INTEGRATION IMPLICITE CUBIQUE
DES SYSTEMES DIFFERENTIELS

par Charles Hubert

Le 03/01/2011

Introduction

Pour intégrer un système différentiel on utilise souvent la méthode de Runge-Kutta. Quand le pas d'intégration dépasse une certaine valeur dépendant des équations, la méthode est instable, elle diverge. C'est parce qu'elle calcule l'état en fin de pas d'intégration à partir de l'état au début par une chaîne de formules explicites. Des algorithmes mettant en oeuvre cette méthode sont décrits dans l'article

INTEGRATION DES SYSTEMES DIFFERENTIELS PAR RUNGE-KUTTA

qu'il sera utile de consulter, notamment quand figurera le mot "ISDRK".

Complément de ISDRK, le présent article décrit une fonction "Icub" mettant en oeuvre une méthode implicite. Elle consiste à poser que l'état en fin de pas d'intégration est solution d'un système d'équations numériques. L'utilisation de "Icub" est plus facile si on sait déjà utiliser RunKutCtl (ISDRK).

L'algorithme d'intégration

Considérons le système différentiel linéaire

```
       t' = 1     x' = b y     y' = a (1 - x - y)
```

qu'on peut décrire (ISDRK) par les fonctions

```
     ∇  Detat←EqdLin etat
[1]     arr←1E6≤|etat
[2]     Detat←etat
[3]     Dt 1
[4]     Dx b×y
[5]     Dy a×1-x+y
     ∇
     ∇  Detat←IniLin;etat
[1]     1 VarEtat 't x y'
[2]     a←1E5 ⋄ b←1
     ∇
```

Intégrons par "RunKutCtl" (ISDRK) :

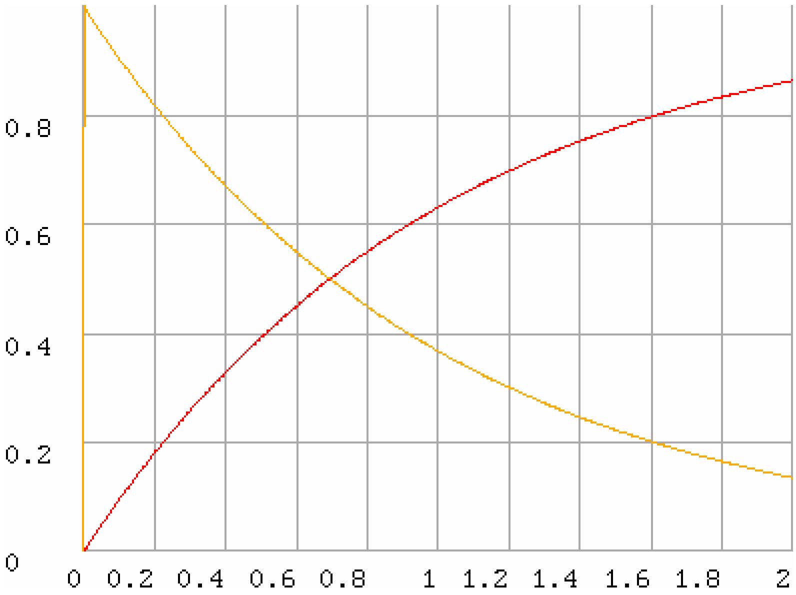
```
      petat← (2 512) 'EqdLin' 'arr' RunKutCtl IniLin
74151 3
```

La plupart des pas d'intégration valent 2^{-15} et 2^{-16}, entraînant beaucoup de calculs, parce que la méthode de Runge-Kutta est dans cet exemple à la limite de l'instabilité.

Intégrons par une autre fonction proposée ici "Icub" :

```
      petat← (2 512) 'EqdLin' 'arr' Icub IniLin
551 3
```

Les pas d'intégration commencent à 2^{-18} et augmentent progressivement jusqu'à $2^{-8} = 2/512$ au fur et à mesure de l'évanouissement du phénomène rapide. C'est parce que cette méthode est stable. On obtient les mêmes courbes :



Si on complète l'argument gauche de cet exemple on a

```
⍝              tndt      prec   dx    équations arrêt
⍝
   etat← (2 512 1024) 1E¯3 (2⋆¯6)  'EqdLin' 'arr' Icub 1 3 ρ0
⍝                                => dérivées partielles
⍝                                précision
⍝                                nombre max de subdivisions
⍝                                nombre min d'intervalles
```

Il peut y avoir trois sous-arguments gauches numériques : les étapes de l'intégration (tndt), la précision (prec) et les variations à appliquer à l'état pour le calcul des dérivées partielles (dx). Les parenthèses évitent les ambiguïtés. Les structures de prec et de dx doivent être compatibles avec celle d'un état du système. Si on appelle Icub sans argument gauche, elle indique sa syntaxe et les valeurs par défaut de l'argument gauche. Si un sous-argument gauche est absent ou vide, Icub prend sa valeur par défaut. La méthode de programmation est semblable à celle de RunKutCtl (ISDRK). Notamment, si on appelle Icub par

```
⍝                    équations
⍝
     ... (5 100) 'a b Equat c' Icub ...
```

la copie textuelle de la fonction "équations" lui fait solliciter les équations ainsi

```
⍝            équations
⍝
     ... a b Equat c (argumentLocal)
```

Comme pour RunKut et RunKutCtl (ISDRK) on peut intégrer plusieurs cas à la fois, ou appeler Icub avec un vecteur comme argument droit au lieu d'une matrice, les choses se passent comme avec RunKutCtl. La liste des variables d'état est dans "varEtat". La liste des objets créés par VarEtat est dans "objEtat".

La méthode d'intégration

Examinons d'abord un pas d'intégration. Au début du pas l'état x(t) est connu. Partons d'un état approché en fin de pas x(t+2h). Les équations différentielles "Eqd" appliquées à ces deux états fournissent les dérivées correspondantes

```
      x'(t) = Eqd(x(t))     x'(t+2h) = Eqd(x(t+2h))
```

Il existe un polynôme vecteur du troisième degré unique qui, aux bornes du pas, prend les valeurs x(t) et x(t+2h) et dont la dérivée par rapport à t prend les valeurs x'(t) et x'(t+2h). On pose que ce polynôme est une approximation de la solution du système différentiel. On en déduit la valeur de ce polynôme au milieu du pas :

$$x(t+h) = \frac{1}{2}\Big[x(t) + x(t+2h)\Big] + \frac{h}{4}\Big[x'(t) - x'(t+2h)\Big]$$

et les équations Eqd donnent la dérivée approchée en t+h

$$x'(t+h) = \mathrm{Eqd}(x(t+h))$$

La différence entre la variation de x de t à t+2h et la formule de Simpson est

$$\mathrm{Eqn}(x(t+2h)) = x(t+2h) - x(t) - \frac{h}{3}\Big[x'(t) + 4\ x'(t+h) + x'(t+2h)\Big]$$

ce qui est une fonction de x(t+2h) par la chaîne de formules ci-dessus. Si x(t+2h) est l'état en fin de pas on doit avoir

$$\mathrm{Eqn}(x(t+2h)) = 0$$

L'intégration implicite consiste à résoudre en x(t+2h) ce système d'équations numériques "Eqn" par la méthode de Newton. Celle-ci nécessite le calcul des dérivées partielles de Eqn(x(t+2h)) par rapport aux composantes de x(t+2h). Icub le fait en donnant à x(t+2h) de petites variations égales au troisième sous-argument gauche numérique.

Comme pour la méthode de Runge-Kutta, l'erreur est approximativement proportionnelle à la quatrième puissance du pas. Le contrôle de la précision se fait alors comme dans RunKutCtl (ISDRK). Pour estimer l'erreur Icub rassemble les trois états de deux pas consécutifs égaux

$$x(t) \quad x(t+2h) \quad x(t+4h)$$

les équations "Eqd" en déduisent les dérivées correspondantes

$$x'(t+nh) = \mathrm{Eqd}(x(t+nh)) \quad n \in \{0,\ 2,\ 4\}$$

Icub y applique la formule de Simpson et calcule la différence avec la variation de l'état sur ces deux pas

$$x(t+4h) - x(t) - \frac{2h}{3}\Big[x'(t) + 4\ x'(t+2h) + x'(t+4h)\Big]$$

Si la valeur absolue de cette erreur estimée est supérieure à la précision demandée, Icub rejette ces deux pas et recommence à x(t) avec pour nouveau pas 2h/2. Si la valeur absolue de cette erreur estimée est inférieure à 1/16 de la précision demandée, Icub poursuit à x(t+4h) avec pour nouveau pas 2×2h. Sinon Icub poursuit à x(t+4h) avec le même pas 2h.

Avantage et inconvénient de Icub

Comme on l'a vu sur un exemple, "Icub" adapte le pas d'intégration à la rapidité des phénomènes qu'elle rencontre, ce qui économise les calculs et les résultats.

Appliquée à un système d'ordre n, "Icub" résout des équations algébriques linéaires de dimension n, ce qui consomme beaucoup de temps de calcul si n est grand.

Les fonctions

On peut trouver les fonctions

Icub, VarEtat, IndVec, ValVec, EqaLin

dans le fichier APL*PLUS

EQUAT.SF

La composante 1 est une table des matières, les autres composantes sont les ⎕vr des fonctions de ce fichier.

Le code de VarEtat est listé dans l'article

INTEGRATION DES SYSTEMES DIFFERENTIELS PAR RUNGE-KUTTA

Les codes de IndVec et ValVec, appelées par VarEtat, sont listés dans l'article

TABLEAUX DANS UN VECTEUR UNIQUE

Le code de EqaLin, appelée par Icub, est listé dans l'article

RESOLUTION DES EQUATIONS ALGEBRIQUES LINEAIRES

```
     ∇  δ_z←δ_a Icub δ_x;δ_Int;δ_c;δ_d;δ_dx;δ_eq;δ_er;δ_er2;δ_f;δ_f0;
        δ_f1;δ_f2;δ_g;δ_h;δ_hM;δ_i;δ_k;δ_n;δ_q;δ_q0;δ_qM;δ_u;δ_x0;δ_y
[1]      δ_y← 'EqaDif' '2⋆10' '1E¯3' '2⋆¯6'
[2]     ⍝∇ x(...t0...tn) ← ((tn-t0){ n m}) {prec} {dx} {'Eq'} {'arr'}
        Icub x(...t0)
[3]     ⍝∇ Eq : équations  x'(t) ← Eq(x(t))          ; % par défaut
[4]     ⍝∇ n    ≡ nombre min de pas d'intégration   ; 1 par défaut
[5]     ⍝∇ m    ≡ nombre max de subdivisions        ; % par défaut
[6]     ⍝∇ prec ≡ précision                         ; % par défaut
[7]     ⍝∇ dx pour calculer les dérivées partielles ; % par défaut
[8]     ⍝∇ arr  ≡ arrêter l'intégration
[9]     ⍝∇ tableaux gigognes si 2>⍴⍴x(...t0)
[10]     δ_d←(δ_d[;⎕io]='⍝')≠δ_d←⎕cr 'Icub' ⋄ →(⎕nc 'δ_a')⍴δ_A
[11]     δ_d←'⍝', 0 2 ↓(δ_d[;⎕io+1]='∇')≠δ_d ⋄ (('%'=∊δ_d)/∊δ_d)←δ_y
[12]     δ_d←⊃∊¨⊂[⎕io+1]δ_d ⋄ δ_z←(⌽∨≠∨\' '≠⌽δ_d)/δ_d ⋄ →0
[13]    δ_A:δ_i←0=↑¨0⍴¨∊¨δ_a
[14]     (δ_c δ_u)←,¨2⍴((~δ_i)/δ_a),⊂'' ⋄ δ_c←↑(δ_c∧.=' ')↓ δ_c (⎕io⊃
        δ_y)
[15]     δ_g←2>⍴⍴δ_z←δ_x ⋄ δ_d← 0 2 ↓(δ_d[;⎕io+1]∊(⎕io+δ_g)⊃ 'i0' 'i1')
        ≠δ_d
[16]     →(δ_u∧.=' ')↓δ_B ⋄ δ_d←(δ_d∧.≠'#')≠δ_d
[17]    δ_B:(('#'≡¨,δ_d)/,δ_d)←⊂δ_u ⋄ (('@'≡¨,δ_d)/,δ_d)←⊂δ_c
[18]       δ_d←⎕fx⊃∊¨⊂[⎕io+1]δ_d
[19]     δ_u←δ_i/δ_a←,δ_a
[20]     δ_k←∊1↑δ_u ⋄ δ_u←,¨2↑(1↓δ_u), ⍬ ⍬
[21]     (δ_er δ_dx)←,¨|δ_u,¨(⍴¨δ_u)↓¨⍎¨δ_y[2 3 +⎕io] ⋄ δ_dx←δ_dx∘.×,2
[22]     (δ_hM δ_k δ_qM)←3↑δ_k,(⍴δ_k)↓ 0 1 ,⍎(1+⎕io)⊃δ_y ⋄ δ_hM←δ_hM÷
        δ_k←1⌈⌈δ_k
[23]     →δ_g⍴δ_C ⋄ δ_x←⊂[¯1↓⍳⍴⍴δ_x]δ_x
[24]    δ_C:δ_x←,δ_x++/(⍴¨δ_x)⍴¨0 ⋄ →(1↑⍴δ_z←⊃[⎕io],¨δ_x)↓0
[25]     δ_x←(¯1,¯1↑⍴δ_z)↑δ_z ⋄ →δ_g↓δ_D ⋄ δ_x←⊂¨δ_x
[26]    δ_D:δ_x0←((δ_g↓1),¯1↑1,⍴δ_x)⍴δ_x ⋄ δ_i←¯2+↑⍴δ_z
[27]     δ_u←(2⍴¯1↑⍴δ_x)⍴0 ⋄ (⎕io ⎕io ⍉δ_u)←0.5×,δ_dx ⋄ δ_u←⊂[⍳δ_g]δ_u;
        0;-δ_u
[28]     →δ_g↓δ_E ⋄ δ_z←⊂¨δ_z
[29]    δ_E:δ_er2← 0.0625 1 ×⊂δ_er ⋄ δ_q←1 ⋄ δ_x←(3,¯1↑⍴δ_x0)⍴δ_x0
[30]       →(δ_Int 0)↓δ_K ⋄ δ_f←(3,¯1↑⍴δ_f0)⍴δ_f0 ⋄ →δ_J
[31]    δ_F:δ_x0←δ_x[(δ_g↓1)⍴⎕io;] ⋄ →(δ_qM<δ_q←δ_q×2)⍴δ_K
[32]    δ_G:δ_h←δ_hM÷δ_q
[33]       →(δ_Int 1)↓ δ_K δ_F ⋄ δ_f[⎕io+1;]←(¯1↑⍴δ_f0)⍴δ_f0
[34]       →(δ_Int 2)↓ δ_K δ_F ⋄ δ_f[⎕io+2;]←(¯1↑⍴δ_f0)⍴δ_f0
[35]       δ_c←(-≠ 1 0 1 ≠δ_x)+δ_h×(⊃ 1 4 1 +.×δ_f)÷3
[36]       →(+/∧/¨∊¨δ_er2≥|⊂,δ_c)↓ δ_F δ_H ⋄ δ_q←1⌈δ_q÷1+0=4|2+δ_k×δ_q
[37]    δ_H:→((¯1+1↑⍴δ_z)>δ_i←δ_i+2)⍴δ_I ⋄ δ_z←δ_z;(1000,¯1↑⍴δ_z)⍴0
[38]    δ_I:δ_z[δ_i+⍳2;]← 1 0 ↓δ_x ⋄ δ_k←δ_k-2÷δ_q0
[39]       δ_x← 0 0 3 ≠δ_x ⋄ δ_f← 0 0 3 ≠δ_f
[40]    δ_J:→(δ_k>0)⍴δ_G
[41]    δ_K:δ_z←((δ_i+2),¯1↑⍴δ_z)⍴δ_z
[42]       →δ_g↓0 ⋄ δ_z←⊂[⎕io]⊃δ_z
[43]    ⍝i δ_i←δ_Int δ_j
[44]    ⍝i →δ_j↓δ_Z ⋄ δ_n←5 ⋄ δ_i←1 ⋄ δ_d←0 ⋄ δ_q0←δ_q
[45]    ⍝iδ_Y:→(δ_n←δ_n-1)↓0 ⋄ δ_x0←(⍴δ_u)⍴δ_x0 ⋄ δ_f0←(⍴δ_u)⍴δ_f[
        δ_j-1;]
[46]    ⍝0  δ_d←δ_u+(⍴δ_u)⍴δ_d
[47]    ⍝1  δ_d←δ_u+       δ_d
[48]    ⍝i  δ_f2←@(δ_d+δ_x0)
[49]    ⍝i  δ_f1←4×@(δ_x0+0.5×δ_d+(δ_h×0.25)×δ_f0-δ_f2)
[50]    ⍝0  δ_eq←       δ_d-δ_h×(δ_f1+δ_f0+δ_f2)÷6
[51]    ⍝1  δ_eq←⊃[⎕io]δ_d-δ_h×(δ_f1+δ_f0+δ_f2)÷6
[52]    ⍝0  δ_c←⍉δ_dx×EqaLin⍉-≠(2, 1 0 +¯1↑⍴δ_eq)⍴δ_eq;0
```

```
[53]   ⍝1  δ_c←,δ_dx×EqaLinq-≠(2, 1 0 +¯1↑⍴δ_eq)⍴δ_eq;0
[54]   ⍝0  δ_d←  δ_d[⎕io+¯1↑⍴δ_x0;]-δ_c
[55]   ⍝1  δ_d←⊂¨((⎕io+⍴δ_x0)⊃¨δ_d)-δ_c
[56]   ⍝i  →(∨/∊δ_er<|,δ_c)↑δ_Y
[57]   ⍝0 δ_x0←δ_x[δ_j+⎕io;]←δ_d+δ_x0[,⎕io;]
[58]   ⍝1 δ_x0←δ_x[δ_j+⎕io;]←δ_d+δ_x0
[59]   ⍝iδ_Z:δ_f0←⍎ δ_x0
[60]   ⍝i →(δ_i←^/0≥∊#)↓0
[61]   ⍝i δ_i←2
     ∇
```