COURS 8

Solutions séries d'équations différentielles

Résumé

L'algorithme de Newton dans un cadre différentiel permet de calculer des séries tronquées solutions d'équations différentielles en complexité quasi-optimale. Pour certaines classes particulières importantes d'équations, la complexité peut être encore abaissée.

Comme pour le cours 3 sur les calculs rapides de séries, N sera utilisé pour représenter le nombre de termes à calculer, et « série » sera employé pour « série tronquée à l'ordre N », $\mathsf{M}(N)$ dénote une borne supérieure sur le nombre d'opérations arithmétiques nécessaires pour multiplier deux polynômes de degré au plus N (et par conséquent deux séries tronquées à l'ordre N). On suppose pour simplifier les expressions asymptotiques que la multiplication est plus coûteuse que l'addition, c'est-à-dire que $\mathsf{M}(N)/N$ tend vers l'infini avec N. L'efficacité des algorithmes sera mesurée par leurs complexités arithmétiques. La complexité est alors quasi-optimale lorsqu'elle est linéaire en N à des facteurs logarithmiques près. Nous nous attacherons par ailleurs à expliciter la dépendance de la complexité vis-à-vis des autres paramètres (ordre r de l'équation, degré d des coefficients lorsqu'ils sont polynomiaux). Pour le cas particulier très important des équations et des systèmes différentiels *linéaires*, les résultats de complexité de ce cours sont présentés sur le tableau 1, sur lequel il faut comparer les complexités aux tailles des entrées et des sorties. Par comparaison, la méthode la plus directe (les coefficients indéterminés), est quadratique en N pour le cas où les coefficients sont des séries.

Tous les algorithmes sont énoncés pour des coefficients dans un anneau $\mathbb{A}$ (avec en vue le cas de coefficients polynomiaux par exemple). Dans tous ce cours nous ferons *l'hypothèse supplémentaire que* $2, 3, \ldots, N$ *sont inversibles dans* $\mathbb{A}$.

Problème (entrée, sortie)	coefficients séries	coefficients polynômes	coefficients constants	taille résultat
(équation, base)	$O(r^2\mathsf{M}(N))$	$O(dr^2N)$	$O(rN)$	rN
(équation, une solution)	$O(r\,\mathsf{M}(N)\log N)$	$O(drN)$	$O(\mathsf{M}(r)N/r)$	N
(système, base)	$O(r^2\mathsf{M}(N))$	$O(dr^\omega N)$	$O(r\mathsf{M}(r)N)$	r^2N
(système, une solution)	$O(r^2\,\mathsf{M}(N)\log N)$	$O(dr^2N)$	$O(\mathsf{M}(r)N)$	rN

TAB. 1. Complexité de la résolution d'équations et de systèmes différentiels linéaires pour $N \gg r$.

1. Équations différentielles d'ordre 1

Le cas de l'ordre 1 est plus simple que le cas général. Sa présentation sert d'introduction à la section suivante, où les techniques employées ici seront généralisées.

1.1. L'équation linéaire homogène du premier ordre. Cette équation,

$$y' = A(X)y,$$

où $A(X)$ est une série, admet la solution

$$y(X) = y(0)\exp\left(\int A(X)\right). \tag{1}$$

Le calcul utilisant cette formule est dominé par celui de l'exponentielle, donc en $O(\mathsf{M}(N))$ par les algorithmes du cours 3.

1.2. L'équation linéaire inhomogène du premier ordre. Il s'agit de l'équation

$$y' = A(X)y + B(X),$$

où A et B sont des séries. La méthode de la variation de la constante consiste à poser

$$y(X) = f(X)\exp\left(\int A(X)\right),$$

et à injecter cette expression dans l'équation pour obtenir que f vérifie

$$f'(X) = B(X)\exp\left(-\int A(X)\right).$$

L'ensemble du calcul de y est donc encore borné par $O(\mathsf{M}(N))$ opérations.

1.3. Le cas non-linéaire. Ces préliminaires avaient pour but de prouver le résultat plus général suivant.

Proposition 1. *Soit $F(X,Y)$ une série bivariée telle que pour toute série $s(X)$, les N premiers termes de $F(X,s(X))$ et de $\frac{\partial F}{\partial y}(X,s(X))$ se calculent en $O(\mathsf{L}(\mathsf{N}))$ opérations. Alors, l'équation différentielle*

$$y' = F(X,y), \qquad y(0) = a,$$

admet une série formelle solution, et ses N premiers termes peuvent être calculés en $O(\mathsf{L}(N) + \mathsf{M}(N))$ opérations.

L'énoncé de cette proposition en terme d'une fonction L permet de l'adapter à différents besoins : dans le cas le pire, il est possible d'invoquer l'algorithme de Brent et Kung du cours 3 pour avoir $\mathsf{L}(N) = O(\sqrt{N\log N}\mathsf{M}(N))$ qui domine alors la complexité de la résolution. Cependant, pour des équations plus simples, la composition avec F et sa dérivée pourra être en $O(\mathsf{M}(N))$. Ce sera le cas par exemple si F s'obtient à l'aide d'un nombre constant de sommes, d'exponentielles, de logarithmes, et de puissances.

Démonstration. L'idée de la preuve repose sur la méthode de Newton, appliquée cette fois-ci à un opérateur :

$$\Phi : y \mapsto y' - F(X,y).$$

Le principe consiste à linéariser l'opérateur Φ au voisinage d'une approximation et à en annuler la partie linéaire, pour obtenir une nouvelle approximation. À partir de

$$\Phi(y+h) = \Phi(y) + h' - \frac{\partial F}{\partial y}(X,y)h + O(h^2),$$

il s'agit donc de calculer un incrément h solution de l'équation *linéaire* inhomogène du premier ordre

$$h' = \frac{\partial F}{\partial y}(X,y)h - \Phi(y),$$

qui peut être résolue par la méthode de la section précédente. On déduit l'itération suivante

$$y_{k+1} := y_k + h_k,$$
$$h_k := \exp(\int \frac{\partial F}{\partial y}(X,y_k)) \int (F(X,y_k) - y_k') \exp(-\int \frac{\partial F}{\partial y}(X,y_k)) \bmod x^{2^{k+1}}.$$

Il reste à prouver la convergence quadratique. Comme pour le théorème sur l'itération de Newton sur les séries du cours 3 (p. 31), la preuve se fait par récurrence sur k en prouvant simultanément $\Phi(y_k) = O(X^{2^k})$ et $h_k = O(X^{2^k})$. Les calculs sont les mêmes et sont donc laissés en exercice.

Pour obtenir le résultat de complexité, il suffit d'observer que l'itération requiert à chaque étape la résolution d'une équation différentielle linéaire inhomogène du premier ordre et d'invoquer le lemme « Diviser pour régner ». □

2. Équations différentielles linéaires d'ordre supérieur et systèmes d'ordre 1

L'équation différentielle d'ordre r

$$a_r(X)y^{(r)}(X) + \cdots + a_1(X)y'(X) + a_0(X)y(X) = 0 \tag{2}$$

où les coefficients a_i sont des séries en X peut être récrite comme une équation d'ordre 1

$$Y' = A(X)Y \tag{3}$$

en prenant pour Y le vecteur de séries $(y(X), \ldots, y^{(r-1)}(X))$ et A une matrice (compagnon) de séries en X.

À l'inverse, un système (3) induit une relation de dépendance linéaire à coefficients des séries entre $Y, Y', Y'', \ldots, Y^{(r)}$ si r est la dimension de A (on a $r+1$ vecteurs en dimension r). Le vecteur Y et chacun de ses coefficients vérifient donc une équation de type (2).

Résoudre un problème est donc équivalent à résoudre l'autre. Dans certains cas, exploiter le caractère compagnon de la matrice induite par (2) mène à une meilleure complexité pour le cas des équations.

Une différence importante avec le cas des équations d'ordre 1 est que nous avons, pour les équations de type (2) comme pour les systèmes (3) deux problèmes à considérer :

- Calculer une base des séries solutions ;
- étant données des conditions initiales, calculer *la* série solution correspondante.

Les complexités de ces problèmes sont liées : si l'on sait résoudre le second pour un coût C, alors on sait résoudre le premier pour un coût borné par $r\mathsf{C}$. Si l'on sait résoudre le premier, alors une combinaison linéaire des solutions permet de résoudre le second en au plus rN opérations supplémentaires.

Il est cependant utile de considérer les quatre problèmes (système ou équation, base ou solution unique) car la structure de chacun des problèmes permet de concevoir des algorithmes plus efficaces que n'en donnent les conversions directes d'un problème à l'autre.

2.1. Une méthode par diviser pour régner. L'équation à résoudre à précision X^N est ici

$$Y' - AY = B, \quad Y(0) = v,$$

où A est une matrice de séries formelles. L'idée est de résoudre d'abord à précision moitié et de déterminer puis résoudre l'équation satisfaite par le reste. La condition initiale est d'abord traitée séparément en écrivant $Y = v + XU$ et en injectant dans l'équation, ce qui mène à

$$XU' + (I - XA(X))U = C, \quad C = B + A(X)v.$$

Soit maintenant $d < N$ et $U = U_0 + X^d U_1$ où U_0 est n polynôme de degré au plus $d-1$ en X, l'équation satisfaite par U donne :

$$XU_1' + ((d+1)I - XA(X))U_1 = -X^{-d}(XU_0' + (I - XA(X))U_0 - C).$$

Si U_0 est une solution à précision d, le membre droit de l'équation est bien un polynôme. L'application du paradigme « diviser pour régner » amène donc naturellement à considérer l'équation plus générale

$$XY' + (pI - XA)Y = R,$$

où R est une série, A une matrice de séries et $p \in \{1, \ldots, N\}$.

L'algorithme récursif est donc le suivant :

DivideAndConquer(A, p, s)

Input : $A_0, \ldots, A_{N-p}$ dans $\mathcal{M}_{r\times r}(\mathbb{A})$, $A = \sum A_i X^i$, $s_0, \ldots, s_{N-p}$ dans $\mathcal{M}_{r\times \ell}(\mathbb{A})$, $s = \sum s_i X^i$, $p \in \{1, \ldots, N\}$.

Output : $y = \sum_{i=0}^{N-p} y_i X^i$ dans $\mathcal{M}_{r\times \ell}(\mathbb{A})[X]$ tel que $Xy' + (pI - XA)y = s + O(X^{N-p+1})$.

If $m = 1$ then return $p^{-1}s(0)$
else
  $m \leftarrow N - p$
  $d \leftarrow \lfloor m/2 \rfloor$
  $y_0 \leftarrow$ DivideAndConquer$(A, p + d, s \bmod X^d)$
  $R \leftarrow [s - Xy_0' - (pI - XA)y_0]_d^m$
  $y_1 \leftarrow$ DivideAndConquer$(A, p + d, R)$
  return $y_0 + X^d y_1$

La notation $[s]_d^m$ désigne le quotient de la division euclidienne de $s \bmod X^m$ par X^d.

Cet algorithme est invoqué par l'algorithme de résolution suggéré ci-dessus et dont voici une version détaillée.

SolvebyDAC(A, N, B, v)

Input : $A_0, \dots, A_N$ dans $\mathcal{M}_{r\times r}(\mathbb{A})$, $A = \sum A_i X^i$, $B_0, \dots, B_N$ et v dans $\mathcal{M}_{r\times \ell}(\mathbb{A})$, $B = \sum B_i X^i$.

Output : $y = \sum_{i=0}^{N} y_i X^i$ dans $\mathcal{M}_{r\times\ell}(\mathbb{A})[X]$ tel que $y' - Ay = B$ et $y(0) = v$.

return $v + X$DivideAndConquer$(A, 1, B + Av)$

Les résultats de complexité se déduisent de cette méthode pour différentes valeurs de A et de ℓ. En particulier, le cas des équations a une meilleure complexité que le cas général d'un système d'ordre 1 car la matrice correspondante est une matrice compagnon. Le produit d'une telle matrice par un vecteur ne coûte que r opérations au lieu de r^2 dans le cas général.

THÉORÈME 1 (Diviser pour régner pour les équations différentielles). *Étant donnés les N premiers coefficients de $A \in \mathcal{M}_{r\times r}(\mathbb{A}[[X]])$, de $B \in \mathcal{M}_{r\times\ell}(\mathbb{A}[[X]])$ ainsi que des conditions initiales $v \in \mathcal{M}_{r\times\ell}(\mathbb{A})$, l'algorithme* SolvebyDAC *calcule l'unique solution de*

$$Y' - AY = B + O(X^N), \qquad Y(0) = v$$

en un nombre d'opérations dans $\mathbb{A}$ borné par

1. $O(r^2\ell \mathsf{M}(N)\log N)$ *en général ;*
2. $O(r\ell \mathsf{M}(N)\log N)$ *si A est une matrice compagnon.*

Il est possible d'améliorer les estimations de complexité pour cet algorithme lorsque $\ell = r$, mais dans ce cas du calcul de toute une base des solutions, il vaut mieux employer les algorithmes de la section suivante, qui sont plus efficaces.

DÉMONSTRATION. Il suffit de prouver le résultat pour N une puissance de 2, et le cas général s'en déduit par les hypothèses habituelles sur la fonction de multiplication M.

Les opérations de troncature ne demandent pas de calcul dans $\mathbb{A}$. Outre les deux appels récursifs, le calcul demande une dérivation (linéaire), un produit par p fois l'identité (linéaire) et un produit par A. La complexité $C(N)$ vérifie donc

$$C(N) = 2C(N/2) + \lambda\ell N + \mathsf{Mult}(A, V, N),$$

où $\mathsf{Mult}(A, V, N)$ désigne le coût du produit de la matrice A par la matrice $r \times \ell$ de séries à précision N. La conclusion s'obtient par le lemme « Diviser pour régner » en observant les complexités de base pour $\mathsf{Mult}(A, V, N)$. □

2.2. L'itération de Newton matricielle. Comme dans la preuve de la Proposition 1, le point de départ de l'algorithme consiste à appliquer l'itération de Newton à l'opérateur dont on cherche les solutions, en linéarisant au voisinage d'une solution approchée. L'opérateur

$$Y \mapsto Y' - A(X)Y$$

étant linéaire, il est son propre linéarisé. Formellement, l'itération de Newton s'écrit donc

$$Y_{k+1} = Y_k - U_k, \quad U_k' - AU_k = Y_k' - AY_k.$$

Lorsque Y_k est une solution approchée, le membre droit de l'équation en U_k a une valuation strictement positive, et la solution U_k cherchée doit avoir aussi une

SolveHomDiffSys(A, N, Y_0)

Input : $Y_0, A_0, \ldots, A_{N-2}$ in $\mathcal{M}_{r\times r}(\mathbb{A})$, $A = \sum A_i X^i$.

Output : $Y = \sum_{i=0}^{N-1} Y_i X^i$ dans $\mathcal{M}_{r\times r}(\mathbb{A})[X]$ tel que $Y' = AY + O(X^{N-1})$, et $Z = Y^{-1} + O(X^{N/2})$.

$Y \leftarrow (I + XA_0)Y_0$
$Z \leftarrow Y_0^{-1}$
$m \leftarrow 2$
while $m \leq N/2$ do
    $Z \leftarrow Z + Z(I_r - YZ) \bmod X^m$
    $Y \leftarrow Y - Y\left(\int Z(Y' - A \bmod X^{2m-1}Y)\right) \bmod X^{2m}$
    $m \leftarrow 2m$
return Y, Z

FIG. 1. Résolution de $Y' = A(X)Y$, $Y(0) = Y_0$ par itération de Newton

telle valuation. Une telle solution est obtenue par la méthode de la variation de la constante *si l'on dispose d'une base des solutions*, c'est-à-dire dans notre cas si Y_k est une matrice $r \times r$ avec $\det Y_k(0) \neq 0$. Dans ce cas, la méthode de la variation de la constante suggère de poser $U_k = Y_k T$, ce qui donne

$$U_k' - AU_k = Y_k T' + \underbrace{AY_k T - AY_k T}_{0} = Y_k' - AY_k$$

d'où finalement

$$U_k = Y_k \int Y_k^{-1}(Y_k' - AY_k).$$

Cette méthode requiert le calcul de l'inverse d'une base de solution, inverse qui peut être calculé simultanément par l'itération de Newton :

$$Z_{k+1} = Z_k + Z_k(I - Y_k Z_k).$$

Une version précise de l'algorithme, avec les ordres de troncatures, est donnée en Figure 1.

LEMME 1 (Correction de l'algorithme). *Soit m un entier pair, soient y et z dans $\mathcal{M}_{r\times r}(\mathbb{A}[X])$ tels que*

$$I - yz = O(X^{m/2}), \quad y' - Ay = O(X^m).$$

Soient ensuite Y et Z définis par

$$Z := z(2I - yz) \bmod X^m, \quad Y := y(I - \int Z(y' - Ay)) \bmod X^{2m}.$$

Alors Y et Z vérifient

$$I - YZ = O(X^m), \quad Y' - AY = O(X^{2m-1}).$$

DÉMONSTRATION. D'après l'hypothèse sur $y' - Ay = O(X^m)$, $Y = y + O(X^m)$ et donc la convergence de l'inversion est donnée par

$$\begin{aligned} I - YZ &= I - y(z + z(I - yz)) + O(X^m), \\ &= (I - yz) - yz(I - yz) + O(X^m), \\ &= (I - yz)^2 + O(X^m) = O(X^m). \end{aligned}$$

La seconde propriété s'obtient en injectant la définition de Y dans $Y' - AY$ (on pose $Q = \int Z(y' - Ay)$) :

$$\begin{aligned} Y' - AY &= y' + y'Q - Ay - AyQ - yZ(y' - Ay) + O(X^{2m}), \\ &= (I - yZ)(y' - Ay) - (y' - Ay)Q + O(X^{2m}), \\ &= O(X^m)O(X^m) + O(X^m)O(X^{m-1}) + O(X^{2m}) = O(X^{2m-1}). \end{aligned}$$

□

Comme d'habitude, l'application du lemme « Diviser pour régner » mène au résultat de complexité.

THÉORÈME 2 (Newton pour les systèmes différentiels linéaires). *L'algorithme* SolveHomDiffSys *calcule les N premiers termes d'une base de solutions de $Y' = AY$ en $O(\mathsf{MM}(r, N))$ opérations dans $\mathbb{A}$.*

La notation $\mathsf{MM}(r, N)$ représente la complexité du produit de matrices $r \times r$ de polynômes de degré au plus N ; des bornes non triviales ont été données au cours 7 :

$$\mathsf{MM}(r, N) = O(r^\omega N + r^2 \mathsf{M}(N)).$$

EXERCICE 1. L'exponentielle d'une série est obtenue en $O(\mathsf{M}(N))$ opérations par cet algorithme lorsque $r = 1$. Vérifier que la constante est meilleure que celle de l'algorithme du cours 3.

EXERCICE 2. Le cas d'un système *inhomogène* $Y' = AY + B$ où B est une matrice de séries s'obtient à partir de l'algorithme précédent par variation de la constante. Étudier les précisions requises dans les troncatures, et donner le résultat de complexité.

3. Cas particuliers

3.1. Équations différentielles linéaires à coefficients polynomiaux. Pour ces équations, la méthode des coefficients indéterminés donne une complexité linéaire en N. L'explication tient à une propriété classique : les solutions séries de ces équations ont des coefficients qui vérifient des récurrences linéaires. En effet, si $A = a_0 + a_1 X + \cdots$ est une série solution de

$$q_0(X)A^{(r)} + \cdots + q_r(X)A = 0,$$

alors en notant $[X^n]f(X)$ le coefficient de X^n dans la série $f(X)$, on a les relations

$$[X^n]f'(X) = (n+1)[X^{n+1}]f(X), \qquad [X^n]X^k f(X) = [X^{n-k}]f(X).$$

Par conséquent, l'extraction du coefficient de X^n de l'équation différentielle fournit une récurrence linéaire sur les a_n valide dès lors que $n \geq n_0 := \max_{0 \leq i \leq r} \deg q_i$.

Les résultats du cours 6 sur les récurrences linéaires à coefficients polynomiaux s'appliquent alors, et en particulier les N premiers coefficients d'une solution s'obtiennent en $O(drN)$ opérations si d est une borne sur le degré des q_i.

Le cas matriciel se traite de la même façon, la récurrence ci-dessus étant alors à coefficients matriciels. Les complexités sont de même nature, avec en outre la possibilité d'utiliser un produit rapide de matrices dans le cas du calcul d'une base de solutions. Les résultats correspondants sont donnés dans la table 1.

3.2. Équations différentielles linéaires à coefficients constants. Dans le cas où les coefficients de l'équation ou du système sont constants, la formule (1) s'applique encore et devient :

$$y(X) = \exp(XA)y(0).$$

Pour étudier les propriétés de cette solution, il est usuel de faire intervenir la forme de Jordan de la matrice, mais cette forme ne se prête pas facilement au calcul.

Un algorithme efficace est obtenu en exploitant la *transformée de Laplace formelle*. Si $S = s_0 + s_1X + s_2X^2 + \dots$ est une série, cette transformée est définie par

$$\mathcal{L}S = s_0 + 1!s_1X + 2!s_2X^2 + \dots.$$

Les troncatures de $\mathcal{L}S + O(X^N)$ et de la transformée inverse $\mathcal{L}^{-1}S + O(X^N)$ se calculent en N opérations. Cette transformation simplifie les systèmes différentiels linéaires à coefficients constants en les rendant linéaires non-différentiels. Ceci découle de

$$\begin{aligned}\mathcal{L}(y) &= \mathcal{L}((I + XA + \frac{1}{2!}X^2A^2 + \dots)y(0)) \\ &= (I + XA + X^2A^2 + \dots)y(0) \\ &= (I - XA)^{-1}y(0).\end{aligned}$$

Les vecteurs solutions ont donc des transformées de Laplace dont les coordonnées sont rationnelles. En outre, d'après les formules de Cramer, le numérateur et le dénominateur de ces fractions ont des degrés bornés par l'ordre r du système. L'algorithme suivant s'en déduit :

ConstantCoefficients(A, v, N)

Input : A in $\mathcal{M}_{r\times r}(\mathbb{A})$, $v \in \mathcal{M}_{r\times 1}(\mathbb{A})$.

Output : $Y = \sum_{i=0}^{N-1} Y_iX^i$ dans $\mathcal{M}_{r\times r}(\mathbb{A})[X]$ tel que $Y' = AY + O(X^{N-1})$.

1. Calculer les vecteurs $v, Av, A^2v, A^3v, \dots, A^{2r}v$;
2. Pour tout $j = 1, \dots, r$:
   (a) reconstruire la fraction rationnelle ayant pour développement en série $\sum (A^iv)_jX^i$;
   (b) développer cette fraction en série à précision X^N en $O(N\,\mathsf{M}(r)/r)$ opérations ;
   (c) reconstruire le développement de y à partir de celui de z, en $O(N)$ opérations.

L'étape 1 est effectuée par la méthode naïve en $O(r^3)$ opérations. Une méthode plus efficace, en $O(r^\omega \log r)$ opérations, est à la base d'un algorithme de calcul du polynôme minimal d'une matrice et sera présentée au cours 27. L'étape 2 (a) se ramène à un calcul d'algèbre linéaire en posant des coefficients indéterminés pour les numérateurs et dénominateurs. Là aussi le coût peut-être diminué en faisant appel au calcul d'approximants de Padé qui sera vu au cours 10, mais cette partie du calcul ne dépend pas de N. L'étape 2 (b) utilise l'algorithme de développement

de fractions rationnelles vu au cours 4, et c'est là que se concentre le gain en complexité.

4. Extensions

4.1. Composer se ramène à résoudre. Le seul algorithme du cours sur les calculs rapides de séries dont la complexité n'est pas quasi-optimale est l'algorithme de composition.

Dans les applications où l'on connaît une équation $\Phi(x, f, f', \dots, f^{(m)}) = 0$ vérifiée par la série f il est parfois possible de calculer la série $h = f \circ g$ efficacement *sans passer par l'algorithme de composition* en remarquant qu'elle vérifie une équation du même ordre. Cette équation est obtenue en remplaçant x par g dans Φ et en composant les dérivées. Si Φ ne faisait intervenir que des polynômes ou des fractions rationnelles, le résultat de cette opération a pour coefficients des séries, ce qui mène assez généralement à une complexité en $O(\mathsf{M}(N))$ opérations en utilisant les algorithmes de ce cours.

Exemple 1. Pour calculer le développement de $h = \tan(f)$, il est possible de calculer en $O(\mathsf{M}(N))$ ceux de $\sin(f)$ et $\cos(f)$ (via l'exponentielle) et de diviser, mais il est aussi possible d'observer que h vérifie l'équation $h' = 1 + h^2 f'$, et d'utiliser les méthodes ci-dessus. Il faut ensuite comparer les constantes dans les $O()$ (ou deux implantations !) pour déterminer laquelle des deux méthodes est préférable.

4.2. Systèmes non-linéaires. La linéarisation effectuée dans le cas des équations d'ordre 1 se généralise aux systèmes. L'énoncé devient alors le suivant.

Théorème 3. *Soient $\phi_1, \dots, \phi_r$ r séries de $\mathbb{A}[[X, Y_1, \dots, Y_r]]$, telles que pour tout r-uplet de séries $(s_1(X), \dots, s_r(X)) \in \mathbb{A}[[X]]^r$, les N premiers termes des $\phi_i(X, s_1, \dots, s_r)$ et de $\mathrm{Jac}(\phi)(X, s_1(X), \dots, s_r(X))$ puissent être calculés en $O(\mathsf{L}(\mathsf{N}))$ opérations dans $\mathbb{A}$. On suppose en outre que $\mathsf{L}(\mathsf{n})/\mathsf{n}$ est une suite croissante. Alors, le système différentiel*

$$\begin{cases} y_1'(t) = \varphi_1(t, y_1(t), \dots, y_r(t)), \\ \qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ y_r'(t) = \varphi_r(t, y_1(t), \dots, y_r(t)), \end{cases}$$

avec conditions initiales $(y_1, \dots, y_r)(0) = v$ admet une solution série formelle, et ses N premiers termes peuvent être calculés en

$$O(\mathsf{L}(\mathsf{N}) + \min(\mathsf{MM}(\mathsf{r}, \mathsf{N}), \mathsf{r}^2\mathsf{M}(\mathsf{N}) \log \mathsf{N})).$$

Le symbole $\mathrm{Jac}(\phi)$ désigne la matrice jacobienne : son coefficient sur la ligne i et la colonne j vaut $\partial\phi_i/\partial y_j$.

L'intérêt d'énoncer le théorème à l'aide de la fonction L est le même que dans le cas des équations.

Démonstration. Il s'agit d'une généralisation de la Proposition 1. La linéarisation de $\Phi : Y \mapsto Y' - \varphi(Y)$ au voisinage d'une solution approchée y s'obtient en écrivant :

$$\Phi(y + h) = \Phi(y) + h' + \mathrm{Jac}(\phi)(y)h + O(h^2).$$

L'équation à résoudre pour une itération de Newton est donc le système linéaire inhomogène

$$h' = \mathrm{Jac}(\phi)(y)h - (y' - \phi(y)).$$

Si h est un vecteur solution de ce système à précision $2m$ et y une solution de Φ à précision m, ces équations montrent que $y + h$ est solution à précision $2m$. Le cas non-linéaire est ainsi ramené au cas linéaire. □

Notes

Les résultats de la Section 1 sont tirés de [2]. Une bonne partie des résultats de ce cours est tirée de [1]. La technique de la section 2.2 dans le cas particulier de l'exponentielle d'une série est due à [3]. L'algorithme diviser pour régner de la section 2.1 se trouve au moins implicitement dans [4].

Bibliographie

[1] Bostan (Alin), Chyzak (Frédéric), Ollivier (François), Salvy (Bruno), Schost (Éric), and Sedoglavic (Alexandre). – Fast computation of power series solutions of systems of differential equations. In *SODA'07*. – *Toappear*.

[2] Brent (R. P.) and Kung (H. T.). – Fast algorithms for manipulating formal power series. *Journal of the ACM*, vol. 25, n° 4, 1978, pp. 581–595.

[3] Hanrot (Guillaume), Quercia (Michel), and Zimmermann (Paul). – The middle product algorithm I. *Applicable Algebra in Engineering, Communication and Computing*, vol. 14, n° 6, March 2004, pp. 415–438.

[4] van der Hoeven (Joris). – Relax, but don't be too lazy. *Journal of Symbolic Computation*, vol. 34, n° 6, 2002, pp. 479–542.