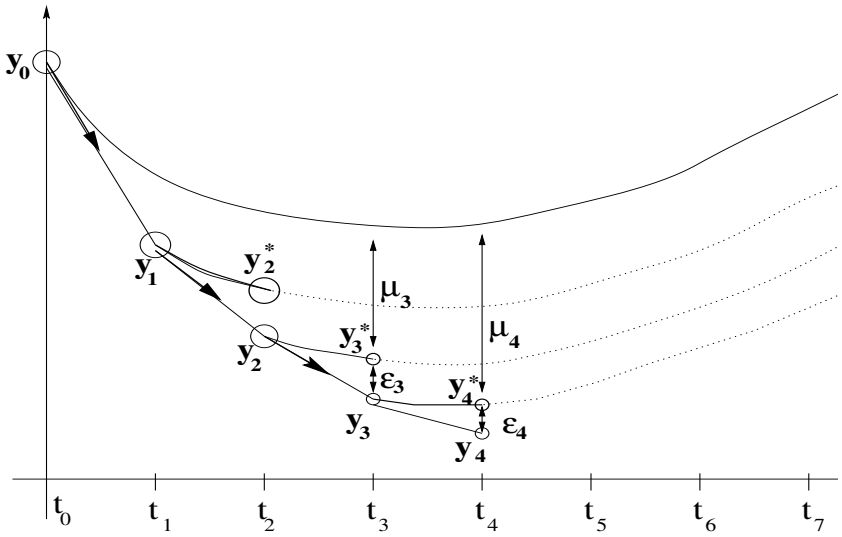
avec, pour tout i, $y_i^* = z_{i-1}(t_i)$ où la fonction $z_{i-1}(t)$ est la solution exacte sur l'intervalle $[t_{i-1},\ t_i]$ du problème

$$\left\{ \begin{array}{rcl} z'_{i-1}(t) & = & f(t, z_{i-1}(t)), \\ z_{i-1}(t_{i-1}) & = & y_{i-1}. \end{array} \right.$$

(c'est comme si on oubliait le fait que y_i comporte lui même une erreur par rapport à la valeur exacte. En effet, on note que $z_{i-1}(t) = y(t)$ sur $[t_{i-1}, t_i]$ si $y_{i-1} = y(t_{i-1})$).
L'erreur e_i peut alors être décomposée sous la forme

$$e_i = \epsilon_i + \mu_i, \tag{3}$$

avec ϵ_i l'erreur de consistance et $\mu_i = y_i^* - y(t_i)$ est due au cumul des erreurs antérieures à l'instant t_{i-1} (on remarque que $\mu_i = 0$ si $y_{i-1} = y(t_{i-1})$).



2 Quelques schémas à un pas.

Le schéma d'Euler explicite est un schéma à un pas qui s'écrit sous la forme :

$$\left\{ \begin{array}{rcl} y_0 & = & a, \\ y_{i+1} & = & y_i + h_i f(t_i, y_i), \quad \forall i \in \{0, ..., N-1\}. \end{array} \right. \tag{4}$$

D'un point de vue géométrique, ce schéma signifie que la pente de la tengente à la courbe $(t, y(t))$ au point $(t_i, y(t_i))$ peut être approchée par la pente du segment $[M_i(t_i, y(t_i)),\ M_{i+1}(t_{i+1}, y(t_{i+1})]$. Quand on approche la pente de la tangente au point $(t_{i+1}, y(t_{i+1}))$ par la pente du même segment on obtient le schéma d'Euelr implicite :

$$\left\{ \begin{array}{rcl} y_0 & = & a, \\ y_{i+1} & = & y_i + h_i f(t_{i+1}, y_{i+1}), \quad \forall i \in \{0, ..., N-1\}. \end{array} \right. \tag{5}$$

Ce schéma est dit implicite car y_{i+1} ne s'exprime pas explicitement en fonction de y_i, mais comme solution de l'équation suivante (qui n'est pas facile à résoudre en général) :

$$y - h_i f(t, y) = y_i. \tag{6}$$

On peut aussi obtenir ces deux schémas en utilisant une formule d'intégration par rectangle à gauche ou à droite dans l'égalité :

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} f(s, y(s))\, ds, \tag{7}$$

Le schéma de Crank-Nicolson est obtenu lui en utilisant une approximation de l'intégrale par l'aire d'une trapèze :

$$\left\{ \begin{array}{rcl} y_0 & = & a, \\ y_{i+1} & = & y_i + \dfrac{h_i}{2}(f(t_{i+1}, y_{i+1}) + f(t_i, y_i)), \quad \forall i \in \{0, ..., N-1\}. \end{array} \right. \tag{8}$$

C'est aussi un schéma implicite car le calcul de y_{i+1} nécessite la résolution de l'équation :

$$y_{i+1} - \frac{h_i}{2} f(t_{i+1}, y_{i+1}) = y_i + \frac{h_i}{2} f(t_i, y_i).$$

Si on remplace dans le schéma de Crank-Nicolson y_{i+1} par $y_i + h_i f(t_i, y_i)$ dans le terme $f(t_{i+1}, y_{i+1})$, on obtient le schéma explicite appelé schéma de Heun :

$$\left\{ \begin{array}{rcl} y_0 & = & a, \\ y_{i+1} & = & y_i + \dfrac{h_i}{2}(f(t_{i+1}, \underline{y_i + h_i f(t_i, y_i)}) + f(t_i, y_i)), \quad \forall i \in \{0, ..., N-1\}. \end{array} \right. \tag{9}$$

On peut aussi déduire un schéma en utilisant une formule de Taylor d'ordre p, $p \geqslant 1$, et en négligeant les termes d'ordre supérieur dans la formule :

$$y(t_{i+1}) = y(t_i) + h_i F(t_i) + \frac{h_i^2}{2} F'(t_i) + ... + \frac{h_i^p}{p!} F^{(p-1)}(t_i) + O(h^{p+1}), \tag{10}$$

où la fonction $F(t)$ est définie par $F(t) = f(t, y(t))$. La régles de dérivation composée des fonctions à plusieurs variables donne :

$$F'(t) = \frac{\partial f}{\partial t}(t, y(t)) + y'(t) \frac{\partial f}{\partial x}(t, y(t)) = [\frac{\partial f}{\partial t} + f. \frac{\partial f}{\partial x}](t, y(t)),$$

et

$$F''(t) = [\frac{\partial^2 f}{\partial t^2} + 2f \frac{\partial^2 f}{\partial t \partial x} + \frac{\partial f}{\partial t} \frac{\partial f}{\partial x} + f \left(\frac{\partial f}{\partial x} \right)^2 + f^2 \frac{\partial^2 f}{\partial x^2}](t, y(t)).$$

On obtient à l'ordre 3 par exemple :

$$y_{i+1} = y_i + h_i F_i^{(0)} + \frac{h_i^2}{2} F_i^{(1)} + \frac{h_i^3}{3!} F_i^{(2)}, \tag{11}$$

où $F_i^{(0)}$, $F_i^{(1)}$ et $F_i^{(2)}$ sont obtenus en remplaçant $y(t_i)$ par y_i dans les formules de $F(t_i)$, $F'(t_i)$ et $F''(t_i)$. Regardons maintenant l'erreur de consistance pour le schéma d'Euler explicite par exemple :

$$z_i(t_{i+1}) - y_{i+1} = z_i(t_i) + h_i z_i'(t_i) + \frac{h_i^2}{2!} z''(t_i) + o(h_i^2) - (y_i + h_i f(t_i, y_i)) = \frac{h_i^2}{2!} z''(t_i) + o(h_i^2).$$

On dit donc qu'elle est de l'ordre de h_i^2.
Exercice : déterminer l'ordre de l'erreur de consistance pour les schémas de Taylor d'ordre p, d'Euler implicite et de Crank-Nicolson.

3 Cas où $f(t, x) = \lambda x$

On considère maintenant le cas particulier où $f(t, x) = \lambda x$ avec λ une constante donnée. La solution exacte du système (1) vaut dans ce cas :

$$y(t) = y_0 e^{\lambda(t - t_0)}. \tag{12}$$

Cette solution nous servira pour tester les schémas décrits précédemment. Ces schémas s'écrivent dans ce cas :

— Schéma d'Euler explicite : $y_{i+1} = (1 + \lambda h_i) y_i$.
— Schéma d'Euler implicite : $y_{i+1} = \dfrac{y_i}{1 - \lambda h_i}$.
— Schéma de Crank-Nicolson : $y_{i+1} = \dfrac{2 + \lambda h_i}{2 - \lambda h_i} y_i$.

— Schéma de Taylor d'orde 3 : $y_{i+1} = \left(1 + \lambda h_i + \frac{(\lambda h_i)^2}{2} + \frac{(\lambda h_i)^3}{6}\right) y_i$.

Dans l'annexe A, on a mis le listing d'un programme type en Pascal qui compare ces schémas et qui écrit les erreurs dans un fichier 'errors.dat'. Dans le cas particulier où $t_0 = 0$, $T = 1.0$, $a = 1.0$, $\lambda = 1.0$ et $N = 10$, on obtient le résultat suivant :

Temps	La solution	LES ERREURS			
	exacte	Euler Exp.	Euler Imp.	Cra.- Nic.	Taylor (3)
0.00000000	1.00000000	0.00000000	0.00000000	0.00000000	0.0000000000
0.10000000	1.10517092	0.00517092	0.00594019	0.00009224	0.0000042514
0.20000000	1.22140276	0.01140276	0.01316514	0.00020389	0.0000093971
0.30000000	1.34985881	0.01885881	0.02188330	0.00033801	0.0000155780
0.40000000	1.49182470	0.02772470	0.03233321	0.00049811	0.0000229551
0.50000000	1.64872127	0.03821127	0.04478751	0.00068814	0.0000317116
0.60000000	1.82211880	0.05055780	0.05955762	0.00091266	0.0000420559
0.70000000	2.01375271	0.06503561	0.07699887	0.00117680	0.0000542254
0.80000000	2.22554093	0.08195212	0.09751638	0.00148642	0.0000684894
0.90000000	2.45960311	0.10165542	0.12157168	0.00184817	0.0000851539
1.00000000	2.71828183	0.12453937	0.14969016	0.00226959	0.0001045660

4 Cas où $f(t,x) = x^2$:

Si maintenant on considère la fonction $f(t,x) = x^2$. Cette fonction n'est pas lipschitzienne en x sur tout $\mathbb{R}$. Toutefois, elle est continue au voisinage de tout point (t_0, x_0) de $\mathbb{R} \times \mathbb{R}$, ce qui nous assure d'après le théorème de Cauchy-Peano l'existence d'une solution localement. La solution exacte du système (1) est :

$$y(t) = \frac{a}{1 - at} \tag{13}$$

Si $a > 0$, cette solution est maximale sur $[0, \frac{1}{a}[$. Elle comporte donc une singularité au point $x = \frac{1}{a}$. Le schéma d'Euler explicite s'écrit :

$$y_{i+1} = y_i + h_i y_i^2,$$

tandis que le schéma d'Euler implicite conduit à l'équation du second degré :

$$h_i y_{i+1}^2 - y_{i+1} + y_i = 0,$$

qui admet, si h_i est suffisament petit, deux solutions qui sont $\frac{1 - \sqrt{1 - 4h_i y_i}}{2h_i}$ et $\frac{1 + \sqrt{1 - 4h_i y_i}}{2h_i}$. La continuité de y fait qu'on prend $y_{i+1} = \frac{1 - \sqrt{1 - 4h_i y_i}}{2h_i}$ car cette dernière racine est plus proche de y_i quand h_i est petit. On remarque par ailleurs qu'en choisissant un pas h régulier, on risque d'avoir une erreur d'exécution car rien n'assure à l'avance que $1 - 4h_i y_i > 0$. En effet, on ne connaît pas a priori le comportement de y_i. Il est donc plus prudent et plus commode d'utiliser un pas *non uniforme* qui prévoit toute croissance exagérée de y, et donc au moins le bon déroulement du caclul. D'autre part, il est préférable que le calcul s'arréte si le pas de temps devient beaucoup trop petit. On propose le raffinement de pas suivant à chaque pas de temps :

$$h_{i+1} = \begin{cases} h_i & \text{si } 1 - 4h_i y_i > 0.0, \\ \dfrac{1}{10 y_i} & \text{sinon.} \end{cases}$$

Si $h_i < h_{\min}$ alors FIN DE CALCUL.

Dans la pratique, on constate qu'effectivement un pas uniforme conduit à une erreur d'éxécution après un certain nombre d'itérations car y_i devient trop grand (donc $1 - 4hy_i < 0$).

Références

[1] M. CROUZEIX ET A.L. MIGNOT, *Analyse numérique des équations différentielles*, Collection mathématiques appliquées pour la maîtrise, Masson, 1992.

[2] J.-P. DEMAILLY, *Analyse numérique et équations différentielles*, Collection Grenoble Sciences, Presse universitaire de Grenoble, 1991.

ANNEXE A

```
program equa_diff;
 { Programme ecrit par T. Boulmezaoud le 10/03/1999}

uses
     crt ;
const nmax     = 100;
      lambda   = 1.0;
type  tableau = array[0..nmax]  of real;

function       fonc(t, x : real) : real; forward;
   { retourne f(t, x) }

function     d1fonc(t, x : real) : real; forward;
  { retourne d[f(t, y(t))]/dt = (d_t f + f d_y f) (t, y(t)) }

function     d2fonc(t, x : real) : real; forward;
  { retourne d^2[f(t, y(t))]/dt^2 (formule dans le corrige ecrit)}

function Sol_Exacte(t0, y0, t : real) : real; forward;
    { retourne la valeur de la solution exacte de
       y'(t) = f(t, y(t)); y(t0) = y0 a l'instant t}

function foncEqua(t, coef, rhs : real) : real ; forward;
    { Retourne la solution de l'equation y - coef*f(t, y) = rhs }



procedure resolution(meth : integer; n : integer;
            t0, y0 : real; h : tableau; var  y : tableau);
 var
   i            : integer;
   temp, rhs    : real;
 begin
     y[0]   := y0;
     temp   := t0;
     case meth of
       1 :  { methode d'Euler explicite }
            for i := 1 to N do
             begin
              y[i] := y[i-1] + h[i]*fonc(temp, y[i-1]);
              temp := temp + h[i];
             end;
       2 :  { methode d'Euler implicite }
            for i := 1 to N do
             begin
              temp := temp + h[i];
              rhs  := y[i-1];
              y[i] := foncEqua(temp, h[i], rhs);
```

```
            end;
       3 :  { methode de Crank-Nicolson }
            for i := 1 to N do
            begin
              rhs   :=  y[i-1] + 0.5*h[i]*fonc(temp, y[i-1]);
              temp  :=  temp + h[i];
              y[i]  := foncEqua(temp, 0.5*h[i], rhs);
            end;
       4 :  { methode de Taylor d'ordre 3 explicite }
            for i := 1 to N do
            begin
              y[i] :=  y[i-1] + h[i]*fonc(temp, y[i-1])
                     + h[i]*h[i]*d1fonc(temp, y[i-1])*0.5
                     + h[i]*h[i]*h[i]*d2fonc(temp,y[i-1])/6.0;
              temp := temp + h[i];
            end;
       5 : { valeurs exactes }
           for i :=  1 to N do
            begin
             temp := temp + h[i];
             y[i] := Sol_Exacte(t0, y0, temp);
           end;
      end;
    end;

    procedure ystar(n : integer; t0: real; y, h : tableau;
                    var  yst : tableau);
     var
         i    : integer;
         temp : real;
         begin
          temp   := t0;
          yst[0] := y[0];
          for i:= 1 to N do
           begin
            yst[i] := Sol_Exacte(temp, y[i-1], temp + h[i]);
            temp := temp + h[i];
           end;
         end;


procedure diff_max(n : integer; y1, y2 : tableau;
                        var e : tableau; var vmax : real);
      var
           i    : integer;
      begin
        vmax := 0.0;
        for i := 0 to N do
         begin
           e[i] := abs(y1[i] - y2[i]);
           if (e[i] > vmax) then
                vmax :=  e[i];
         end;
     end;

function fonc ;
 begin
   fonc :=  lambda*x;
```

```
   end;

  function d1fonc;
   begin
      d1fonc := lambda*lambda*x;
   end;

  function d2fonc;
    begin
      d2fonc := lambda*lambda*lambda*x;
    end;

  function  foncEqua;
  begin
   if (coef*lambda <> 1.0 ) then
     foncEqua := rhs/(1.0 - coef*lambda)
   else
    begin
     writeln('Erreur dans la resolution de y-coef*f(t,y) = rhs');
     writeln('Programme arrete !');
     halt;
    end;
  end;

  function Sol_Exacte;
   begin
    Sol_Exacte := y0*exp(lambda*(t - t0));
   end;

var

 yex, y,  pas                              : tableau;
 er1, er2, er3, er4                      : tableau;
 t0, y_init, temp, Tmax, vmax   : real;
 nbpas, i                                    : integer;
 fsort                                          : Text;

 begin
    t0     := 0.0;
    Tmax   := 1.0;
    nbpas  :=  10;
    y_init := 1.0;

 { Pas regulier en temps }
    for i :=  1 to Nbpas do
      pas[i] := (Tmax-t0)/nbpas;

  resolution(5, nbpas, t0, y_init, pas, yex);
  resolution(1, nbpas, t0, y_init, pas, y);
  diff_max(nbpas, yex, y, er1, vmax);
  resolution(2, nbpas, t0, y_init, pas, y);
  diff_max(nbpas, yex, y, er2, vmax);
  resolution(3, nbpas, t0, y_init, pas, y);
  diff_max(nbpas, yex, y, er3, vmax);
  resolution(4, nbpas, t0, y_init, pas, y);
  diff_max(nbpas, yex, y, er4, vmax);

  { Affichage (ou ecriture dans le fichie de sortie)
```

```
   de la solution exacte et des erreurs }
  Assign(fsort, 'erreurs.dat');
  Rewrite(fsort);
  write(fsort, 'Temps     ': 13 , 'La solution': 13);
  writeln(fsort, 'LES ERREURS                ':52);
  write(fsort, ' ':13, 'exacte   ': 13);
  write(fsort, 'Euler Exp.':13);
  write(fsort, 'Euler Imp.':13);
  write(fsort, 'Cra.- Nic.':13);
  write(fsort, ' Taylor (3)  ':13);
  writeln(fsort);;

  temp   := t0;
  pas[0] := 0;
  for i:= 0 to nbpas do
   begin
      temp := temp + pas[i];
      write(fsort, temp:13:8 , yex[i]:13:8, er1[i]:13:8);
      write(fsort, er2[i]:13:8, er3[i]:13:8,  er4[i]:13:10);
      writeln(fsort);
   end;
   close(fsort);
end.
```