7. Lauricellaの超幾何関数F_Dの変換公式と Garnier系 …… 110
東大・教養　木村　弘信 (Hironobu Kimura)
8. 有限モノドロミー群をもつ一般化された超幾何方程式 …… 123
都立大・理　佐々井　崇雄 (Takao Sasai)
9. 岩野・渋谷理論の展開 …… 140
大分大・工　大河内　茂美 (Shigemi Ohkohchi)

微分差分作用素環用 Gröbner 基底パッケージの O.D.E. section 問題への応用

徳島大学・総合科学部　高山信毅
(Nobuki Takayama)

本稿で解法を紹介する問題Sの出発点となった問題は次の問題である。

問題S。　多変数超幾何関数と Hypergeometric type O.D.E の関係を調べよ。

(多変数超幾何関数については [Erd1] を、Hypergeometric type O.D.E. については [Oku] を参照。)

二変数関数 $f(x,y)$ を (yを定数とみなし) x のみの関数とみたものを $f(x,y)$ の y-section と呼ぶことにする (resp. xを定数とみて y のみの関数とみたものを x-section)。たとえば、Appellの関数 F_1 の y-section, x-section は、Jordan-Pohhammer の O.D.E. をみたす関数である。また、F_2, F_3 の section は、Goursat-Sasai の O.D.E. をみたす関数となっている。Jordan-Pohhammer の O.D.E. は3階の Hypergeometric system へ変換できるし、Goursat-Sasai の O.D.E. は4階の Hypergeometric system へ変換できる。つまり、F_1, F_2, F_3 は Okubo のアクセサリパラメータ0の

Hypergeometric system の分類表へ([Oku]) ぴったりおさまるのである。では、他の多変数超幾何関数についてはどうであろうか?というのが問題Sの由来である。(この問題については、[Suz-Oho]を参照。) これらの例のような、よい関係が両者の間にあれば、多変数超幾何関数のモノドロミ群の構造が Hypergeometric system をとおして、よく見えてくるわけである。

では問題Sを少し考えてみよう。まず考えるべき関数を、Horn の表にありかつ確定特異点のみをもつ関数に限る([Erd1])。$F_1 \sim F_4$, $G_1 \sim G_3$, $H_1 \sim H_7$ のうち、独立変数の rational な変換で、F_1, F_2 へ reduce しないのは、F_4, H_1, H_5 のみであるから([Erd2])、最も興味深いのは、F_4, H_1, H_5 である。まず F_4 の y-section を実験的に求めてみる。実験であるから、パラメータ $\alpha, \beta, \gamma, \gamma'$ にも適当な値

$\alpha = 1/2$, $\beta = 1/5$, $\gamma = 9/7$, $\gamma' = -2/9$ を入れてやる。§1の算法Iにより、y-section が求まる。結果は fig1。ここで X0 が x, X1 が y, D0 が $\frac{\partial}{\partial x}$, * は"かける"の意味である。$\left(\frac{\partial}{\partial x}\right)^4$ の係数が x の5次式であり、これは Hypergeometric type O.D.E. ではない。fig1. の O.D.E. を L_0 とおき、fig2. の5階の O.D.E. を L_1 とおく。L_1 は、§2の算法IIによって計算した y-section である。このとき、

$$L_1 = \left[\frac{1}{79597x_0 + 147223x_1 - 147223}\left(180x_0\frac{\partial}{\partial x_0} + 173\right)\right] L_0$$

が成り立つ。L_1 は Hypergeometric type O.D.E. であるが、F_4 と L_1 は

---71442000.-----   たがいに素.

```
                  2                                 2
396900*(X0  - 2*X0*X1 - 2*X0 + X1  - 2*X1 + 1)*(79597*X0 + 147223*X1 -

               4   2
   147223)*D0  *X0

 - 1260*((625368263*X1 + 804461513)*(X1 - 1)*X0 - 14*(11895489*X1 -

                        2                         3              3    3
             59949335)*X0  - 212001120*(X1 - 1)  - 246830297*X0 )*D0 *X0

                   3                     2                        2
(735583815547*X0  + 1143525132106*X0 *X1 - 2309462863673*X0  -

                          2
 1433153630853*X0*X1  - 165966988327*X0*X1 + 1599120619180*X0 +

                  3                 2
 171720907200*X1  - 515162721600*X1  + 515162721600*X1 - 171720907200)*

   2
 D0

                         2
+7*(52460631566*X0  + 132161838767*X0*X1 - 164756093894*X0 - 50053763133

     2
  *X1  - 22904798607*X1 + 72958561740)*D0

+97216*(79597*X0 + 300403*X1 - 300403)
--------------------------------------------------------------/
```

fig 1.

```
                                           2      2     5   3
 - 71442000*(2*(X1 + 1)*X0 - (X1 - 1)  - X0 )*D0 *X0
```

$2xy + 2x - y^2 + 2y - 1 - x^2$.

```
                                        2            2    4   2
 - 56700*(29405*(X1 + 1)*X0 - 9491*(X1 - 1)  - 19915*X0 )*D0 *X0

                                                     2             2
 - 180*(7*(4019626*X1 + 4099223)*X0 - 4724640*(X1 - 1)  - 28625702*X0 )

    3
 *D0 *X0

                2                                                   2
+(7419521543*X0  - 3973785543*X0*X1 - 4349562980*X0 + 201787200*X1  -

                                     2
  403574400*X1 + 201787200)*D0

+7*(353788414*X0 - 58817583*X1 - 85732740)*D0

+34317248
--------------------------------------------------------------/
```

fig2.

よい関係にない。なぜなら、L_1 は hypergeometric type O.D.E. だが、reducible であり、F_4 のモノドロミの計算には、L_1 は役に立たないからである。以上の観察より次のことがわかった。

"問題S は、単に O.D.E. section を求めるだけでは解けない。"

しかしながら、O.D.E. section を求めるための算法をいろいろ考えておくことは、問題Sやその他の問題のいろいろな場面で有利であろう。このような動機により、本稿では、次の問題W の解法を紹介することにする。

問題W. (1) O.D.E. section 全体を求める算法を作れ。

(2). O.D.E. section 全体は単項イデアルとなるが、イデアルの生成元を求めるのがむずかしい場合は、生成元でなくとも、とにかくO.D.E. section をひとつ見つけよ。//問題

O.D.E. section を正確に定義しておく。

$$A := \mathbb{C}(x,y)\left[\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right] \qquad \text{微分作用素環}$$

A の左イデアル J が 0-次元イデアル であるとは、A/J を $\mathbb{C}(x,y)$ 上の線型空間とみたとき、

$$\dim_{\mathbb{C}(x,y)} A/J < +\infty$$

となることである。

例。

$$L_1 = x\frac{\partial}{\partial x}\left(x\frac{\partial}{\partial x}+y\frac{\partial}{\partial y}+\gamma-1\right) - x\left(x\frac{\partial}{\partial x}+y\frac{\partial}{\partial y}+\alpha\right)\left(x\frac{\partial}{\partial x}+\beta\right)$$

$$L_2 = y\frac{\partial}{\partial y}\left(x\frac{\partial}{\partial x}+y\frac{\partial}{\partial y}+\gamma-1\right) - y\left(x\frac{\partial}{\partial x}+y\frac{\partial}{\partial y}+\alpha\right)\left(y\frac{\partial}{\partial y}+\beta'\right)$$

とおく。$L_1f=L_2f=0$ の解は Appell の F_1 である。L_1, L_2 により生成される A のイデアルを J とおく。このとき、

$$\dim_{\mathbb{C}(x,y)} A/J = 3$$

である。(パラメータ $\alpha,\beta,\gamma,\beta'$ には値が確定しているものとする) //例

定義。 J を A の 0次元イデアルとする。このとき、$\mathbb{C}(x,y)\left[\frac{\partial}{\partial x}\right]$ のイデアル

$$J \cap \mathbb{C}(x,y)\left[\frac{\partial}{\partial x}\right]$$

を J の O.D.E. y-sections のなすイデアル と呼ぶ。このイデアルの要素を J の O.D.E. y-section と呼ぶ。上のイデアルは単項イデアルとなるので、その生成元を、O.D.E. sections の生成元(generator) と呼ぶ。//定義

定理。 J を A の 0-次元イデアルとする。$J \neq (0)$ なら、

$$J \cap \mathbb{C}(x,y)\left[\frac{\partial}{\partial x}\right] \neq (0)$$

すなわち、自明でない O.D.E. section が必ず存在する。//定理

注意。 例外的な場合を除き、

$$\dim_{\mathbb{C}(x,y)} A/J = \dim_{\mathbb{C}(x,y)} \mathbb{C}(x,y)\left[\frac{\partial}{\partial x}\right] \Big/ J \cap \mathbb{C}(x,y)\left[\frac{\partial}{\partial x}\right]$$

となる。つまり、元の方程式系の rank と ODE section の generator の rank は等しい。また、左辺 ≧ 右辺 は、常に成り立つ。//注意

(本稿は、二変数関数について述べているが、n変数の場合もほぼ同様に扱える。)

§1. 微分作用素環のGröbner基底による算法。

$\mathcal{A}$のイデアルのGröbner基底を求める算法(Buchberger算法)を、lexicographic orderで用いれば、O.D.E. sectionのgeneratorは、Gröbner基底のひとつの要素として必ず求まる。

<u>参考。</u> [Fur]はこの算法(多項式環)に対するチュートリアルである。$\mathcal{A}$のイデアルやWeyl Algebraのイデアルの Gröbner基底は、いろいろな問題に対して有効である。特殊関数のcontiguity relationの導出と微分差分作用素環のGröbner基底一般については[Tak1]及びその参考文献を見よ。Weyl algebraのGröbner基底、Hilbert多項式の計算、cohomologyの計算等については[Gal]を見よ。特殊関数を含む等式や二項係数の公式の自動証明システムについては、[Zei], [Tak2]を見よ。また、Gröbner基底の概念は、ソリトン理論などでも用いられた[Nou]。// 参考.

算法Iは問題ひ(1)の解である。(多項式環については、Buchbergerのころからよく知られている算法)

<u>算法I.</u> (本質的に、Buchbergerによる)

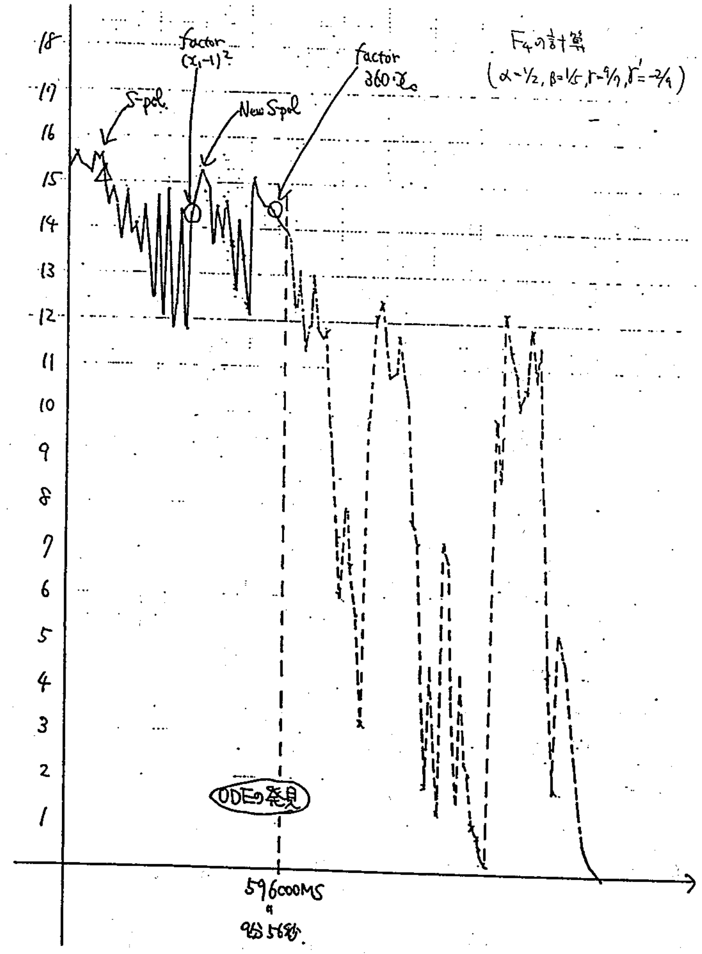
$\mathcal{A}$のイデアルIのGröbner基底をlexicographic orderで求めよ。基底中にODE sectionsのgeneratorがある。// 算法

この算法の詳細については、付録1のREDUCEによるアルゴリズムの記述を見よ。付録1は、この算法の解説兼プロトタイプ実現用のREDUCE 3.2, 3.3用のプログラムである。

<u>例.</u> Hornの表のΦ_2の微分方程式([Erd1])のlexicographic orderによるGröbner基底は、

O.D.E. sectionsのgenerator

$$
\left\{
\begin{array}{l}
-(x-y)x\dfrac{\partial^3}{\partial x^3}-\left[(x-y)\gamma+(y+2)x-x^2-y+\beta x+\beta' y\right]\dfrac{\partial^2}{\partial x^2} \\
\qquad -(\beta+1)(\gamma-2x+y)\dfrac{\partial}{\partial x}+\beta(\beta+1), \\
y\beta\dfrac{\partial}{\partial y}-(x-y)x\dfrac{\partial^2}{\partial x^2}-\left[(\gamma-x)(x-y)+\beta' y\right]\dfrac{\partial}{\partial x}+(x-y)\beta
\end{array}
\right.
$$



<u>fig 3.</u>

である。//例

この算法は、どんな 0-次元イデアルに対しても通用する、一般性が高い算法である。しかし、空間計算量がきわめて大きいのが欠点である。式の中間膨張がおきてメモリ爆発がおきてしまう。fig3.は計算の進行にともなって空きメモリ量がどう変化するかのグラフである。中間膨張の様子がわかる。

例.

関数名	F_1	F_2	F_3	F_4
ODE sectionが一般のパラメータに対して、算法Ⅰで求まるか?	○	○	○	×

計算は、5MByteのメインメモリをもつ VAXstation/Ⅱ 上の、REDUCE3.3 でおこなった。//例

§2. 微分差分作用素環の Gröbner 基底を用いる算法

この節で述べる算法Ⅱは問題Ⅳ(2)の解である。ただし、多変数超幾何関数にしか通用しない。また、O.D.E. sections の generator が必ずしも求まるとは限らない。一般に generator より高階になってしまう。(注。もし、一般のO.D.E.の解の contiguity relation を求める算法が作れれば、この算法は、一般の関数に対して通用する)。

算法Ⅱの基本的 idea を述べる。Hypergeometric function を、

$$F(x,y)=\sum_{k=0}^{\infty} f_k(y)\,C_k\,x^k$$

$f_k(y)$: Gaußの Hypergeometric function の族。

C_k : Γ-関数の積・商。

と書く。$f_k(y)C_k$ が k についてみたす差分方程式を求めて、$F(x,y)$ のみたす x についての O.D.E. を計算すれば、それが求めるものである。

例.

$$\underbrace{F_2(x,y)}_{F}=\sum_{k=0}^{\infty} \underbrace{F(\alpha+k,\beta',\gamma';y)}_{f_k(y)}\;\underbrace{\frac{(\alpha,k)(\beta,k)}{(\gamma,k)(1,k)}}_{C_k}\;x^k$$

//例.

さて、f_k, C_k についての仮定より、f_k, C_k は次のような関係式をみたすはずである。

$$\begin{cases} C_{k+1}=\dfrac{s_1(k)}{r_1(k)}C_k \\[2ex] C_{k+2}=\dfrac{s_2(k)}{r_2(k)}C_k \end{cases} \qquad \text{ここで、} s_i, r_i\in\mathbb{C}[k]$$

$$\begin{cases} H(k)f_k=f_{k+1} \\ L(k)f_k=0 \end{cases} \qquad \text{ここで、} H,L\in\mathbb{C}(k,y)\left[\frac{\partial}{\partial y}\right]。$$

以下、E_k を k についてのシフト作用素

$$E_k f_k = f_{k+1}$$

とする。

算法Ⅱ.

入力。 L, H, s_1, r_1, s_2, r_2

出力。 $F(x,y)$ を x のみの関数とみたときの、$F(x,y)$ がみたす O.D.E.。

Step1. $E_k - H(k)$, $L(k)$ の Gröbner 基底を $\mathbb{C}(y,k)\left[\frac{\partial}{\partial y}, E_k\right]$ で、lexicographic order $\frac{\partial}{\partial y} \succ E_k$ で求める。この基底は k についての2階の差分作用素を必ずふくむ(算法Ⅰの類似)。

〔もしくは、$f_k(y)$は、高々超幾何関数（Gaußの）だから公式集の公式より適当に導く。しかしこれは3変数以上では通用しないので、（3変数以上では）Gröbner基底を用いる必要がある（[Tak1]）〕

この差分作用素を、

$$G_1 = a_2 E_k^{\ 2} + a_1 E_k + a_0$$

とおく。ここで、$a_i \in \mathbb{C}(k, y)$。 （注 $G_1 f_k = 0$ である。）

Step 2.

$$a_2 := a_2 * \frac{r_2}{s_2} * R; \quad a_1 := a_1 * \frac{r_1}{s_1} * R; \quad a_0 := a_0 * R;$$

（Rは、a_0, a_1, a_2を多項式かつ$\gcd(a_0, a_1, a_2) = 1$とするために適当にかける因子）

Step 3.

$$L := a_2\Big|_{k = x\frac{\partial}{\partial x} - 2} + x * \left(a_1\Big|_{k = x\frac{\partial}{\partial x} - 1}\right) + x^2 * \left(a_0\Big|_{k = x\frac{\partial}{\partial x}}\right)$$

Lを出力。 //算法

例　Appell F_2 の場合

入力：
$$H(k) = \frac{1}{\alpha + k}\left(y\frac{\partial}{\partial y} + 1\right),$$
$$L(k) = y(1-y)\frac{\partial^2}{\partial y^2} + [\gamma' - (\alpha + k + \beta' + 1)y]\frac{\partial}{\partial y} - (\alpha + k)\beta',$$
$$s_1(k) = (\alpha + k)(\beta + k),$$
$$r_1(k) = (\gamma + k)(1 + k),$$
$$s_2(k) = (\alpha + k)(\alpha + k + 1)(\beta + k)(\beta + k + 1),$$
$$r_2(k) = (\gamma + k)(\gamma + k + 1)(1 + k)(2 + k).$$

1.
$$a_2 = (k + \alpha + 1)(y - 1),$$
$$a_1 = -\{(k + \alpha + 1 - \beta')y - 2(k + \alpha + 1) + \gamma'\},$$
$$a_0 = -(k + \alpha + 1 - \gamma').$$

2. $R = (\alpha + k)(\beta + k)(\beta + k + 1)$ ととる。
$$a_2 = (y - 1)(\gamma + k)(\gamma + k + 1)(1 + k)(2 + k),$$
$$a_1 = -\{(k + \alpha + 1 - \beta')y - 2(k + \alpha + 1) + \gamma'\}(\gamma + k)(1 + k)(\beta + k + 1),$$
$$a_0 = -(k + \alpha + 1 - \gamma')(\alpha + k)(\beta + k)(\beta + k + 1).$$

3.
$$\begin{aligned} L = {} & (y-1)\left(x\frac{\partial}{\partial x} - 2 + \gamma\right)\left(x\frac{\partial}{\partial x} - 2 + \gamma + 1\right)\left(x\frac{\partial}{\partial x} - 2 + 1\right)\left(x\frac{\partial}{\partial x} - 2 + 2\right) \\ & - x\left\{\left(x\frac{\partial}{\partial x} - 1 + \alpha + 1 - \beta'\right)y - 2\left(x\frac{\partial}{\partial x} - 1 + \alpha + 1\right) + \gamma'\right\}\left(x\frac{\partial}{\partial x} - 1 + \gamma\right)x\frac{\partial}{\partial x}\left(x\frac{\partial}{\partial x} + \beta\right) \\ & - x^2\left(x\frac{\partial}{\partial x} + \alpha + 1 - \gamma'\right)\left(x\frac{\partial}{\partial x} + \alpha\right)\left(x\frac{\partial}{\partial x} + \beta\right)\left(x\frac{\partial}{\partial x} + \beta + 1\right) \end{aligned}$$

出力：　L。 //例.

算法IIをREDUCEでインプリメントし、Horn list [Erd1] の関数 $F_1 \sim F_4$, $G_1 \sim G_3$, $H_1 \sim H_7$ の O.D.E. y-section および $H_1 \sim H_7$ の O.D.E. x-section を計算した。$F_1 \sim F_4$, $G_1 \sim G_3$ は x, y について対称なので、x-section は計算しない。

計算結果。

使用計算機、言語：　VAX station II/RC（メインメモリ5MByte）。REDUCE 3.3。
O.D.E. section のデータは約800K byte あるため、ここには掲載しない。section はすべて Hypergeometric type ODE（Fuchs型かつ、最高次の微分の係数式の次数が、方程式のランクに等しい）である。

関数	方程式のランク	算法IIによるODE sectionのランク	特異点 (下の式=0 と∞)
F_1	3	3	$xy(x-1)(y-1)(y-x)$
F_2	4	4	$xy(x-1)(y-1)(x+y-1)$
F_3	4	4	$xy(x-1)(y-1)(xy-x-y)$
F_4	4	5	$xy(x^2+y^2-2xy-2x-2y+1)$
G_1	3	4	$xy(x+y+1)(4xy-1)$
G_2	3	3	$xy(x+1)(y+1)(xy-1)$
G_3	4	4	$xy(27x^2y^2-18xy-4x-4y-1)$
H_1	4	5	$xy(x-1)(y^2-4xy+2y+1)$
H_2	4	4	$xy(x-1)(y+1)(xy-y-1)$
H_3	3	6	$xy(4x-1)(y^2-y+x)$
H_4	4	5	$xy(4x-1)(y^2-2y-4x+1)$
H_5	4	6	$xy(27y^2x-36xy-y+16x^2+8x+1)$
H_6	3	5	$xy(4x+1)(y^2x-y-1)$
H_7	4	5	$xy(4x-1)(4xy^2-y^2-2y-1)$
H_1-x	4	5	
H_2-x	4	4	
H_3-x	3	3	
H_4-x	4	4	
H_5-x	4	5	
H_6-x	3	3	
H_7-x	4	4	

計算結果

これからの課題。

1. メモリ爆発をおこさないような、Gröbner基底を求める算法を開発すること。

2. 微分作用素のかけ算の高速算法を開発すること。

3. 人間と対話的に処理をすすめやすいよう、数式をみやすいようにかっこづけて表示するための算法を開発すること。(何とおりか表示してくれるか、または、問題に即した意味理解をして表示してくれる。)

§. 付録1. $k(x_0, x_1, \cdots)\left[\frac{\partial}{\partial x_0}, \frac{\partial}{\partial x_1}, \cdots\right]$ 用Gröbner基底計算プログラム。

注. 解説版兼プロトタイプ版のため単純だが、計算速度がおそい。またREDUCE 3.2, 3.3 兼用のため冗長な部分がある。実用的パッケージを作るには、S-lisp を用い、かつ、微分作用素をどう計算機の中で表現するか(データ構造)まで、考えないといけない。cf. REDUCE 3.3 の groebner.red のソース。

注. このプログラムは REDUCE 3.3, 3.2 で動作する。disp() は、3.3 用、olddisp() は 3.2 用である。なお Franz Lisp にのせている一部の REDUCE では動作しない。

0. Order は、$\cdots > \frac{\partial}{\partial x_2} > \frac{\partial}{\partial x_1} > \frac{\partial}{\partial x_0}$ なる lexicographic order を用いている。

1. !-VARS へ、変数 $\frac{\partial}{\partial x_0}, \frac{\partial}{\partial x_1}, \cdots$ の個数を入れる。

例. $\frac{\partial}{\partial x_0}, \frac{\partial}{\partial x_1}$ のみを使用するのであれば、!-VARS := 2。

2. 対応. (大文字、小文字は区別しない)

$x_0 \leftrightarrow$ XØ ← ゼロ.　　$\frac{\partial}{\partial x_0} \longleftrightarrow$ DØ

$x_1 \leftrightarrow$ X1　　$\frac{\partial}{\partial x_1} \longleftrightarrow$ D1

$\vdots$　　$\vdots$

3. 微分作用素は、微分をうしろへもっていった正準形

$$\sum_\alpha a_\alpha(x)\left(\frac{\partial}{\partial x}\right)^\alpha \qquad (\alpha \text{ は multi index.})$$

を2の対応で、多項式に直して入力する。

例. $(x_0-x_1)\frac{\partial^2}{\partial x_0 \partial x_1} - a\frac{\partial}{\partial x_0} + b\frac{\partial}{\partial x_1} + c$ であれば.

(XØ−X1)*DØ*D1 −a*DØ +b*D1 +C

と入力。　出力もこの形式である。 (A は +, − については多項式環と同じ構造をしているため、多項式用 method (関数) を多く利用している。)　opmult(L,M) は作用素の積 LM を正準形でかえす。

4. プログラムのロードは、

in "dhØ.rr"$ in "dbØ.rr"$

イデアルの generator を ag(1), ag(2), … へいれる。ag(k) が最後のときおわりのマークとして、ag(k+1):=Ø ← ゼロ と Ø を代入しておく。

5. O.D.E. section を求めるだけなら、

!-ODE :=1;

としておく。(答えは FFFF に入る。)

6. gbasis()$ で計算開始。　答えも ag に入る。disp() は表示用。

7. 次は、Appell F_1 の section を求める計算例である。

(dhØ.rr → dh.r , dbØ.rr → db.r となっている。)

入力: generator.

$$\begin{cases} x_0\frac{\partial}{\partial x_0}\left(x_0\frac{\partial}{\partial x_0} + x_1\frac{\partial}{\partial x_1} + c - 1\right) - x_0\left(x_0\frac{\partial}{\partial x_0} + x_1\frac{\partial}{\partial x_1} + a\right)\left(x_0\frac{\partial}{\partial x_0} + b\right), \\ x_1\frac{\partial}{\partial x_1}\left(x_0\frac{\partial}{\partial x_0} + x_1\frac{\partial}{\partial x_1} + c - 1\right) - x_1\left(x_0\frac{\partial}{\partial x_0} + x_1\frac{\partial}{\partial x_1} + a\right)\left(x_1\frac{\partial}{\partial x_1} + b'\right) \end{cases}$$

← プログラム中では bp と書いている。

出力:

$$x_0(x_0-1)(x_1-x_0)\frac{\partial^3}{\partial x_0^3} + \left(a x_1 x_0 - a x_0^2 + b x_1 x_0 - 2b x_0^2 + b x_0 - c x_1 + c x_0 - x_1 x_0 b' + 3x_1 x_0 + x_1 b' - x_1 - 4x_0^2 + 2x_0\right)\frac{\partial^2}{\partial x_0^2} + (b+1)(a x_1 - 2a x_0 - b x_0 + c - x_1 b' + x_1 - 2x_0)\frac{\partial}{\partial x_0} - b(b+1)a$$

```
% reduce                                  ← reduce の起動.
REDUCE 3.3, 15-Jan-88 ...

1: on comp;                               ← 以後、読みこんだ procedure はコンパイルする。

2: in "dh.r"$ in "db.r"$                  ← Q(x,y)[∂/∂x,∂/∂y] 用、G-basis package (~~total order用~~ lex.) の読みこみ。(ロード)

-VARS := 2

      ...6514, length 2268 bytes

GBASIS                                    プログラムの読み込み終る.

4: f1:=opmult(x0*d0,x0*d0+x1*d1+c-1)-x0*opmult(x0*d0+x1*d1+a,x0*d0+b);

F1 :=  - X0*(A*B + A*X0*D0 + B*X1*D1 + B*X0*D0 - C*D0 + X1*X0*D0*D1 -

                         2    2          2
          X1*D0*D1 + X0 *D0  - X0*D0  + X0*D0)

5: ff1:=opmult(x1*d1,x0*d0+x1*d1+c-1)-x1*opmult(x0*d0+x1*d1+a,x1*d1+bp);

                                   2    2                           2
FF1 :=  - X1*(A*X1*D1 + A*BP - C*D1 + X1 *D1  + X1*X0*D0*D1 - X1*D1

          + X1*D1*BP + X1*D1 - X0*D0*D1 + X0*D0*BP)

6: ag(1):=f1$ ag(2):=ff1$ ag(3):=0;
                                          ← gbasis() 用の変数設定.

AG(3) := 0
```

F1の方程式を2つ入力.
opmultは、operatorとしての積を求めるprocedure.

```
10: !-ode:=1;
```

basisを計算するのではなく、ODE sectionが出てた時点で停止するためのフラグ。

```
-ODE := 1

Time: 68 ms

11: gbasis();
```

{ag(1),ag(2),ag(3)}の gbasisの計算

```
Size of old Ag is ..... 0

Compacted. Size of reduced Ag is ..... 2

***** ID fill no longer supported --- use lists instead

*** Z3 Z2 Z1 Z0 are non zero

Recuded Field factor: X1*X0

Spol of 1 and 2

***** ID fill no longer supported --- use lists instead

*** Z2 Z1 Z0 are non zero

Recuded Field factor:  - (A - C + 1)

                                                                2
1: New s-pol is added. A*B*X1 - A*B*X0 + A*X1*X0*D0 - A*X0 *D0 + B*

  2                            2                                    2
X1 *D1 + B*X1*X0*D0 - B*X1*D1 - B*X0 *D0 - C*X1*D0 + C*X0*D0 + X1*X0

   2          2                                          3    2      2
*D0  - X1*X0*D0  - X1*X0*D0*BP + X1*X0*D0 + X1*D0*BP - X0 *D0  + X0 *

  2      2
D0  - X0 *D0

Size of old Ag is ..... 2

***** ID fill no longer supported --- use lists instead

*** Z3 Z2 Z1 Z0 are non zero

Recuded Field factor: X0*(X0 - 1)

Found target! Answer is FFFF.
```

システムよりのメッセージ

答えはFFFFに。

```
Completion is completed.

   Basis consists of 3 elements.(Ag(1),....)

Time: 242386 ms
```

時間。約242秒＝約4分。

```
12: disp(ffff)$
```

ffffの表示。

```
  3
D0

   [X0]

   [X0 - 1]

   [X1 - X0]
```

結果。

以下略。

--------------------------------------//7. 計算例.


```
/usr/users/z007/basis/Rims/dh0.rr                    1989/1/14 14:8:25   1 p. 1--

%Order definition for the ring of differential Operators.
%       dh0.rr  1988/02/26
%               1989/01/14
%       Ring of differential operators with meromorphic coefficients.
%       Order is lexico graphic.  ( ....D3>D2>D1>D0 )
%
%       This program is implemented on REDUCE 3.2 and 3.3.
%       We do not use ``symbolic procedures'' to make a simple program.
%       But the time performance is not good for the reason.
%%%%%%%%%%%%%%%%%%%%%%%%% Modify here %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
!-VARS:=2;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%% global vars %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% X0,X1,.... D0,D1,D2,.....  are global variables.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
off factor;

symbolic operator numberp;
symbolic operator xx;
symbolic operator dd;
symbolic operator myzz;

symbolic procedure xx k;
intern compress(list('x,car explode(k)))$

symbolic procedure dd k;
intern compress(list('d,car explode(k)))$

symbolic procedure myzz k;
intern compress(list('z,car explode(k)))$

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

procedure head(f);
begin
        scalar m;
        for m:=(!-VARS-1) step (-1) until 0 do f:=mylterm(f,dd(m));
        return f;
end$

% headcof(f) is the head coefficients of f.
procedure headcof(f);             %for monic.  Calculate head coefficients
begin
        scalar m;
        for m:=(!-VARS-1) step (-1) until 0 do f:=mylcof(f,dd(m));
        return f;
end$

procedure mylcof(f,v);
begin
        scalar tmpDen;
        tmpDen:=den(f); f:=num(f);
        if (deg(f,v) neq 0) then return( lcof(f,v)/tmpDen)
        else return (f/tmpDen);
end$
procedure mylterm(f,v);
```

```
begin
        scalar tmpDen;
        tmpDen:=den(f); f:=num(f);
        if deg(f,v)=0 then return (f/tmpDen)
        else return( lterm(f,v)/tmpDen );
end$
procedure mypart(f);
begin
        for i:=(!-VARS-1) step (-1) until 0  do f:=mylterm(f,dd(i));
        return f;
end$

%  f and g must be monomials.
%  If f>g then 1 else 0.
procedure larger(f,g);
begin
        scalar fd,gd;
        if f neq 0 then fd:=f/headcof(f)
        else fd:=0;
        if g neq 0 then gd:=g/headcof(g)
        else gf:=0;
        return( largers0(fd,gd,!-VARS-1) );
end$
procedure largers0(f,g,m);
begin
        if m < 0 then return 0;
        if deg(f,dd(m))>deg(g,dd(m)) then return 1;
        if deg(f,dd(m))=deg(g,dd(m)) then return (largers0(f,g,m-1));
        return 0;
end$

%  If f is ODE of x0 then 1 else 0.
procedure isODE(f);
begin
        if f = 0 then return 0;
        if larger(dl,head(f))=1 then return 1 else return 0;
end$

%   f and g must be monomials.
%   If f is m-reducible by g then 1 else 0.
procedure reducible(f,g);
begin
        if infield(den(f/g))=1 then return 1 else return 0;
end$

%   If f does not contain D0,D1,... then 1 else 0.
procedure infield(f);
begin
        scalar result,m;
        if numberp(f) then return 1;
        result:=1; m:=!-VARS-1;
        while (m>=0) and (result=1) do <<
                if deg(f,dd(m)) neq 0 then result:=0;
                m:=m-1;
        >>;
        return result;
```



```
d$

    binomial coefficient.
rocedure binom(a,m);
egin
        scalar result,k;
        if (a<=0) or (m<=0) or (a=m) then return 1;
        if 2*m>a then m:=a-m;
        result:=1;
        for k:=1:m do <<
                result:=result*a/k;
                a:=a-1;
        >>;
        return result;
nd$

  Function for pretty printing
%%%%%%%%  disp() for REDUCE 3.3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
rocedure disp(f);
egin
        scalar tmp;
        if (f=0) then << write "0"; return(0); >>;
        while f neq 0 do <<
                tmp:=head(f); f:=f-tmp;
                disp!-monomial(tmp);
        >>;
nd$

rocedure disp!-monomial(f);
egin
        scalar tmp,tmp0,m,k;
        tmp:=1;
        for m:=0:(!-VARS-1) do <<
                k:=deg(f,dd(m));
                f:=f/(dd(m)**k);
                tmp:=(dd(m)**k)*tmp;
        >>;
        write tmp;
        tmp:=factorize(f);
        for each tmp0 in tmp do <<
                write "   [",tmp0,"]"$
        >>;
        write "------------------------------------"$
nd $

%%%%%%%%%%%  disp() for REDUCE 3.2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
rocedure olddisp(f);
egin
        scalar tmp;
        write "/-------------------------------------------------------------";
        while f neq 0 do <<
                tmp:=mypart(f);
                on factor; write "   ",tmp; off factor;
                f:=f-tmp;
                tmp:=0; %to save memory.
        >>;
```

```
        write "-------------------------------------------------------------/
end$

% opmult(f,g) is the multiplication of f and g as operators.
procedure opmult(f,g);  % don't write tail recursive program to save memory.
begin
        scalar tmp,result;
        if (f=0) or (g=0) then return 0;
        result:=0;
        while ( f neq 0 ) do <<
                tmp:=mypart(f);
                result:=opmults1(tmp,g)+result;
                f:=f-tmp;
        >>;
        return result;
end$
%  f must be monomial.
procedure opmults1(f,g);
begin
        scalar a,fd,k,m,alpha,tmp;
        if numberp(g) or numberp(f) then return (f*g);
        a:=headcof(f);
        fd:=f/a;
        if numberp(fd) then return (f*g);
        for k:=0:(!-VARS-1) do <<
                tmp:=0;
                alpha:=deg(fd,dd(k));
                for m:=0:alpha do <<
                        tmp:=tmp+binom(alpha,m)*df(g,xx(k),m)*dd(k)**(alpha-m
                >>;
                g:=tmp;
        >>;
        return (a*g);
end$
%   opPower(f,p) is f*f*....*f ( p times ).
procedure opPower(f,p);
begin
        scalar tmp;
        tmp := 1;
        if p = 0 then return 1;
        if p = 1 then return f;
        for i:=1:p do <<
                tmp :=opmult(f,tmp);
        >>;
        return(tmp);
end$

on gcd;
write " !!!!! ON GCD. !!!!! ";
end$
```



```
%  The generic  procedures to compute the Groebner basis
%  for Differential Operators, difference operators and Weyl algebras.
%  An order must be defined in dh0.rr
%       db0.rr  1988/12/05
%               1989/01/14
%%%%%%%%%%%%%%%% Global variables %%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%   modify here %%%%%%%%%%%%%%%%%%%%%%%%%%%
SIZEAG:=40; SIZEAS:=80;
!-ODE  := 0;    % If you want to stop when you find ODE in the
                % Ag, then !-ODE:=1 else !-ODE:=0.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
array Ag(SIZEAG);       % G-basis

clear FFFF;     % If you want to find ODE in the generators, the result
                % is put to FFFF.   cf. !-ODE
%%%%%%%%%%%%%%%%% variables for the system. %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
!-DBG  := 0$    % Debug flag of mred().
!-DBG0 := 0$    % Debug flag of reduceAg()
!-DBG1 := 0$    % Debug flag of gbasis(). Output the candidate of S-pol.
!-RED1FLAG := 0$% used in mred1().   1==> reduced.
!-REDFLAG :=0$  % used in mred().
!-FFFFFLAG:=0$  % Flag for the ODE check
%%%%%%%%%%%%%%%%%%% end of definition of global vars %%%%%%%%%%%%%%%%%%%%%
symbolic operator myzz;

%%%%%%%%%%%%%%%%   modify here %%%%%%%%%%%%%%%%%%%%%%%%%%%%
procedure checkAgForStop();
begin
        scalar result;
        result:=0;
        return( result );
end$

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% mred1(f,g) is m-reduction of f by g.
procedure mred1(f,g);
begin
  scalar hf,hg,result;
  !-RED1FLAG := 0;
  if (g=0) then write "@@@ Error in mred1() : devide by 0";
  if numberp(g)=1 then return 0;
  result:=f;
  repeat <<
    f:=result;
    if (f=0) then result:=0
    else <<
        hf:=head(f);
        hg:=head(g);
        if (reducible(hf,hg) = 0) then
                result := f
        else    <<
                !-RED1FLAG:=1;
                result:=f-opmult(den(hf/hg)*hf/hg,g)/den(hf/hg);
                result:=result*den(result); % (x+y)/2;
        >>;
```

```
     >>;
  >> until ( f = result );
  return result;
end$

% mredAg(f) is m-reduction of f by the set Ag.
procedure mredAg(f);
begin
        scalar oldf,ii,tmp,k,ftmp,mm,kk;
        !-REDFLAG:=0;
        k:=0; oldf := 0;
        while (oldf neq f) and (f neq 0) do <<
           ii:=1; oldf:=f;
           while (Ag(ii) neq 0) and (f neq 0) do <<
                tmp := mred1(f,Ag(ii));
                if !-RED1FLAG=1 then !-REDFLAG:=1;
                if ( !-DBG = 1 ) then   <<
                        write "###mredAg():",f," ===> ",tmp;
                        write "               by Ag(",ii,")  :  ",Ag(ii);
                >>;
                if (tmp neq f) then <<
                        k:=k+1;
                        f:=tmp; tmp:=0; %to save mem.
                >>;
                ii:=ii+1;
           >>;
        >>;
        return(f );
end$

% mredaa(f) is m-reduction of f by the set aa.
procedure mredaa(f);
begin
        scalar oldf,ii,tmp,k,ftmp,mm,kk;
        !-REDFLAG:=0;
        k:=0; oldf := 0;
        while (oldf neq f) and (f neq 0) do <<
           ii:=1; oldf:=f;
           while (aa(ii) neq 0) and (f neq 0) do <<
                tmp := mred1(f,aa(ii));
                if !-RED1FLAG=1 then !-REDFLAG:=1;
                if ( !-DBG = 1 ) then   <<
                        write "###mredaa();",f," ===> ",tmp;
                        write "               by aa(",ii,")  :  ",aa(ii);
                >>;
                if (tmp neq f) then <<
                        k:=k+1;
                        f:=tmp; tmp:=0; %to save mem.
                >>;
                ii:=ii+1;
           >>;
        >>;
        return(f );
end$

% sp(f,g) is the s-polynomial of f and g.
```



```
procedure sp(f,g);
begin
        scalar hf,hg,lc,tmp,hcoff,hcofg,coflc,ftmp,kk,mm;
        if (f=0) then return g;
        if (g=0) then return f;
        hf:=head(f); hg:=head(g);
        hcoff:=headcof(hf); hcofg:=headcof(hg);
        hf:=hf/hcoff; hg:=hg/hcofg;
        lc:=hf*hg/gcd(hf,hg);
        coflc:=hcoff*hcofg/gcd(hcoff,hcofg);
        tmp:=opmult((coflc/hcoff)*lc/hf,f)-opmult((coflc/hcofg)*lc/hg,g);
        tmp:=tmp*den(tmp);
        return( tmp );
end$

procedure getsizeAg();
begin
        scalar n;
        n:=1;
        while Ag(n) neq 0 do n:=n+1;
        return(n-1);
end$

procedure getsizeAs();
begin
        scalar n;
        n:=1;
        while As(n) neq 0 do n:=n+1;
        return(n-1);
end$

%  The procedure reduces the set Ag.
procedure reduceAg;
begin
    scalar n,j,k,tmp,m;
    n:=getsizeAg();
    array aa(n),newAg(n+1);n:=getsizeAg();
    for k:=1:n do newAg(k):=Ag(k);
    repeat <<
        %  newAg ===(compaction)===> Ag
        n:=getsizeAg();  newAg(n+1):=0;
        k:=1;
        for j:=1:(n+1) do <<
                Ag(k):=newAg(j);
                if (Ag(k) neq 0) then k:=k+1;
        >>; % compaction
        Ag(k):=0; n:=getsizeAg();
        %  Ag ==> newAg
        for j:=1:(n+1) do newAg(j):=Ag(j);
        if !-DBG0 neq 0 then write "#reduceAg() : size of Ag is ",n;
        %  Do reduction of newAg
        K:=0;
        while (K < N) do <<
                K:=K+1;
                % aa<== complement of newAg(k) except 0.
                m:=1;
```

```
                for j:=1:n do <<
                        if (j neq k) and (newAg(j) neq 0) then <<
                                aa(m):=newAg(j); m:=m+1;
                        >>;
                >>;
                aa(m):=0;
                %
                newAg(k):=0; %*% to save memory.
                tmp:=mredaa(Ag(k));
                if !-ODE=1 then <<
                        if isODE(tmp)=1 then <<
                                write "Found target! Answer is FFFF. ";
                                ffff:=tmp; !-FFFFFLAG:=1;K:=N;
                        >>;
                >>;
                if !-DBG0 neq 0 then
                        write "#reduceAg(): M-reduction of Ag(",k,") : "
                                ,Ag(k)," --> ",tmp;
                newAg(k):=tmp;
        >>;
    >> until (seteqAgAndNewAg()=1 OR !-FFFFFLAG=1);
    clear aa,newAg;
end$

% If the set Ag and newAg is equal, then 1 else 0.
procedure seteqAgAndNewAg();
begin
        scalar result,k;
        k:=1;
        while (k>0) do <<
                if Ag(k) neq newAg(k) then <<result:=0; k:=-3;>>
                else <<
                        if (Ag(k)=0) and (newAg(k)=0) then <<
                                result:=1; k:=-3;
                        >>;
                >>;
                k:=k+1;
        >>;
        return result;
end$

%  The procedure computes a Grobner basis.
% input:  Ag
% output: Ag  or FFFF.
procedure gbasis;
begin
        scalar k,j,n,m,tmp,kkk;
        array As(SIZEAS);       % work area of G-basis
        !-FFFFFLAG:=0;
        n:=getsizeAg();
        for k:=1:n do As(k):=Ag(k);
        As(n+1):=0; Ag(1):=0;
        repeat <<
           % Ag:=Ag+As
           j:=getsizeAs();
           n:=getsizeAg();
```



```
           write "gbasis():(reduction of Ag): Size of old Ag is ... ",n;
           m:=1;
           for k:=(n+1):(n+j+1) do <<Ag(k):=As(m); m:=m+1; >>;
           % reduce Ag
           reduceAg();

           !-FFFFFLAG := checkAgForStop();

           if (!-FFFFFLAG = 0) then <<
                n:=getsizeAg();
                write "gbasis():     Compacted. Size of reduced Ag is ... ",n;
                % generate new S-pols
                kkk:=0;As(1):=0;
                K:=0;
                while (K<N) do <<
                K:=K+1;
                        J:=K;
                        while (J<N) do <<
                                J:=J+1;
                                tmp:=sp(Ag(k),Ag(j));
                                write "gbasis(): S-pol of ",k," and ",j;
                                if !-DBG1 neq 0 then
                                   write "##gbasis(): The candidate  is ",tmp;
                                tmp:=mredAg(tmp);
                                if tmp neq 0 then <<
                                        kkk:=kkk+1;
                                        As(kkk):=tmp; As(kkk+1):=0;
                                        write "gbasis():",kkk
                                           ,": New s-pol is added. ",tmp;
                                        if !-ODE=1 then <<
                                         if isODE(tmp)=1 then <<
                                           write
                                            "gbasis(): Found target!",
                                                 " Answer is FFFF. ";
                                           ffff:=tmp; !-FFFFFLAG:=1;
                                           K:=N;J:=N;
                                         >>;
                                        >>;
                                >> else <<
                                        write
                                          "gbasis(): The S-pol is reduced to 0.
";
                                >>;
                        >>;
                >>;
           >>;
      >> until (getsizeAs() = 0 OR !-FFFFFLAG=1);
      if !-FFFFFLAG neq 1 then <<
             write "gbasis(): Completion is completed.";
             write
              "   Basis consists of ",getsizeAg()," elements.(Ag(1),...)";
      >>;
      clear As;
end$

end$
```

参考文献

[Erd1] A. Erdélyi et al., Higher Transcendental Functions. MacGraw-Hill, New York, 1953.

[Erd2] A. Erdélyi, Transformations of Hypergeometric Functions of Two Variables. Proc. Roy. Soc. Edinburgh Sect A. 62 (1948) 378–385.

[Fur] (フュルカワ) A. Furukawa, Gröbner-Bases について, 数理研考究録 612, 1–23.

[Hea] A.C. Hearn, REDUCE USER'S MANUAL, Santa Monica, The Rand Corporation, 1988.

[Oku] K. Okubo, On The Group of Fuchsian Equations, 昭和55年度科研, 研究成果報告書, 1981.

[Suz-Ono] T. Suzuki, O. Onodera, REDUCEでReduceを, 数理研考究録 663 (1988), 23–39.

[Tak1] N. Takayama, Gröbner Basis and the Problem of Contiguous Relations, Japan J. Appl. Math., 6 (1989), 147–160.

[Tak2] N. Takayama, 微分差分作用素用 Gröbner 基底パッケージを用いた超越関数式の零決定問題の解法, 理研シンポ予稿, 1989.

[Zei] D. Zeilberger, A holonomic systems approach to special functions identities, preprint.

[Nou] M. Noumi, Wronskian determinants and the Gröbner representation of a linear differential equation, preprint, Sophia University.

[Gal] A. Galligo, Some algorithmic questions on ideals of differential operators. Lect. Note in Computer Sci, 204 (1985), 413–421.

非線形常微分方程式の特異点

東大・理　村田嘉弘　(Yoshihiro Murata)

§0. はじめに

まず、次の方程式を考えてみよう。

(E1)
$$y'' = -\frac{(y')^2}{y} + \frac{y'}{x} - \frac{y}{2x^2}$$

この方程式は、実は求積できて、一般解は

$$y = \sqrt{x(A + B\log x)} \qquad (A, B) \neq (0, 0)$$

である。よって、(E1) の解の特異点は次のように分類される。

位置		種類
$x=0$	解に依らない (動かない)	超越的
$x=e^{-\frac{A}{B}}$	解に依る (動く)	代数的

この例でもわかるように、非線形の場合は、線形のときと異なり、方程式の形からは予測できない位置に解の特異点が現れ、しかもそれがあちこち動きまわるという現象が起きる

RIMS *Kokyuroku* 681

Algebraic Manipulation for Differential Equations

February, 1989

Research Institute for Mathematical Sciences

Kyoto University, Kyoto, Japan