数理解析研究所講究録 1395 巻 2004 年 224-230



# 多項式行列の行列式の補間による計算

木村欣司
KINJI KIMURA
神戸大学自然科学研究科
GRADUATE SCHOOL OF SCIENCE AND TECHNOLOGY, KOBE UNIVERSITY *

## 1 はじめに

行列式は, 多くの分野たとえば数学, 物理, 化学, 工学などで重要な存在である. そして, 可積分系においてはその役割の重要性はいまさら述べるまでもない. この講究録では, 多変数多項式を要素とする行列式の補間を用いる算法を紹介する. 我々の Singular との比較実験では, 多変数多項式の変数の数が少ない場合 (1-4 variables) には Singular に実装されている fraction free Gaussian elimination よりも我々の方法のほうが短い時間で計算できた. ここで断っておくが, Singular に実装されている fraction free Gaussian elimination は通常のものではない. 詳しくは, Singular の source を見ていただきたい.

## 2 アルゴリズムを述べるための準備

### 2.1 多変数多項式の補間法

#### 2.1.1 中間メモリを必要としない Newton 補間アルゴリズム

補間のために, 中間メモリを必要としない Newton 補間アルゴリズムを用意する.
入力:$n+1$ 次元のベクトル $y_i$ と $x_i$
ここで, $y_i$ は $x_i$ における $n+1$ 個の sampling data である.
数式処理では, $y_i$ は数に限らず多変数多項式が入力されることもある.
出力:$n+1$ 次元のベクトル $y_i$
もし, $n+1$ 個の sampling data があれば

$$f(x) = a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1) + ... + a_n(x-x_0)(x-x_1)\cdots(x-x_n) \tag{1}$$

の係数は一意に決まる. その係数は, 最終的に

$$y_i \leftarrow a_i \tag{2}$$

の対応で再び $y_i$ に格納される.
for $i = 1, ..., n$
  for $j = n, ..., i$
    $y_j \leftarrow (y_j - y_{j-1})/(x_j - x_{j-i})$

*kimura@math.koba-u.ac.jp

### 2.1.2 Newton 補間の途中のデータ形式

いま, 係数を整数とする 1 変数多項式

$$f(x) = b_0 + b_1 x + b_2 x^2 + ... + b_n x^n \tag{3}$$

から, data を sampling し Newton 補間

$$f(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + ... + a_n(x - x_0)(x - x_1)\cdots(x - x_n) \tag{4}$$

することを考える.

そのとき, $x_i$ が整数ならば $y_i = f(x_i)$ は当然整数である.
このような状況の下で, アルゴリズム (2.1.1) は for 文の最終結果として $a_i$ を得るわけであるが for 文の途中にあらわれる演算

$$y_j \leftarrow \frac{y_j - y_{j-1}}{x_j - x_{j-i}} \tag{5}$$

にはどのような性質があるであろうか?

(5) の右辺が差分商を計算していることは有名ある. もっとも一般の差分商で考える,

$$f[x_0, x_1, x_2, ..., x_k] = \frac{\begin{vmatrix} 1 & 1 & 1 & ... & 1 \\ x_0 & x_1 & x_2 & ... & x_k \\ x_0^2 & x_1^2 & x_2^2 & ... & x_k^2 \\ ... & ... & ... & ... & ... \\ x_0^{k-1} & x_1^{k-1} & x_2^{k-1} & ... & x_k^{k-1} \\ f(x_0) & f(x_1) & f(x_2) & ... & f(x_k) \end{vmatrix}}{\begin{vmatrix} 1 & 1 & 1 & ... & 1 \\ x_0 & x_1 & x_2 & ... & x_k \\ x_0^2 & x_1^2 & x_2^2 & ... & x_k^2 \\ ... & ... & ... & ... & ... \\ x_0^{k-1} & x_1^{k-1} & x_2^{k-1} & ... & x_k^{k-1} \\ x_0^k & x_1^k & x_2^k & ... & x_k^k \end{vmatrix}}, \tag{6}$$

で表わされる. 証明は, Jacobi の恒等式を使う. $a_k$ の explicit な表示は,

$$a_k = f[x_0, x_1, x_2, ..., x_k]. \tag{7}$$

ここで,

$$x_u^k - x_v^k = (x_u - x_v)(...), \tag{8}$$

$$f(x_u) - f(x_v) = (x_u - x_v)(...), \tag{9}$$

に注目する. 列基本変形を用いると式 (6) の分子は分母によって割りきれることが保証できる. ゆえに, $b_i, x_i$ が整数ならば途中の計算

$$y_j \leftarrow \frac{y_j - y_{j-1}}{x_j - x_{j-i}} \tag{10}$$

は必ず割り切れ for 文の途中の $y_j$ はすべて整数になる.

同様の議論が $b_i$ が整数を係数としてもつ多変数多項式の場合にも成立し, for 文途中の $y_j$ は整数を係数とする多変数多項式になる.

### 2.1.3 Newton 補間の性質

1 変数 2 次の多項式

$$f(x) = 2 + 3x + 5x^2$$

は, 3 つの sampling data

$$f(0) = 2, \quad f(1) = 10, \quad f(2) = 28$$

より以下の手順で復元できる. (*) は演算の順番をあらわす.

$$\begin{array}{lll} g_0 = f(0) & & \\ & (2)g_1 = f(1) \leftarrow (f(1) - f(0))/1 & \\ f(1) & & (3)g_2 = f(2) \leftarrow (f(2) - f(1))/2 \\ & (1)f(2) \leftarrow (f(2) - f(1))/1 & \\ f(2) & & \end{array}$$

$g_0 = 2, g_1 = 8, g_2 = 5$ より

$$f(x) = g_0 + g_1 x + g_2 x(x-1) = 2 + 3x + 5x^2.$$

$g_j$ は $f(0)$ から $f(j)$ までの 線形結合 によって計算される. 仮に

$$f(0) = 2, \quad f(1) = 10,$$

しかサンプリングしていなくとも, 計算の手順より

$$g_0 = 2, \quad g_1 = 8,$$

まで正確に計算できる. Newton 補間は data と結果の依存関係に特徴がある.

## 3 アルゴリズムの概略

### 3.1 補間のための行列式における上限の計算

次の行列式を例に議論を進める,

$$A = \begin{vmatrix} x+y & 2 \\ 3 & xy \end{vmatrix} \tag{11}$$

#### 3.1.1 1 変数について行列式の最大次数

補間のため, 行列をながめて 1 変数それぞれについて最大次数を見積もる.
変数 $x$ に注目し, 各行と列をながめて変数 $x$ について行列式における最大次数を計算する,

$$A = \begin{array}{|cc|l} x+y & 2 & \to 1 \\ 3 & xy & \to 1 \\ \downarrow & \downarrow & \\ 1 & 1 & \end{array} . \tag{12}$$

第1行の$x$の最大次数は1. 第2行の$x$の最大次数は1. 第1行と第2行の最大次数の合計は2. 第1列の$x$の最大次数は1. 第2列の$x$の最大次数は1. 第1列と第2列の最大次数の合計は2. ゆえに, 行列式における変数$x$の最大次数の上限は2である. 同様のことを$y$についてもおこなうと, 行列式における変数$y$の最大次数の上限も2である. 行列式のサイズが小さい場合, より厳密な見積もりを行うがここでは省略する.

### 3.1.2 total degreeについて行列式における上限

計算量を低減させるためには, total degreeの上限を計算することが重要となる.
各行と列をながめてtotal degreeについて行列式における上限を計算する,

$$A = \left| \begin{array}{cc|c} x+y & 2 & \to 1 \\ 3 & xy & \to 2 \\ \hline \downarrow & \downarrow & \\ 1 & 2 & \end{array} \right| . \tag{13}$$

第1行の$x$のtotal degreeは1. 第2行の$x$のtotal degreeは2. 第1行と第2行のtotal degreeの合計は3. 第1列の$x$のtotal degreeは1. 第2列の$x$のtotal degreeは2. 第1列と第2列のtotal degreeの合計は3. ゆえに, total degreeについて行列式における上限は3である.

### 3.1.3 1変数の最大次数とtotal degreeより行列式を仮定する

上の結果から, 我々は行列式を以下ように仮定できる,

$$A = c_0 + c_1 x + c_2 y + c_3 xy + c_4 x^2 + c_5 y^2 + c_6 x^2 y + c_7 x y^2 + c_8 x^2 y^2, \tag{14}$$

ただし, $c_8 = 0$.
$c_8 = 0$と仮定できるのは, total degreeの上限を計算したからである.
補間多項式の一意性より8個のsampling dataから(14)は原理的に一意に決まる. しかし, 連立一次方程式をGaussian eliminationで解いて係数を求めるのは効率的でない.
Newton補間をもちいる,

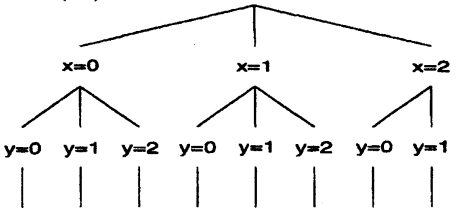
$$\begin{aligned} A \;=\; & (d_0 + (y-y_0)d_1 + (y-y_0)(y-y_1)d_2) + (x-x_0)(d_3 + (y-y_0)d_4 + (y-y_0)(y-y_1)d_5) \\ & + (x-x_0)(x-x_1)(d_6 + (y-y_0)d_7 + (y-y_0)(y-y_1)d_8) \end{aligned} \tag{15}$$

ただし, $d_8 = 0$. 式(15)の形式を仮定して計算する. $c_8 = 0$が$d_8 = 0$に対応することは容易にわかるであろう. ここで, sampling pointは$x_0 = 0, x_1 = 1, x_2 = 2, y_0 = 0, y_1 = 1, y_2 = 2$とする.

## 3.2 dataのsamplingと多変数補間の効率的実装法

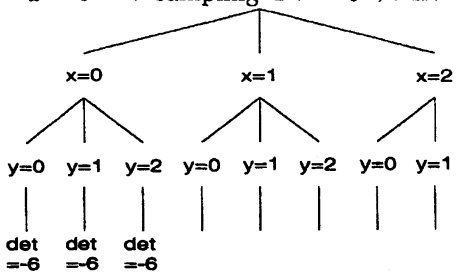
### 3.2.1 多変数多項式の補間の方法

式(15)の仮定から次の木を作る.

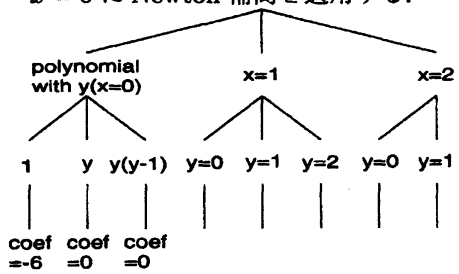


正確には, 各階層の tree の枝は配列内にポインタとして格納されている.

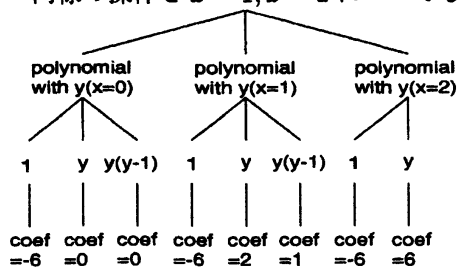
$x=0$ のみ sampling をおこなう. Hadamard bound を用いた modular 技法により det を計算する.




$x=0$ に Newton 補間を適用する.




同様の操作を $x=1, x=2$ についてもおこなう.




$x=2$ における多項式 $-6+6y$ は正しくない. $A|_{x=2}=-6+6y+2y(y-1)$ である. しかし, この正しない多項式からでも正しい $A$ を得ることができる.

$d_8=0$ であることはすでに確定している. さらに, "Newton 補間の性質"から $x=2$ の項 $y(y-1)$ の係数が影響をあたえるのは $d_8$ のみである. よって, 項 $y(y-1)$ の係数は最終結果に影響を与えないため $x=2$ の項 $y(y-1)$ の係数が関係する演算はすべて省略すればよい. それを具体例で見る.

$x$ について Newton 補間をおこなう. ここで, sampling data は多項式である.

$$\begin{aligned} f(0) &= A|_{x=0}=-6 \\ f(1) &= A|_{x=1}=-6+2y+y(y-1) \\ f(2) &= \text{truncate}(A|_{x=2})=-6+6y \end{aligned}$$

から, $A$ を計算する. $(*)$ は演算の順番をあらわす.

$$\begin{array}{ccccc} h_0=f(0) & & & & \\ & & (2)h_1=f(1)\leftarrow(f(1)-f(0))/1 & & \\ f(1) & & & & (3)h_2=f(2)\leftarrow(f(2)-f(1))/2 \\ & & (1)f(2)\leftarrow(f(2)-f(1))/1 & & \\ f(2) & & & & \end{array}$$

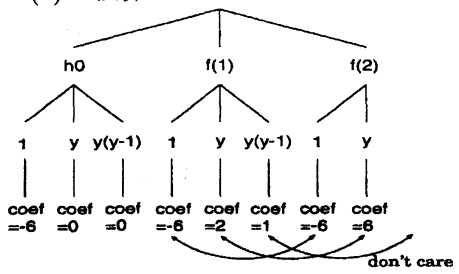
演算の手順

$(1)f(2)=4y\leftarrow \text{truncate}[(f(2)-f(1))/1]=((-6+6y)-(-6+2y))/1$

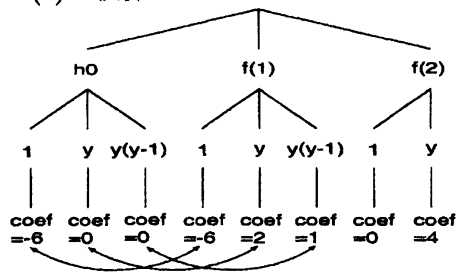
$(2)h_1 = f(1) = 2y + y(y-1) \leftarrow (f(1) - f(0))/1 = ((-6 + 2y + y(y-1)) - (-6))/1$

$(3)h_2 = f(2) = y \leftarrow \text{truncate}[(f(2) - f(1))/2] = ((4y) - (2y))/2$

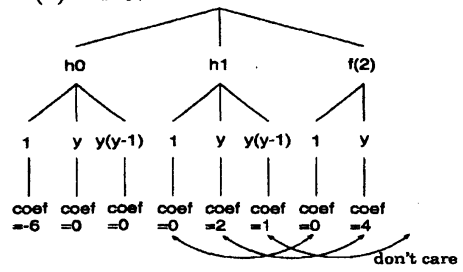
(1) の演算



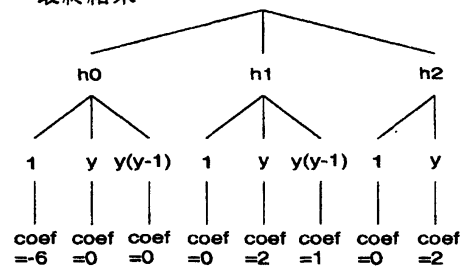

(2) の演算




(3) の演算




最終結果




truncate を適切におこなえば, $\text{truncate}(A|_{x=2})$ をつかっても

$$A = (-6) + x(2y + y(y-1)) + x(x-1)(y + \underline{0} \times y(y-1)),$$

正しい結果が得られる.

## 4 有効性

このアルゴリズムならびに実装は, 可積分系の研究者からの要請をうけておこなったものである. 後日, その目的には別のアルゴリズムが有効であることがわかりこのアルゴリズムならびに実装を用いる必要はな

かった.

しかし, なにも成果を得ることがなかったというわけではない. 最近我々は量子コンピュータの問題に取り組んでいる. そこでは, 大規模な連立代数方程式求解が必要となる. そのために, Gelfand,Kapranov,Zelevinsky multipolynomial resultant を計算する必要がある. この補間による行列式の計算法は, multipolynomial resultant の計算に有効であった. 具体的内容は, 別の機会に報告することとする.

## 5 Appendix:Pfaffian 版 fraction free Gaussian elimination

以下, 神戸大学の太田泰広先生の研究を紹介する.

1. $a_{ij}$ $(1 \leq i < j \leq 2N)$ に対して、Pfaffian $\mathrm{Pf}(a_{ij})_{1\leq i<j\leq 2N}$ を計算したかったとする.

2. 初期値の入力：

$$s_{ij}^{(1)} := a_{ij} \qquad 1 \leq i \leq 2N-1,\ i+1 \leq j \leq 2N$$
$$s_{-1,0}^{(0)} := 1$$

3. $k = 1, 2, \cdots, N-1$ に対し順に

$$s_{ij}^{(k+1)} := \frac{s_{2k-1,2k}^{(k)} s_{ij}^{(k)} - s_{2k-1,i}^{(k)} s_{2k,j}^{(k)} + s_{2k-1,j}^{(k)} s_{2k,i}^{(k)}}{s_{2k-3,2k-2}^{(k-1)}} \qquad 2k+1 \leq i \leq 2N-1,\ i+1 \leq j \leq 2N$$

を実行する.

4. 結果の出力：$k = N-1$ のステップで計算された $s_{2N-1,2N}^{(N)}$ が Pfaffian $\mathrm{Pf}(a_{ij})_{1\leq i<j\leq 2N}$ を与える.

$$s_{2N-1,2N}^{(N)} = \mathrm{Pf}(a_{ij})_{1\leq i<j\leq 2N}$$

5. 分母が 0 になる場合の処理：第 $k$ ステップで $s_{2k-1,2k}^{(k)} = 0$ となってしまったら、

   (a) $s_{2k-1,j}^{(k)}$ $(2k+1 \leq j \leq 2N)$ の中から 0 でないものを探す.

   (b) もし $s_{2k-1,j}^{(k)}$ $(2k+1 \leq j \leq 2N)$ がすべて 0 なら、もともとの Pfaffian $\mathrm{Pf}(a_{ij})_{1\leq i<j\leq 2N}$ も 0.

$$\mathrm{Pf}(a_{ij})_{1\leq i<j\leq 2N} = 0$$

   (c) $s_{2k-1,j}^{(k)}$ $(2k+1 \leq j \leq 2N)$ の中に 0 でないものがあったとき：

$$s_{2k-1,j}^{(k)} \neq 0 \qquad (2k+1 \leq j \leq 2N)$$

   となる $j$ に対して、

$$\begin{aligned}
&s_{2k-1,2k}^{(k)new} := s_{2k-1,j}^{(k)old} \qquad s_{2k-1,j}^{(k)new} := -s_{2k-1,2k}^{(k)old} \\
&s_{2k,l}^{(k)new} := -s_{lj}^{(k)old} \qquad s_{lj}^{(k)new} := s_{2k,l}^{(k)old} \qquad 2k+1 \leq l \leq j-1 \\
&s_{2k,l}^{(k)new} := s_{jl}^{(k)old} \qquad s_{jl}^{(k)new} := -s_{2k,l}^{(k)old} \qquad j+1 \leq l \leq 2N
\end{aligned}$$

   とデータを入れ換えて、$s_{2k-1,2k}^{(k)} \neq 0$ にする.