© *J.-B.A.K. <jean-baptiste.apoung@math.u-psud.fr>*

Fiche de TP2 : Schémas de Runge Kutta à pas Adaptatifs

Attention : *Au terme de la séance de TP, les questions qui n'auront pas été traitées, feront l'objet d'un devoir maison, à rendre* **avant** *le début de la prochaine séance de TP.*

Thème - 1 *Mis en en évidence de la nécessité d'adapter le pas : Equation de Van Der Pol*

On considère l'équation suivantes :

$$\left\{\begin{array}{rcl} x''(t) & = & \mu(1-x^2(t))x'(t) - x(t), \qquad t \in]t_0, t_0+T[\\ x(t_0) & = & x^0, \\ x'(t_0) & = & y^0. \end{array}\right. \tag{1}$$

Avec μ un paramètre réel positif. C'est une forme simplifiée d'une équation dite de Van Der Pol, proposée dans sa forme générale dans les années 1920, et dont l'utilité a été déterminante dans les domaines aussi variés que, l'étude des circuits à valves thermodynamique, des tubes à vide comme les téléviseurs cathodiques ou des magnétons comme les fours à micro-ondes.

Cette équation, bien que simple d'apparence, présente une difficulté numérique non négligeable en particulier lorsque le paramètre μ devient de plus en plus grand. Dans un autre langage on dirait qu'il est *de plus en plus raide pour μ grand*. Il est donc, pour nous, un bon candidat dans la mise en oeuvre du processus d'adaptation de pas dans la résolution numérique des EDOs.

Exercice-1 **:** **Mise en évidence de la difficulté à résoudre (1) numériquement**

Ici on va utiliser les solveurs intégrés de Matlab.

Q-1**-**1 **:** Mettre l'équation (1) sous la forme d'un système d'équations différentielles du premier ordre.

$$\left\{\begin{array}{l} X'(t) = F(t, X(t)), \quad t \in]t_0, t_0+T[. \\ X(t_0) = X^0 \in \mathbb{R}^2. \end{array}\right. \tag{2}$$

Pour toute la suite, on va prendre

$$t_0 = 0, \quad T = 6.6632868593231301896996820305, \quad x^0 = 2.0086198608748431365094 0188, \quad y^0 = 0. \tag{3}$$

Q-1**-**2 **:** Pour μ donné, on désigne par $Node23(\mu)$ respectivement $Node45(\mu)$ le nombre de pas de temps générés par le solveur de Matlab $ode23$ respectivement $ode45$ pour résoudre l'equation (2) avec les paramètre (3).

- Afficher $Node23(10^{-1})$, $Node45(10^{-1})$.
- Sur un même graphique, représenter $Node23$ et $Node45$ en fonction de μ, on prendra $\mu = [10^{-1}, 1, 10, 10^2, 10^3]$
- Mettre en évidence la difficulté à résoudre 1 lorsque μ devient grand.

Q-1**-**3 **:** Au regard des pas de temps générés dans les question ci-dessus, quelle serait pour ces valeurs de μ le plus grand pas de temps qu'il aurait fallut prendre pour résoudre le problème par un schéma de Runge Kutta à pas constant.

Exercice-2 **:** **Solutions proposées par Matlab pour contourner les difficultés précédentes**

Q-2**-**1 **:** Reprendre l' expérience précédente avec les solveurs $ode23s$ et $ode15s$.

Q-2-2 : Représenter, sur deux graphiques côte à côte, pour $\mu = 10^1$ les portraits de phase (c'est-à-dire la courbe de la trajectoire de x' en fonction de x en utilisant les points (*'-*' sous Matlab*) et comparer la densité des points générés par $ode23$ et $ode23s$. Conclure que Matlab dispose d'un outil adéquat pour résoudre les EDOs lorsqu'on a des doutes sur le caractère difficile des équations.

Thème - 2 *Vers la génération des schémas à pas adaptatifs : Monitoring des erreurs globales*

Afin de mettre en oeuvre les schémas à pas adaptatifs, est il primordiale d'être à mesure d'estimer, dans un schémas à pas constant, l'erreur commise à chaque instant $t_n, n = 0, \dots N$. **Attention il s'agit ici de l'erreur globale à l'instant** t_n. Ceci soulève de problème de *calculabilité* de l'erreur globale à l'instant t_n, dans un autre langage on parlerait d'estimation *a posteriori* de l'erreur globale.

Quelques définitions s'imposent (*voir le cours pour plus de détails*).

On considère l'équation différentielle suivante

$$\begin{cases} x'(t) = f(t, x(t)), \quad t \in]t_0, t_0 + T[, \\ x(t_0) = x_0, \end{cases} \tag{4}$$

ainsi que le schéma numérique à un pas (5), supposé convergent d'ordre p, pour sa résolution numérique

$$\begin{cases} x_{n+1} = x_n + h\Phi(t_n, x_n, h), \quad n = 0, 1, \dots N-1 \\ x_0 = x^0. \end{cases} \tag{5}$$

Définition (erreur locale) : On appelle erreur locale du schéma (5) à l'instant t au point y, la quantité

$$\xi(t, y, h) = x(t + h) - (y + h\Phi(t, y, h)) \tag{6}$$

où x est solution de $x'(s) = f(s, x(s)) \quad s \in]t, t + h][, \qquad x(t) = y.$

Cette écriture de l'erreur locale introduit une fonction que l'on appelle fonction principale d'erreur :

Définition (fonction principale) : On appelle fonction principale erreur du schéma (5), la fonction $\tau(\cdot, \cdot)$ telle que :

$$\xi(t, y, h) = \tau(t, y)h^{p+1} + \mathcal{O}(h^{p+2}). \tag{7}$$

Exercice-1 : Exemples de fonctions principales d'erreurs.
(*Ce poblème sera traité en TDs, on peut en admettre le résultat*).

On suppose le problème (4) mono-dimensionnel.

On considère le schéma de Runge-Kutta à deux étages

$$(rk2) \begin{cases} x_{n+1} = x_n + h(\alpha_1 p_{n,1} + \alpha_2 p_{n,2}) \\ p_{n,1} = f(t_n, x_n) \\ p_{n,2} = f(t_n + \mu h, x_n + h\mu p_{n,1}) \end{cases} \tag{8}$$

Q-1-1 : Montrer que le schéma est d'ordre 2 pour $\alpha_2 \neq 0, \quad \alpha_1 = 1 - \alpha_2, \quad \mu = 1/(2\alpha_2)$ et que dans ce cas,

$$\tau(t, y) = -\frac{1}{2}\left(\left(\frac{1}{4\alpha_2} - \frac{1}{3}\right)(f_{tt} + 2f_{ty}f + f_{yy}f) - \frac{1}{3}f_y(f_t + f_y f)\right)(t, y), \tag{9}$$

où, $f_s = \frac{\partial f}{\partial s}, \quad f_{sz} = \frac{\partial^2 f}{\partial s \partial z}, \qquad s, z \in \{t, y\}.$

Q-1-2 : En déduire la fonction principale d'erreur des schémas d'Euler modifié (ou point milieu) et de Heun.

Q-1-3 : Proposer un meilleur choix de α_2 et en déduire le schéma résultant. On l'appellera (rk2).

Exercice-2 : Validation

Affichage de l'erreur. On va montrer numériquement qu'on a une approximation calculable de l'erreur globale à chaque instant, donnée par

$$x(t_n) - x_n = e_n h^p + \mathcal{O}(h^{p+1}), \quad n = 0, \dots N-1, \qquad \text{quand } h \to 0 \tag{10}$$

où $\{e_n\}$ vérifie

$$\begin{cases} e_0 = x^0 - x_0, \\ e_{n+1} = e_n + h\left(f_y(t_n, x_n)e_n + \tau(t_n, x_n)\right), \quad n = 0, \dots, N-1. \end{cases} \tag{11}$$

On considère dans cette question que la fonction $f(\cdot,\cdot)$ de (4) est donnée par :

$$f(t,y) = (\cos(t) - 2t\tan(t^2))y, \qquad t_0 = 0, \quad T = \pi/3, \quad x^0 = 1.$$

Q-2-1 : Montrer que la solution exacte est $x(t) = \exp(\sin(t))\cos(t^2)$.

Q-2-2 : Pour programmer les schémas (8) et (11), fournir les fonctions Matlab suivantes

1. Fonction définissant le problème à résoudre

```
function [f, ft, fy] = monEDO (t,y)
% f  est  la valeur de la fonction second-membre f  au point (t,y)
% ft est la dériée partielle par rapport a t de la de f au point (t,y) (i.e df/dt )
% fy est la dériée partielle (jacobienne) par rapport a y de la de f au point (t,y) (i.e df/dy )
```

2. Fonction qui avance tout d'un pas. C'est-à-dire qui retourne x_{n+1} et e_{n+1} à partir de x_n et e_n

```
function [xnn,enn] = monUnPasErreur(tn,xn,en,h)
%ENTREE
% tn -> instant tn
% xn -> solution approchée a l'intant tn
% en -> erreur à tn (voir euqation (11)
% h  -> pas de temp utilisé
%SORTIE
% xnn correspond  a xn+1
% enn correspond  a en+1
```

Cette fonction ferra appel à la fonction **monEDO**, qu'on pourra lui passer en paramètre.

Q-2-3 : Fournir enfin un script Matlab

```
function [] = monTest(t0,tf,x0)
% ENTREE
% t0 instant initial
% tf instant final
% x0 solution initiale
```

qui résout numérique le problème (4) pour $h = 10^{-3}$ et qui affiche

```
t0      |x(t0) - x0|       |e0 * h^2|
t1      |x(t1) - x1|       |e1 * h^2|
...          ...               ...
tN      |x(tN) - xN|       |eN * h^2|
```

On commentera les résultats obtenus.

On va maintenant utilisé de qui précède pour fixer les pas h au cours de la résolution de l'EDO.

Exercice-1 : Approche naïve

Dans cette approche, on utilise la suite e_n calculée en utilisant la fonction principale d'erreur. Elle est naïve puisque dans la pratique on ne dispose pas toujours de cette fonction. et la génération de e_n est coûteuse.

Modifier le script **monTest** en fournissant à la place une fonction

```
function [t,x] = monTestNaif(t0, tf, x0, tol)
% Entree:
% t0  -> instant initial
% tf  -> instant final ( i.e T = tf - t0)
% x0  -> solution initiale
% tol -> telerance à l'erreur (ou erreur absolue)
%SORTIE:
% t liste des instants générés
% x liste des solutions aux instants tn
```

qui génère les instants t_n de la manière suivantes : pour un h donné,

- si l'erreur $|e_n|h^p < tol$ alors le pas h est accepté et on passe à l'étape suivante avec un pas $h = \min(5h, (\frac{0.8\ tol}{|e_n|})^{\frac{1}{p}})$. Pour éviter un d'avoir des pas trop grand on fournit un pas maximal $hmax$.
- sinon, on recommence avec un pas $h = \max(\frac{h}{5}, \left(\frac{0.8\ tol}{|e_n|}\right)^{\frac{1}{p}})$. Pour éviter que h ne devienne trop petit, on se donne un pas minimal $hmin$.

(*On peut remarquer qu'on peut résumer le choix du prochain pas par la relation*) $h = \max\left(\frac{h}{2}, \min\left(5h, \left(\frac{0.8\ tol}{|e_n|}\right)^{\frac{1}{p}}\right)\right)$.

Pour le pas initial h_0, on utilisera la fonction **pasInitial** fournie ci-dessous (*voir Thème 5*).

Exercice-2 : Approche évoluée

Dans une approche évoluée, on évite de résoudre explicite l'équation (11). A la place, on fait plutôt le choix de contrôler l'erreur locale dans le but visé de contrôler l'erreur globale. Pour cela pour une tol (tolérance) à l'erreur globale donnée, on choisit convenablement une tolérance à l'erreur locale tol_loc. Ceci nécessite de connaître *comment l'erreur locale est propagée par l'équation différentielle*, ce dont nous ne nous préoccupons pas ici. On va donc supposer que l'on dispose déjà de tol_loc. (*On peut montrer, voir TD que pour* $tol\_loc = \frac{\Lambda\ tol}{\exp(\Lambda T)-1}$, *où* Λ *est la constante de Lipschitz de* ϕ *par rapport à sa deuxième variable et uniformément par rapport aux autres, alors* $|\xi(t, x_n, h_n)| \leq h_n tol\_loc, \forall n$ *serait suffisant*).

Il est donc question de choisir h_n à l'instant t_n tel que

$|\xi(t, x_n, h_n)| \approx |\tau(t_n, x_n)|h_n^{p+1} \leq h_n\ tol\_loc$. Soit encore $\quad |\tau(t_n, x_n)|h_n^p \leq tol\_loc$.

Q-2-1 : Fournir une fonction **monUnPasEvolue** qui modifie la fonction **monUnPasErreur** comme décrit ci-dessous

```
function [xnn,tau] = monUnPasEvolue(tn,xn,h)
% xnn -> xn+1
% tau -> valeur de la fonction principale d'erreur a l'instant tn, au point xn
```

Q-2-2 : Reprendre l'exercice précédent en remplaçant la condition $|e_n|h^p < tol$ par $|\tau(t_n, x_n)|h^p \leq tol$. On fournira pour la circonstance une fonction Matlab

```
function [t,x] = monTestEvolue(t0, tf, x0, tol)
```

```
% t liste des instants générés
% x liste des solutions aux instants tn
```

Thème - 4 *Vers les schémas à pas adaptatifs : approximation de la fonction principale d'erreur approchée*

Dans la pratique, on ne dispose de la fonction principale de l'erreur, mais la théorie (voir cours) nous montre que si l'on peut remplacer la fonction principale d'erreur par une approximation d'odre 1 en h. c'est-à-dire qu'on peut remplacer τ par τ_a où $\tau_a(t,y,h) = \tau(t,y) + \mathcal{O}(h)$. (*Pour ce qui est du choix de τ_a, on montre (voir cours) que si ϕ est associé à un schéma d'ordre p, et si on peut trouver un schéma ϕ^* d'ordre au moins $p+1$, alors $\tau_a(t,y,h) = \frac{\phi^*(t,y,h)-\phi(t,y,h)}{h^p}$ est un bon candidat*). De tels exemples abondent en particulier dans les méthodes de Runge-Kutta dites emboîtés.

On considère par exemple la pair Dormand-Prince (5/4) plus connue sous le nom de DOPRI5, donnée ci-dessous sous forme de script Matlab

```
function [xnn, tau, ordre] = monUnPasDopri5(tn,xn,h,f)
% xnn -> solution calculee
% tau -> approximation de la fonction principale d'erreur
% ordre   -> ordre du schéma
% f   est le second membre de l'equation differentielle a resoudre
%%REMARQUE: pour plus d'efficacite on aurai pu retourner plutot ynn a la
%%          place de tau, car on a la relation  tau = (ynn-xnn)/h^(ordre+1)
  k1 = f(t,x);
  k2 = f(t+h/4, x + h*k1/4);
  k3 = f(t+3*h/8, x+h*(3*k1/32 + 9*k2/32));
  k4 = f(t+h*12/13, x+ h*(1932*k1/2197 - 7200*k2/2197+ 7296*k3/2197));
  k5 = f(t+h, x+h*(439*k1/216 -8*k2 +3680*k3/513 -845*k4/4104));
  k6 = f(t+h/2, x+h*(-8*k1/27 + 2*k2 -3544*k3/2565 + 1859*k4/4104 -11*k5/40));
  xnn  = x + h*(25*k1/216 + 1408*k3/2565 + 2197*k4/4104 - k5/5);
  ynn  = x + h*(16*k1/135 + 6656*k3/12825 +  28561*k4/56430 - 9*k5/50 + 2*k6/55);
  tau  = (ynn - xnn)/(h^5);
  ordre = 4;
```

Exercice-1 **:** Reprendre l'exercice en utilisant cette fois le schéma de DOPRI5.

On fournira à cette fin la fonction

```
function [t,x] = monTestEvolueApprochee(t0, tf, x0, tol)
% t liste des instants générés
% x liste des solutions aux instants tn
```

Exercice-2 **:** Fournir un script Matlab qui évalue vos implémentations en affichant côte à côte sur un graphique, le portrait de phase de la solution de l'équation de Van Der Pol pour $mu = 1$, obtenue par ode45 et par votre implémentation du schémas de DOPRI5.

Thème - 5 *Quelques utilitaires*

1. Une difficulté dans l'adaptation de pas est de choisir le pas initial. On propose a cet effet le script suivant :

```
function [h] =pasInitial(t0,x0,f,tol,ordre)
% fontion qui retourne le pas initial
% pour resoudre une edo par un schema à pas  addaptatifs
%
% t0  -> instant initial
% x0  -> solution initial
% f   -> second membre de l'EDO
% tol -> tolerance à l'erreur (ou erreur absolue)
% ordre -> l'ordre du schema considéré
```

```
        atol = tol;
        rtol = tol;
        n=length(x0);
        f1 = f(t0,x0);
        tol = atol + rtol * sqrt( dot(x0,x0)/n);
        d0 = sqrt( dot(x0,x0)/n)/tol;
        d1 = sqrt( dot(f1,f1)/n)/tol;
        if( (d0<1e-5) || (d1 < 1e-5))
        h0 = 1e-6;
        else
        h0 = 0.01*(d0/d1);
        end
        x1 = x0 + h0*f(t0,x0);
        f2 = f(t0+h0,x1);
        vd2=f2 -f1;
        d2 = sqrt( dot(vd2,vd2)/n)/(tol*h0);
        r=max(d0,d1);
        if(r <1e-15)
        h1 = max(1e-6,h0*1e-3);
        else
        h1 = (0.01/max(d1,d2))^(1.0/(ordre));
        end
       h = min(100*h0, h1);
```

2. Autres schémas utilisables

```
function [xnn, tau, ordre] = monUnPasFehlberg45(tn,xn,h,f)
% xnn -> solution calculee
% tau -> approximation de la fonction principale d'erreur
% ordre   -> ordre du schéma
% f   est le second membre de l'equation differentielle a resoudre
k1 = f(t,x);
k2 = f(t+h/4, x + h*k1/4);
k3 = f(t+3*h/8, x+h*(3*k1/32 + 9*k2/32));
k4 = f(t+h*12/13, x+ h*(1932*k1/2197 - 7200*k2/2197+ 7296*k3/2197));
k5 = f(t+h, x+h*(439*k1/216 -8*k2 +3680*k3/513 -845*k4/4104));
k6 = f(t+h/2, x+h*(-8*k1/27 + 2*k2 -3544*k3/2565 + 1859*k4/4104 -11*k5/40));
xnn = x + h*(25*k1/216 + 1408*k3/2565 + 2197*k4/4104 - k5/5);
ynn = x + h*(16*k1/135 + 6656*k3/12825 +  28561*k4/56430 - 9*k5/50 + 2*k6/55);
tau  = (ynn - xnn)/(h^5);
ordre = 4;
```