数理解析研究所講究録
1514巻 2006年 101-107

多変数近似GCD用算法の性能比較

讃岐 勝
MASARU SANUKI
筑波大学 数理物質科学研究科
GRADUATE SCHOOL OF PURE AND APPLIED SCIENCES, UNIVERSITY OF TSUKUBA *

Abstract

本稿では、近似PC-PRS算法と特異値分解に基づく三つの算法を簡単に説明し、各算法の効率・精度について考察する。またそれぞれの算法の長所・短所について考察する。

1 はじめに

多変数多項式の近似GCD計算について、Zhi-Noda[ZN00]が三つの算法の実験を行った結果、次の知見を得ている。a) Ochi *et al.*[ONS91]による互除法に基づく多項式剰余列(PRSと略記)算法は、精度は安定しているが、項数が増えて時間がかかる。b) 一般Hensel構成に基づくEZ-GCD(Extended Zassenhaus-GCD)算法[MY73, Wan80]を近似化した方法は、計算は速いが精度が落ちることが多々ある[SY98]。c) Corless *et al.*[CGTW95]が提案したModular算法など他の方法も欠点がある。

ISSAC2004で、Gao *et al.*[GKMYZ04]とZeng-Dayton[ZD04]が独立に一般化されたSylvester行列の特異値分解に基づく近似GCD算法を提案した。一般化されたSylvester行列の構成法の違いを除けば、両算法はほぼ同じである。しかし両算法のいずれも、変数の個数あるいは次数の個数が増えるにつれてSylvester行列のサイズが急激に膨張する(§3.1 表4)。特異値分解の計算量は列または行の3乗に比例するため、両算法とも多変数・高次では計算効率が大きく落ちる。

多変数多項式のGCD計算では、従変数をべき級数として扱って剰余列を計算するSasaki-Suzuki [SS92]のPC-PRS(Power-series Coefficient PRS)算法がある。この算法は従変数について高次項を打ち切るのでPRS算法に比べて非常に効率がよい。この算法を近似GCD計算用に改変することで、PRS算法の持つ高精度性とPC-PRS算法の持つ高効率性の両面を確認した[San05]。しかし、この算法(近似PC-PRS算法と呼ぶ)には安定性の面で問題が残っており、現時点で解決には至っていない。

本稿では、近似PC-PRS算法と特異値分解に基づく算法を簡単に説明し、各算法の効率・精度について考察する。またそれぞれの算法の長所・短所について考察する。

本稿では次の記号を用いる。多項式 $F(x,u)=f_m x^m+\cdots+f_0\in\mathbb{C}[x,u]=\mathbb{C}[x,u_1,\cdots,u_\ell]$ に対して、$\deg(F)$ と $\mathrm{lc}(F)$ はそれぞれ F の主変数 x に関する次数と主係数を表す。単項式 $T=cu_1^{e_1}\cdots u_\ell^{e_\ell}, c\in\mathbb{C}$ に対して $\mathrm{tdeg}_u(T)=e_1+\cdots+e_\ell$ と定義するとき、$\mathrm{tdeg}_u(F)$ と $\mathrm{tdeg}(F)$ はそれぞれ従変数 $u_1,\cdots,u_\ell$ と変数 $x,u_1,\cdots,u_\ell$ に関する全次数を表す：$\mathrm{tdeg}_u(F)=\max\{\mathrm{tdeg}_u(f_m),\cdots,\mathrm{tdeg}_u(f_0)\}$。$||\cdot||_p$ は p-ノルムを表す(pを省略したときは無限大ノルムを表すことにする)。$\gcd(A,B;\varepsilon)$ は多項式 A と B の許容度 ε の近似GCDを表す。$\mathrm{pp}(F;\varepsilon)$ と $\mathrm{cont}(F;\varepsilon)$ はそれぞれ F の許容度 ε の原始的部分と許容度 ε の係因子を表す：$\mathrm{cont}(F;\varepsilon)=\gcd\big(f_m,\gcd(f_{m-1},\cdots,f_0;\varepsilon);\varepsilon\big)$、$\mathrm{pp}(F;\varepsilon)=F/\mathrm{cont}(F;\varepsilon)$。

*sanuki@math.tsukuba.ac.jp

2 多変数多項式近似GCD算法

2.1 近似PC-PRS算法

与えられた多変数多項式 $P_1, P_2 \in \mathbb{C}[x,u]$ の GCD を求める PRS 算法では、剰余列の最後の要素 P_k は $G = \gcd(P_1, P_2)$ の倍数となる：$P_k = C \cdot G$。ただし、$C \in \mathbb{C}[u]$ である。そこで、P_1 と P_2 を原始的にしておき、$C = \mathrm{cont}(P_k)$ で C を求め、G を $G = \mathrm{quotient}(P_k, C)$ と計算する。ここで、$\mathrm{quotient}(P_k, C)$ の計算では、除多項式と被除多項式の全ての項が必要なわけではない。例として、除多項式の次数が 100、被除多項式の次数が 90 の場合には商の次数は 10 となり、商を求めるには除多項式と被除多項式の高次項 (低次項) の 10 次分があれば十分である。

$P_1, P_2 \in K\{u\}[x]$ の係数をべき級数として扱い、u の全次数が e 以下の項を残し、全次数が $e+1$ 以上の項は捨てることにする。u に関する全次数を表すために全次数変数 t を導入し、$u_i \mapsto tu_i$ $(i = 1, \cdots, \ell)$ と変換し計算する。このようにして計算した PRS を $(\tilde{P}_1, \tilde{P}_2, \cdots, \tilde{P}_k)$ と表す。実際には $\tilde{P}_i$ は次のように計算する：

$$\begin{cases} \beta_i \tilde{P}_{i+1} \equiv (\alpha_i \tilde{P}_{i-1} - Q_i \tilde{P}_i)/\max\{||\alpha_i||, ||Q_i||\} \pmod{t^{e+1}}, \\ \alpha_i = \mathrm{lc}(\tilde{P}_i)^{d_i+1}, \quad d_i = \deg(\tilde{P}_{i-1}) - \deg(\tilde{P}_i), \end{cases} \tag{1}$$

ただし、剰余列は縮小 PRS あるいは部分終結式 PRS 算法で計算し、さらにノルムを 1 に規格化 (Normalize) するものとする (β_i はそのように選ぶ)。近似 PC-PRS 算法は次のようになる。

アルゴリズム 1 (近似 PC-PRS 算法)

Input :	Polynomials $P_1, P_2 \in \mathbb{C}[x,u]$ and a small number ε.
Output :	$G = \gcd(P_1, P_2\ ; \varepsilon)$.
STEP 1 :	主係数 $\mathrm{lc}(P_1)$ と $\mathrm{lc}(P_2)$ の近似 GCD g を計算
	$e := \min_{i=1,2}\{\mathrm{tdeg}_u(P_i) + \mathrm{tdeg}_u(g) - \mathrm{tdeg}_u(\mathrm{lc}(P_i))\}$.
STEP 2 :	Calculate PRS $(\tilde{P}_3, \cdots, \tilde{P}_k, \tilde{P}_{k+1}, ||\tilde{P}_{k+1}||/||\tilde{P}_k|| \le \varepsilon)$.
	/* cut-off higher degree E terms */
	if $\deg(\tilde{P}_k) = 0$ then return 1 else $\tilde{P}_k := \mathrm{Normalize}(\tilde{P}_k)$.
	$\tilde{P} := g\tilde{P}_k/\mathrm{lc}(\tilde{P}_k)$. /* power-series division */
	$G := \mathrm{pp}(\tilde{P}; \varepsilon)$.
	if $||\mathrm{rem}(P_1, G)|| \le \varepsilon$ and $||\mathrm{rem}(P_2, G)|| \le \varepsilon$
	then return G else return 1.
	end.

2.2 特異値分解に基づく算法

$f, g \in \mathbb{C}[x_1, \cdots, x_\ell]$ が与えられた時、$f_1 = f/\gcd(f,g)$、$g_1 = g/\gcd(f,g)$ とする。

補題 (Lemma 2.1 [GKMYZ04]) 方程式

$$uf + vg = 0 \tag{2}$$

を満たす解 u, v はすべて次の形である：

$$u = g_1 q, \quad v = -f_1 q, \text{ where } q \in \mathbb{C}[x_1, \cdots, x_\ell] \tag{3}$$

u と v は線形方程式でから求めることができる。

Gao *et al.* と Zeng-Dayton はいずれも 1 変数の Sylvester 行列を多変数に拡張した：Gao *et al.* の算法は Sylvester 行列 S を多項式の全次数表現をもとに構成し、S のランク落ちにより近似 GCD の全次数 k を定める。この全次数 k で再構成した Sylvester 行列を S_k と表して k 次 Sylvester 行列と呼ぶ。Zeng-Dayton の算法は各変数に関する 1 変数 GCD の次数の組 $\mathbf{k} = [k_1, \cdots, k_\ell]$ により近似 GCD の次数を定める。次数 $\mathbf{k}$ で構成した Sylvester 行列を $S_{\mathbf{k}}$ と表して $\mathbf{k}$ 次 Sylvester 行列と呼ぶ。f と g の余因子をそれぞれ u と v とするとき、それらに対応する係数ベクトルは S_k (or $S_{\mathbf{k}}$) の最小特異値に属する特異ベクトル (Sylvester 行列の null space) $[\mathbf{v}, -\mathbf{u}]^T$ で近似的に与えられる。近似 GCD は次の手順で求めることができる。

アルゴリズム 2 (特異値分解を用いる多変数多項式近似 GCD 算法)

1. f と g の Sylvester 行列 S_k (or $S_{\mathbf{k}}$) を構成する。
2. S_k (or $S_{\mathbf{k}}$) の null space $[\mathbf{v}, -\mathbf{u}]^T$ を求める [YZ05]。
3. (必要であれば) $[\mathbf{v}, -\mathbf{u}]^T$ を最適化する。

3 実験と考察

実験は Xeon 3.05GHz の Windows XP 上で Maple 9.5 と MATLAB 7.1 を用いて行った。ただし、近似 PC-PRS 算法の最大相対誤差は、Solaris 上で GAL の有効浮動小数 [KS97] を用いて計測した。計算時間はすべて Windows 上で計測したものである。近似 PC-PRS 算法は Maple と GAL に実装し、Zeng-Dayton の算法は MATLAB の `sparse` オプションを用いて実装を行い、Gao *et al.* の算法は [Kal04, Zhi04] にある Maple で実装されているものを用いた。

3.1 近似 PC-PRS 算法と特異値分解に基づく方法との比較 (効率)

下記の多変数多項式 f と g で三つの算法を比較した。表中の Ave. CPU と ErrMax はそれぞれ平均計算時間 (秒) と結果式の数係数の最大相対誤差を表し、*back_err* は次式で計算した [ZD04]：

$$back\_err = \sqrt{\frac{||\tilde{c}\tilde{f}_1 - f||_2^2 + ||\tilde{c}\tilde{g}_1 - g||_2^2}{||f||_2^2 + ||g||_2^2}}, \tag{4}$$

ただし、$\tilde{f}_1, \tilde{g}_1$ および $\tilde{c}$ はそれぞれ f, g の余因子および f と g の近似 GCD である。

サンプル 1.

$$\begin{cases} f(x,u) = f_1(x,u)c(x,u) \\ g(x,u) = g_1(x,u)c(x,u) \end{cases} \text{with} \begin{cases} c(x,u) = (x + u_1 + \cdots + u_r + 1)^2 \\ f_1(x,u) = (x^2 - u_1 - \cdots - u_r - 0.5)^2 \\ g_1(x,u) = (x^2 + u_1 + \cdots + u_r + 0.5)^2 \end{cases}.$$

	Approx. PC-PRS		Gao *et al.*'s GCD		Zeng-Dayton's GCD	
	Ave. CPU	ErrMax	Ave. CPU	*back_err*	Ave. CPU	*back_err*
$r = 1$	0.015	5.12e-13	2.647	5.62e-15	0.072	5.80e-15
$r = 2$	0.025	5.12e-13	10.286	1.16e-14	0.119	8.41e-14
$r = 3$	0.052	5.12e-13	96.206	1.36e-10	0.756	1.84e-13
$r = 4$	0.151	5.12e-13	about 1300	1.72e-10	1.519	1.16e-13
$r = 5$	0.220	5.12e-13	over 3000	- - - -	446.755	3.94e-13

表 1: 近似 PC-PRS 算法と特異値分解に基づく算法との比較 (サンプル 1)

サンプル 2.

$$\begin{cases} f(x,u) = f_2(x,u)c(x,u) \\ g(x,u) = g_2(x,u)c(x,u) \end{cases} \text{with} \begin{cases} c(x,u) = (x+u_1+\cdots+u_r^r+1)^2 \\ f_2(x,u) = (x^2-u_1-\cdots-u_r-0.5)^2 \\ g_2(x,u) = (x^2+u_1+\cdots+u_r+0.5)^2 \end{cases}.$$

	Approx. PC-PRS		Gao *et al.*'s GCD		Zeng-Dayton's GCD	
	Ave. CPU	ErrMax	Ave. CPU	*back_err*	Ave. CPU	*back_err*
$r=1$	0.015	5.12e-13	2.647	5.62e-15	0.072	5.80e-15
$r=2$	0.062	4.60e-12	11.114	1.30e-14	0.137	6.06e-14
$r=3$	0.130	8.19e-12	247.329	1.91e-10	1.988	7.94e-13
$r=4$	0.399	8.19e-12	over 3000	- - - -	338.754	7.88e-13
$r=5$	3.736	8.19e-12	over 3000	- - - -	over 3000	- - - -

表 2: 近似 PC-PRS 算法と特異値分解に基づく算法との比較 (サンプル 2)

サンプル 3.

$$\begin{cases} f(x,u) = f_3(x,u)c(x,u) \\ g(x,u) = g_3(x,u)c(x,u) \end{cases} \text{with} \begin{cases} c(x,u) = (x+u_1+\cdots+u_r+1)^2 \\ f_3(x,u) = (x^2-u_1-\cdots-u_r^r-0.5)^2 \\ g_3(x,u) = (x^2+u_1+\cdots+u_r+0.5)^2 \end{cases}.$$

	Approx. PC-PRS		Gao *et al.*'s GCD		Zeng-Dayton's GCD	
	Ave. CPU	ErrMax	Ave. CPU	*back_err*	Ave. CPU	*back_err*
$r=1$	0.015	5.12e-13	2.647	5.62e-15	0.072	5.80e-15
$r=2$	0.082	5.12e-13	7.899	1.47e-14	0.163	4.87e-15
$r=3$	0.151	5.12e-13	426.549	1.32e-10	83.323	2.62e-13
$r=4$	0.438	5.12e-13	3000 over	- - - -	over 3000	- - - -
$r=5$	2.167	5.12e-13	3000 over	- - - -	over 3000	- - - -

表 3: 近似 PC-PRS 算法と特異値分解に基づく算法との比較 (サンプル 3)

表 1, 表 2, 表 3 から、近似 PC-PRS 算法は変数の個数や次数が増えても計算時間はそれほど増加しないことがわかる。一方で特異値分解に基づく算法は、変数の個数や次数の増加に伴い急激に計算効率が悪くなることがわかる。これは変数の個数や次数の増加により S_k や S_k のサイズが急激に膨張し (表 4 参照)、null space を求める計算に時間がかかるためである。Rank-revealing Method [YZ05] のような反復法で高速に null space を計算する算法もあるが、QR 分解を必要とするため S_k や S_k のサイズが大きくなると計算は遅くなってしまう。なお、Gao *et al.* の算法は Zeng-Dayton の算法に比べると著しく遅くなっているが、後者の高速性は MATLAB の高速な行列演算によるものであり、Gao *et al.* の算法も MATLAB など行列演算を得意とするもので実装すれば速くなるが、表が示すように近似 PC-PRS 算法に及ばない。

S_k と $S_{\mathbf{k}}$ のサイズ

サンプル 1 とサンプル 3 で用いた多項式について、k 次 Sylvester 行列 S_k と k 次 Sylvester 行列 $S_{\mathbf{k}}$ のサイズは次表のようになる。

	サンプル 1 (表 1)		サンプル 3 (表 3)	
	S_k	$S_{\mathbf{k}}$	S_k	$S_{\mathbf{k}}$
$r=1$	66×30	77×30	66×30	77×30
$r=2$	286×70	539×90	286×70	819×120
$r=3$	1001×140	3773×270	1829×280	9009×660
$r=4$	3003×252	26411×810	20349×3129	99099×5130
$r=5$	8008×420	184877×2430	74613×8218	1756755×53190

表 4: Sylvester 行列のサイズ

S_k より $S_{\mathbf{k}}$ の方がサイズが大きいことがわかる。これは S_k が多項式の全次数表現に基づき行列を構成しているのに対し、$S_{\mathbf{k}}$ は各変数に関する 1 変数 GCD の次数をもとに構成しているためである。また、いずれの行列も巨大な疎行列である。行列のサイズは大きいが、各列の非零要素は f または g の非零係数のみであり、行列要素全体の 90%以上は 0 である。実装する際には巨大な疎行列に関して有能なパッケージを使う必要があり、今回は MATLAB で実装したが、巨大な疎行列演算ライブラリである SVDpack などの利用も有効と考えられる。

3.2 近似 PC-PRS 算法と特異値分解に基づく算法との比較 (精度)

表 5 はランダムに生成した多項式について、近似 PC-PRS 算法と特異値分解に基づく算法で近似 GCD を計算した結果である。10 個の多項式の組 (A_i, B_i) は次のようにしてランダムに生成した：

$$\begin{cases} A_i = a_i c_i + 10^{-2} d_i \\ B_i = b_i c_i + 10^{-2} e_i \end{cases} (i = 1, \cdots, 5), \quad \begin{cases} A_{i+5} = a_i c_i + 10^{-5} d_i \\ B_{i+5} = b_i c_i + 10^{-5} e_i \end{cases} (i = 1, \cdots, 5),$$

ただし、$a_i, b_i, c_i, d_i, e_i \in \mathbb{C}[x, y]$ $(i = 1, 2, 3)$ かつ $a_i, b_i, c_i, d_i, e_i \in \mathbb{C}[x, y, z]$ $(i = 4, 5)$ であり、$||a_i|| = ||b_i|| = ||c_i|| = ||d_i|| = ||e_i|| = 1$ である。

	Approx. PC-PRS		Gao *et al.*'s GCD		Zeng-Dayton's GCD	
Ex.	Ave. CPU	ErrMax	Ave. CPU	*back_err*	Ave. CPU	*back_err*
1	0.025	7.27e-12	0.871	9.65e-3	0.094	3.52e-1
2	0.044	3.00e-14	1.825	1.66e-2	0.094	4.83e-1
3	0.029	1.37e-13	3.670	1.25e-2	0.090	5.13e-1
4	0.110	3.47e-7	4.892	1.66e-2	0.253	3.32e-0
5	— Error —		4.119	1.18e-2	0.294	2.44e-0
6	0.037	4.68e-11	2.170	9.65e-6	0.098	2.08e-5
7	0.040	1.39e-13	1.121	1.66e-5	0.087	1.80e-5
8	0.039	1.66e-9	0.341	1.25e-5	0.082	1.95e-5
9	0.049	3.22e-6	2.503	1.66e-5	0.102	2.24e-5
10	0.471	1.90e-8	2.563	1.18e-5	0.100	1.76e-5

表 5: 近似 PC-PRS 算法と特異値分解に基づく算法との比較

表5より特異値分解に基づく算法は摂動が少し大きいと不安定であることがわかる (Ex.1 ～ Ex.5)。Gao *et al.* の算法は、Sylvester 行列のランク落ちにより得られた次数 k で S_k を構成し、多項式の余因子に対応する S_k の null space を特異値分解に基づき定めるが、摂動の大きさに非常に敏感である。Ex.3 を見ると、全次数は 1 であるが、計算によって得られる近似 GCD の全次数が 3 であったため、正しい近似 GCD を得られない場合が多々あった。Zeng-Dayton の算法についても同様のことが言えるが、Zeng-Dayton の算法は、近似 GCD の次数を各変数に関する 1 変数 GCD の次数で定めるため、さらに不安定になりやすい。

Ex.4 と Ex.9 では $||B_4||$ と $||B_9||$ がそれぞれ小さい。このため近似 PC-PRS 算法は大きな桁落ちを起こしている。一方で、特異値分解に基づく算法は係数のノルムによる影響は受けない。Ex.5 で "Error" は計算が失敗して終了したことを表す：エラーは $\mathrm{lc}(A_5)$ と $\mathrm{lc}(B_5)$ の近似 GCD を求める際に起きた。$||\mathrm{lc}(A_5)|| < 0.01$ であるため、$\gcd(\mathrm{lc}(A_5), \mathrm{lc}(B_5); 0.01) = \gcd(0, \mathrm{lc}(B_5; 0.01)) = 0$ となって不具合が起きたのである。

4 まとめ

近似 PC-PRS 算法と特異値分解に基づく算法との比較を行った。近似 PC-PRS 算法は従変数の高次項を打ち切る算法であり、精度の良さと高効率性を確かめることができた。しかし、PRS 算法同様、微小主係数の場合に大きな桁落ちを起こすので、この点を考慮したアルゴリズムの改良が必要である。特異値分解に基づく算法は、変数の個数が少ない場合や次数が低い場合には有効であり、PRS 算法と異なり主係数の大きさに影響されない算法である。次数が大きい場合や変数の個数が多い場合、Sylvester 行列のサイズが急激に大きくなるが、MATLAB などの行列演算に優れたシステムを使えばある程度は計算できるだろう。しかし、PC-PRS 算法のような速度は望めない。

今後は近似 PC-PRS 算法が微小主係数の場合にも精度の良さと効率性を保つように、アルゴリズムを改良していきたい。

謝　辞

本研究を遂行するにあたり、多岐にわたってアドバイスをして頂いた筑波大学 佐々木建昭教授に感謝致します。

参 考 文 献

[CGTW95] R. Corless, P. Gianni, B. Trager and S. Watt, *The Singular Value Decomposition for Polynomial Systems*, Proc. of ISSAC'95, ACM, 1995, 195–207.

[GKMYZ04] S. Gao, E. Kaltofen, J. P. May, Z. Yang and L. Zhi, *Approximate Factorization of Multivariate Polynomials via Differential Equations*, Proc. of ISSAC'04, ACM, 2004, 167–174.

[Kal04] E. Kaltofen, `http://www4.ncsu.edu/~kaltofen/software/appfac/issac04_mws/`, 2004.

[KS97] F. Kako and T. Sasaki, *Proposal of "Effective Floating-point Number" for Approximate Algebraic Computation*, Preprint of Tsukuba Univ., 1997.

[MY73] J. Moses and D. Y. Y. Yun, *The EZGCD Algorithm*, Proc. ACM National Conference, ACM, 1973, 159–166.

[ONS91] M. Ochi, M-T. Noda and T. Sasaki, *Approximate Greatest Common Divisor of Multivariate*

Polynomials and Its Application to Ill-Conditioned Systems of Algebraic Equations, J. Inform. Proces., **14** (1991), 292–300.

[San05] M. Sanuki, *Computing Approximate GCD of Multivariate Polynomials (extended abstract)*, Proc. of SNC2005, Dongming Wang and Lihong Zhi editors, 2005, 308–314.

[San06] M. Sanuki, *Computing Approximate GCD of Multivariate Polynomials*, submited, 2006.

[SN89] T. Sasaki and M-T. Noda, *Approximate Square-free Decomposition and Root-finding of Ill-conditioned Algebraic Equations*, J. Inform. Proces., **12** (1989), 159–168.

[SS92] T. Sasaki and M. Suzuki, *Three New Algrithms for Multivariate Polynomial GCD*, J. Symbolic Comput., **13**(1992), 395–411.

[SY98] T. Sasaki and S. Yamaguchi, *An Analysis of Cancellation Error in Multivariate Hensel Construction with Floating-point Number Arithmetic*, Proc. of ISSAC'98, ACM, 1998, 1–8.

[Wan80] P. S. Wang, *The EEZ-GCD Algorithm*, SIGSAM Bulletin **14** (1980), 50–60.

[YZ05] T. Y. Li and Z. Zeng, *A Rank-Revealing Method with Updating, Downdating, and Applications*, SIAM J. Matrix Anal. Appl., **26** (2005), no. 4, 918–946.

[ZD04] Z. Zeng and B. H. Dayton, *The Approximate GCD of Inexact Polynomials Part II: A Multivariate Algorithm*, Proc. of ISSAC'04, ACM, 2004, 320–327.

[Zhi04] L. Zhi, `http://www.mmrc.iss.ac.cn/~lzhi/Research/issac04_mws.html`, 2004.

[ZN00] L. Zhi and M-T. Noda, *Approximate GCD of Multivariate Polynomials*, Proc. of ASCM2000, World Scientific, 2000, 9–18.

[ZY04] L. Zhi and Z. Yang, *Computing Approximate GCD of Multivariate Polynomials by Structure Total Least Norm*, MM Research Preprint, MMRC, AMSS, Academia, Sinica, 2004, No. 23, 388–401.