数式処理 *J.JSSAC (199X)*
Vol. XX, No. XX, pp. 1 - XXX

Research paper

微分作用素環用システム kan/sm1

高山 信毅

1 kan/sm1 とは?

Kan は数学用語の環 (かん, ring) のことである. 環は加減と乗算のさだまった代数系で, n 変数の多項式の全体は環の代表選手である. Kan は環を調べるためのソフトウエアである. 多項式環を扱う専門的なシステムは Macaulay(2), Singular, CoCoA などがあり, より一般的なシステムでも Mathematica, Maple, Reduce, MuPad, Axiom, Magma, Risa/Asir などにも多項式環を扱うためのさまざまな関数が用意されている. さて, 筆者は超幾何方程式に関連した微分作用素を研究対象としていた. その関係で kan は環の中でも特に微分作用素環を扱う機能が充実している. 多くの関数はアルゴリズムの開発, 計算機実験による数学の定理の発見などと同時に実装が進んできた. したがって世界初の実装が多い. Kan は Macaulay, Singular, CoCoA 等と同様 (Risa/Asir も似たところあり), 一点突破型の専門数学ソフトウエアであり, そこではアルゴリズム開発と数学への応用が混然一体としている.

さて kan/sm1 の sm1 の方は何かというと stackmachine 1 の略である. Kan/sm1 の設計と実装を始めた当初は, あたらしい数式処理言語を作りたいという野望があった. そのとき今で言う Java 仮想マシンのような構想があってユーザ言語と実行エンジンを分離しようというわけで実行エンジン部分が sm1 だったわけである. ちなみにこの野望はいまでもあるが意味のあるものを残すのはとても難しいことがわかった. というわけで残りの人生で実現できるかどうか疑わしい.

変質といえば kan/sm1 のサーバ版である `ox_sm1` は現在数学ソフトウエアの統合プロジェクトである OpenXM package プロジェクトにおいて, 各種ソフトウエアをつなぐための糊の役割をこなすようになっている. たとえば数学ワープロである texmacs から asir を呼び出すときは `ox_sm1` が texmacs のプロトコルと asir を呼び出すための OpenXM のプロトコルの翻訳を担当している. 微分作用素とはなんの関係もない仕事である. ユーザは sm1 を使っているとは思っていないだろう.

このあたりで, kan/sm1 の実行例をひとつあげておこう. 微分方程式には解空間の次元 (rank) という概念がある. たとえば f を未知関数とする単振動の方程式

$$(\partial_t^2 + 1)f = 0, \quad \partial_t = \frac{d}{dt}$$

の解空間は $\mathbf{C}$ 上のベクトル空間とみなせる. その次元は 2 であり, $\exp(\sqrt{-1}t)$, $\exp(-\sqrt{-1}t)$ が基底である. ちなみに基底を取り換えて $\sin t$, $\cos t$ にもできる. こちらの方が単振動として物理でおなじみであろう. この場合には解空間の次元 (rank) を得るには方程式の ∂_t についての次数をみるだけであるが, 次のような連立方程式はどうだろう.

$$(\partial_x^2 + \partial_y^2)f = 0,\ (\partial_x\partial_y - 1)f = 0, \quad \partial_x = \frac{\partial}{\partial x}, \partial_y = \frac{\partial}{\partial y}$$

ここで f は (x, y) の未知関数である. このような連立方程式の解の次元を勘定したり, 解を構成しようとすると, グレブナ基底の世話になる必要がある. たとえば kan/sm1 では次のように入力すると rank を計算する.

```
(cohom.sm1) run
[ [( Dx^2 + Dy^2)  (Dx Dy -1)] (x,y)] rank ::
```

答えは 4 である. ちなみに p, q を $p^2 + q^2 = pq - 1 = 0$ の解とするとき, $\exp(px + qy)$ が上の微分方程式系の解空間の基底である.

kan/sm1 の入門的解説は大阿氏の本 [6] の付録, 英語では本 [9] の付録にある. kan/sm1 の配布パッケージ自体に含まれている入門解説は
`http://www.math.kobe-u.ac.jp/OpenXM/Current/doc/kan96xx/ttt/index.html` からも閲覧できる.

以下数学的な話題に関しては細部に立ち入らない. 数学用語に関して不明な点があれば, たとえば大阿久氏の入門書 [6] などを適宜参照されたい. 以下はシステム内部の話題を歴史にそって述べる. 歴史をのべる関係上話題が私事にもわたる. 論文, 解説では私事は述べない方がよいという意見もある. しかしながら計算機システムの設計では “人間や他のシステムとの対話” という視点は極めて重要である. “対話” において普遍的な解答を求めるのは不毛な議論ではないかと思っている. ユーザを仮定してはじめて “対話” に関する議論が成立するのではないか? Kan/sm1 の開発においては, 一番のヘビーユーザは自分自身であった. したがって私事を述べることは全体像の理解に不可欠ではないかと思っている.

2 kan/sm1 Version 1 以前

Kan を作成する前は, PC9801 MSDOS の上の Reduce で微分作用素環の ブレブナ基底計算のプログラムを書いていた (1986 年の秋から 1987 年の始めにかけて). このプログラムは超幾何関数の近接関係式 (パラメータに関する漸化式) を計算機で導出するために開発したものである [1]. 近接関係式の問題とは関係ないが, Reduce で書いたプログラムで一番不満だったのは, 自前でパーサが用意できないので, たとえば $\partial_x x$ みたいな式を入力できないことであった. 微分作用素での計算では, $\partial_x x$ と入力すると, $x\partial_x x + 1$ と結果を返してきて欲しいが, このような機能を Reduce で実現する方法がよくわからなかった.

1988 年の秋に Zeilberger の 2 項係数や特殊関数の公式の計算機援用証明のプレプリントを読んだ [2]. $D_n = \mathbf{Q}\langle x_1, \ldots, x_n, \partial_1, \ldots, \partial_n\rangle$ を n 変数の微分作用素環とする. I を D_n の左

イデアルとする. この Zeilberger の論文を読んで $D_{n-1} \cap (I + \partial_n D_n)$ の生成元の計算 (積分計算とよぶ) が重要な問題であることを理解した. これはグレブナ基底が使えると思った. ここで $I + \partial_n D_n$ が左イデアルなら上の生成元の計算はよく知られたいわゆる消去法の問題である. つまり辞書式順序のグレブナ基底の計算をすればよい. しかるに上の例の場合は $\partial_n D_n$ が右イデアルであるので左イデアルと右イデアルが混合している. 別のアイディアが必要であろう. 1990 年の正月休みに停止性の判定ができない不完全なアルゴリズムながら公式証明等には有効な方法を思い付いた [10]. (こたつにもぐって毎日考えた. はたからみると猫みたいだったであろう.)

このような中 1989 年の夏 (平成元年夏!) 頃から, 全部 C で書いたものを作成しようと思い立って作成したプログラムが Kan の原型である. このプログラムには言語パーサはなにもない. 入力ファイルに微分作用素を書いておいて, プログラムのパラメータを変更することによりいろいろな計算をする. ところでどうして全部 C で書こうなどと面倒なことを考えたのかはっきりは思い出せないが, 次が主な理由であったように思う. (1) とりあえず自分で全部つくってみたかった. (2) 上のアルゴリズムの発見をしたので実装したかった. (3) C で全部書くと早くなるかもと思った. (4) 微分作用素をなんとか直接パースしたい.

(1) は理由にならないが研究者としては自然な心情であろう. 計算機は受動的に使っていてはつまらない. “こうしたい, ああしたい, これをなんとかしたい” と思うのが肝要だと思う. (4) “微分作用素をなんとか直接パースしたい” は入出力に関する些細なことかもしれないが, システム開発の重要な動機となった. じっさい作成してみて (4) はきわめて便利な機能であることが実証できたと思う.

3 kan/sm1 Version 1

上記プログラムを整理して 1991 年に公開した. これが kan の始めての公開版である. 積分計算以外にもいわゆる接続公式と D-加群の制限の実験的計算にも kan が活躍し始めた. 積分の裏である制限の計算は $I + x_n D_n$ の計算が基本である [3]. さて筆者の勤務する神戸大学でもこのころ学内 LAN が整備されはじめていた. 自然科学研究科に 1 台だけおいてあった高価な Sun ワークステーションで開発をおこないインターネットに ftp で公開した. いまでは当り前の環境であるが, Dos Extender で窮屈なメモリ空間で仕事をしないといけないパソコンからこの環境に乗り換えたら, もう元へはもどれなくなった.

さて計算毎にプログラムのパラメータを変更して, コンパイル, 実行というのは面倒である. というわけで, 1992 年の春休みを全部使い sm1 (stackmachine 1) なるユーザ言語を設計実装した. 同時に bignum やグレブナ基底計算プログラムも全面的に書き直し, さらに syzygy 計算, free resolution の計算機能を加えた. また野呂氏から Risa/Asir の利用している Boehm GC の話しを聞いたので Gargage Collector をこの GC にした. 1992 年は夏休み直前まで精力的に Kan/sm1 の開発が続く. 一段落したのが Kan/sm1 Version 1.920702 である.

その直後の 1992 年の夏にアメリカの Amherst カレッジで 1ヵ月にわたる計算代数幾何の夏の学校があった. ここで計算代数幾何システム Macaulay に出会う. また Kan/sm1 がこ

の夏の学校の夜の新人講演で国際デビューした年でもある. この計算代数幾何の夏の学校ではちょうどミラー対称性が流行りだしたころでもあり, パラメータ付き積分のみたす微分方程式の計算に興味を持つ人も多数いた [4]. というわけでそのような計算もこなす Kan/sm1 に興味をもってくれる人も何人かいてくれた. 悔しいがこのころはおもちゃのような問題しか解けなかった.

さて, sm1 は Postscript に似たスタック言語である. 何か計算をおこなうオペレータは一番最後にくる. たとえば "1 足す 2" は `1 2 add` と書く. これは 1 をスタックにつみ, 次に 2 をスタックにつみ, `add` がきたら, スタックからデータを二つ pop して, それを足して結果をまたスタックにつみなさいという意味である. どうしてこのような言語をユーザ言語としたかというとその理由は以下のとおり. (1) ちょうどこのころ, 神戸大学に NeXT ワークステーションが (キャノンの寄付により) 20 台あった. NeXT は Mathematica を標準装備している. 1991 年筆者はこれを利用して数学科の 4 年生に Mathematica のプログラミングを通年にわたり教えた. Mathematica の言語や GUI にはかなり衝撃をうけて, この方向でなにか新しいものを付け加えることは当分なにもできないような気がした. それよりも当分は Mathematica を入出力機構としてその裏で計算をするようなシステムを作ろうと思った. もちろんそのうち, Mathematica よりいい高級言語を考えるぞという野望もあった. (2) 裏で計算するには高級言語と機械語の中間みたいな言語がいいとおもった. 筆者の世代だと Pascal の P-code マシンがすぐ思い付く. (3) 裏で使うとしても独立しても使えないとデバッグにも不便だしねぇ. (4) スタックは計算機科学の大発明であると思う. これを生で利用できる言語があまりないのは残念である. 以上が, Postscript をベースにした逆ポーランド言語を使用した理由である.

4 kan/sm1 Version 2

1993 年の夏から 1994 年の春まで Cornell 大学に滞在した. Cornell 大学には Macaulay の開発者の Mike Stillman 氏やグレブナ基底と組合せ論に関して興味深い仕事をしている Bernd Sturmfels 氏がいた. 彼らの話しを聞くにつけ, Macaulay は一度きっちり勉強しておこうと思った. 1993 年の冬から 1994 年の春にかけて Macaulay のソースコードを読んでいろいろと勉強した. またこの時期の少し前から大阿久氏の大活躍も始まった. D-加群の特性多様体を計算するアルゴリズムの発表や Fuchs 型方程式の特性方程式の計算アルゴリズムなど本質的な進展があった. これらを効率よく計算したかったのであるが, Kan のグレブナ計算は効率が悪くまた Kan/sm1 の API (トップレベルの関数をこの文章では API とよぶ) がこれらのアルゴリズムの実装に容易な形になっておらず不満であった.

さて, 高速だと評判の高い Macaulay のソースコードを読んで, Macaulay の高速性はモノミアルの整数表現による項比較の高速化, それからつねに同次の多項式を扱うことによる Buchgerger アルゴリズムの効率化に支えられていることが理解できた [5]. Macaulay と同じように計算するにはどうすればいいか考えてみた. 1 週間ほど考えて微分作用素を同次化す

ればよいという結論に達した. つまり交換関係式を $\partial_x x = x\partial_x + 1$ から

$$\partial_x x = x\partial_x + h^2$$

と変更するのである. あとで $h = 1$ とおいてやれば微分作用素の計算を再現できる. h が同次化のための新しい変数である. これで Macaulay をちょっと変更すれば微分作用素環の計算もできるようにできるようになることがわかった. この時 Macaulay の高速性をそこなわないように微分作用素のかけ算の高速化について考察した. 微分作用素のかけ算を多項式のかけ算に帰着する Leibnitz 法則を利用し [6], それから項の比較をなるべく減らすようにデータ構造も工夫した [7]. この D-加群用の Macaulay のタイミングデータを "A benchmark test for Grobner basis systems of differential operators I" として翌年 `http://www.math.kobe-u.ac.jp/~taka/bench1.tex` で公開した. このデータは新しい実装をする人が, この程度の速度はでるはずだという目標になったと思う. D-加群用の Macaulay は現在でも `ftp.math.kobe-u.ac.jp/pub/MacaulayForD` から入手可能である. 現在普及しているマシンでコンパイル, 実行可能かは調べていない.

さて積分や制限の (近似) 計算を行うには Macaulay ではやはりいろいろと不自由があった.

そこでいままでの kan/sm1 のソースをほぼ全部捨てて, 1994 年の夏休みに kan/sm1 を書き直すことにした. これが kan/sm1 version 2 である. 設計の基本方針は以下のとおり [8]. (1) 同次化した微分作用素環を基本にする. (2) 順序は weight vector で決めるのを基本とする. (Weight vector に負の成分があっても同次化した微分作用素環なので計算はかならず終了する.) (3) 使えるライブラリがあれば使う. たとえば bignum は自分で実装せずに gmp を使う.

筆者はクーラはあんまり使わないのであるが, この年の夏はクーラーの付いてる部屋でプログラミングをやり続けたのを覚えている. 実装は夏休みだけではおわらず 9 月までもつれこんだ. 十分安定を確かめて 1 年後に最初に公開したのが `kan.Sep27.1995.tgz` である. 上記の基本設計は現在まで変更がない. この時のコードは現在の kan/sm1 に引き継がれている. なお weight vector の考えの全面採用は Sturmfels 氏のグレブナ基底数学の影響である [9].

さて 1995 年の 1 月に阪神淡路の震災がおきた. 偶然ではあるがこの年の前後で D 加群の計算数学に関してもさまざまなことが大きく変わる. 開発マシンを PC unix に変更したのも, kan/sm1 version 2 の公開をしたのも 1995 年前後であるが, 1996 年の 9 月 大阿久氏のプレプリント "Algorithms for b-functions, restrictions, and algebraic local cohomology groups of D-modules" [5] がメールで送られて来た. これは大ニュースであった. 積分計算, 制限計算をおこなう停止性条件付きの完全なアルゴリズムの出現である. この論文で発表されたアイディアが, その後の多くの D-加群のアルゴリズムに基礎となった [10]. このアルゴリズムは新しい設計の kan/sm1 で比較的容易に実装できた. 同次化した微分作用素環および weight vector の考えによりいろいろなアルゴリズムの実装に柔軟に対処できるようになったのである.

ところで同じ頃 Java がポピュラーになりつつあった. Java の設計におおいに刺激をうけて, 新しい数式処理言語という野望にトライすることにした. 1996 年頃には PVM で分散処理対応にしたり, 野呂氏に協力を依頼して asir との接続をおこなった. 1996 年の後半から

1997 年始めにかけて kan/sm1 の辞書を木構造に変更してオブジェクト指向な kan/sm1 に改造した [11]. これらを準備として 1997 年の春休みに kan/k0 という Java に似た高級言語を設計実装した [12]. これを実行する仮想マシンが kan/sm1 である. 1997 年の前期には kan/k0 の講義を大学院で毎週やり, 言語の設計, 実装の改善を試みた. しかし結果としては, 設計, 実装に関する新しい知見は得られなかったと思う. というか, Java のクラス定義の仕組みが数学アルゴリズムの記述に有効かどうかという議論はまだ本格的におこなわれていないと思う. この議論は Axiom や Gap, Macaulay2, Magma のオブジェクト指向機能と比較しながら行うべきであろう. 現在新しい数式処理言語の方向の研究は中断している.

しかしながら, ネットワーク上の分散処理技術については OpenXM プロジェクトとして現在研究が進展している [13].

5 kan/sm1 Version 3 と現在

大阿久氏の制限計算アルゴリズムの発表の後, D 加群に関する新しいアルゴリズムの発表や, D 加群のアルゴリズムを援用した計算機実験が以前より盛んになった. また 1996 年, 97 年は $\mathcal{A}$-超幾何方程式の rank の計算実験, indicial 多項式 (b-function とか特性多項式とよぶこともあり) の計算実験も kan/sm1 を用いて盛んにおこなわれた. これらの計算実験をもとにした理論展開の集大成が文献 [9] である. このあたりの事情, その後の発展, OpenXM プロジェクトについては, 他の文献でも紹介されているのでこの文章では省略したい.

さて $\mathcal{A}$-超幾何方程式の研究と同時に D 加群に関する新しいアルゴリズムがどんどん発表されて, それらを kan/sm1 のユーザ言語で大阿久氏と実装していってライブラリが蓄積された. ひと区切りということで 2000 年に version up したのが kan/sm1 version 3 である. Version 3 のシステムの面からみた寄与は, D-加群のシステム用の API の体系をひとつ提案したということであろう [14]. ここで API は関数の仕様の集まりを意味する. この API を改良したものが H.Tsai 氏と A.Leykin 氏による Macaulay 2 の D-加群ライブラリの API である (と思う).

Version 4 をいつ出すかは未定である. 現在, 局所的な数学をやるための tangent cone algorithm およびそれを用いた各種関数が加えられている. また, (1) グレブナエンジン自体の改良. (2) OpenXM の “糊” としての役割をよりよくするように工夫する. (3) よりプログラムしやすいスタックマシン言語に改良., なども課題である. 現在逆ポーランド系の記法をもつ言語はほぼ絶滅状態なので (3) はだれも取り組んでいない興味深い課題だろうと思っている.

この文章では kan/sm1 に関する歴史をふりかえってみた. いまから新しいシステムを開発したい, 新しいライブラリを開発したい, という人の参考になればうれしい.

6 注釈

1. 近接関係式とグレブナ基底の話題については数学セミナーの記事 [12] を参照. 近接関係式を求めるアルゴリズムのやさしい解説は超幾何関数とパンルベ関数の本 [1] の Chapter 2, 41–47 にある.

2. この話題に関しては現在本 [8] としてまとまっている. 予備知識はあまり必要としない本だが深いアイディアを説明している.

3. D-加群の制限と制限の概念は, 理論全体のかなめである. たとえば大阿久氏の本の 5 章がその入門にあてられている.

4. ミラー対称の理論はもともと理論物理的な考察で発見された. 物理はよく知らないが, 数学的成果を一つ述べる. $\mathbf{P}^4$ の十分一般の 5 次超曲面に含まれる有理曲線の次数ごとの本数を数える問題を考えよう. この問題を直接解くのは難しい. この 1992 年の夏の学校で A.Stromme 氏が交点理論を利用して本数を勘定するアルゴリズムとその Maple での実装を紹介した. ミラー対称の理論が正しいと仮定すれば, 5 次超曲面のミラー対称な多様体を考えその周期写像 (代数関数のパラメータ付き積分!) の満たす微分方程式の解の比をべき級数展開すると答えの母関数がでる. こちらの方は Candelas らによりすでに計算されていた. ミラー対称の理論を研究していた S.Katz 氏もこの夏の学校にきていた. Katz 氏らが中心となり, Stromme 氏のプログラムで 次数 1, 2, 3 の場合を計算して本当に正しいか確かめることができて, 出席者が盛り上がっていたと記憶する.

5. Buchberger アルゴリズムの改良, 高速化の詳細については現在は野呂氏による著書 [3] および 野呂-横山の教科書 [4] が (英語の本を含めて) もっともよくまとまった文献である. 実装に関する立場からは [2] の Hans Schoneman 氏の Singular の内部構造に関する論文が素晴らしい.

6. 微分作用素のかけ算に Leibnitz rule を用いる方法は野海氏や大阿久氏が始めて導入した. この方法が計算量の観点からみて有利であるという考察は論文 [11] を参照.

7. この部分は D 加群用の Macaulay と kan/sm1 で共通である. kan/sm1 のソースコードの `poly3.c`:
`http://www.math.sci.kobe-u.ac.jp/cgi/cvsweb.cgi/OpenXM/src/kan96xx/Kan/poly3.c`
を参照.

8. (1) 同次化して扱うかどうかは 大域変数 `Homogenize` で制御している. (2) kan/sm1 には weight vector で順序を決めるマクロが用意されている. このマクロが順序を指定する基本である. たとえば
`[[(Dx) 1 (x) -1]] weight_vector` と宣言したとしよう. このときモノミアルの順序は, ∂_x に重み 1, x に重み -1 をつけた部分順序の細分となる.

9. $w \in \mathbf{R}^n$ を weight vector とする. $\prec$ を n 変数多項式のモノミアルに対して定義されたモノミアル順序とする. このとき n 変数多項式のモノミアルに対して w を用いて全順序を $\prec_w$ を次のように定義する.

$$x^\alpha \prec_w x^\beta \Leftrightarrow \alpha \cdot w < \beta \cdot w \text{ or } (\alpha \cdot w = \beta \cdot w \text{ and } x^\alpha \prec x^\beta)$$

グレブナ基底の入門書ではこのような順序は練習問題においやられている場合が多い. 我々の理論展開ではこのような順序が基本中の基本である.

10. 原論文以外に [6] に解説がある. [9] の Chapter 5 にもこのアルゴリズムの初等的な解説がある.

11. kan/sm1 の primitive `sendmsg`, `setcontext`, `newcontext` などが名前空間辞書を木構造にするための仕組みである. 名前空間辞書を木構造としてオブジェクト指向を実現した.

12. kan/k0 で複素数を定義した例.

```
class Complex extends Object {
  local re, /* 実部 */
        im; /* 虚部 */
  def new2(a,b) {
    this = new(super.new0());
    re = a;
    im = b;
    return(this);
  }
  def real() { return(re); }
  def imaginary() { return(im); }
  def operator add(b) {
    return( new2(re+b.real(), im+b.imaginary()) );
  }
}
```

kan/k0 は sm1 のコードを吐き出すコンパイラを持つ. 同次化された D での極小自由分解アルゴリズム [7] の実装は kan/k0 で書かれている.

13. OpenXM プロジェクトに関しては `http://www.openxm.org` の文献とソフトウエアシステムが基本的. [2] の野呂氏の論文も簡潔な入門記事.

14. `http://www.math.kobe-u.ac.jp/OpenXM/Current/doc/kan96xx/onlinehelp/index.html` に kan/sm1 の primitive と macro 名一覧がある. API を決めることは決して明らかな仕事でない. 上手に決められた API により, 新しいアルゴリズムがより容易に実装されるようになる.
現在微分作用素を扱えるシステムはいろいろあり, 開発者が重点的に研究した面をみれば kan/sm1 よりはるかに優れている. 以下, 筆者の知る限りの微分作用素環を扱えるシステムをあげておく: D-module package on Macaulay 2 (A.Leykin, H.Tsai), Mgfun on Maple (F.Chyzak), Plural (V.Levendovskyy, Singular ベース), Risa/Asir (野呂), Yang package on Risa/Asir (小原),

参 考 文 献

[1] Iwasaki, K., Kimura, H., Shimomura, S., Yoshida, M.: *From Gauss to Painlevé*, Vieweg, 1991.

[2] Joswig, M., Takayama, N. (Editors): *Algebra, Geometry, and Software Systems*, Springer (2003).

[3] 野呂正行: 計算代数入門, Rokko Lecture Series in Mathematics, No. 9 (2000). http://www.math.kobe-u.ac.jp/Asir/ca.pdf

[4] 野呂正行, 横山和弘: グレブナー基底の計算, 基礎篇－計算代数入門, 東大出版会 (2003).

[5] Oaku, T.: Algorithms for b-functions, restrictions, and algebraic local cohomology groups of D-modules. Advances in Applied Mathematics **19** (1997), 61–105.

[6] 大阿久俊則: D 加群と計算数学, 朝倉書店 (2001).

[7] Oaku, T., Takayama, N.: Minimal Free Resolutions of Homogenized D-modules. Journal of Symbolic Computation 32 (2001), 575–595.

[8] Petkovsek, M., Wilf, H., Zeilberger, D.: $A = B$ 等式証明とコンピュータ, AK ピータース・トッパン (1996).

[9] Saito, M., Sturmfels, B., Takayama, N.: *Gröbner Deformations of Hypergeometric Differential Equations*, Algorithms and Computation in Mathematics **6**, Springer, 2000.

[10] Takayama, N.: An algorithm of constructing the integral of a module — an infinite dimensional analog of Gröbner basis, Proceedings of the international symposium on symbolic and algebraic computation, Edited by Watanabe and Nagata, ACM press and Addison-Wesley (1990), 206-211.

[11] Takayama, N.: Algorithm finding recurrence relations of binomial sums and its complexity. Journal of Symbolic Computation 20 (1995), 637-651.

[12] 高山信毅: 計算代数, 数学セミナー, 2002 年 9 月号 (コンピュータによる数学の進化), 25–29.