数理解析研究所講究録
第1814巻 2012年 148-159

チェビシェフ展開を用いたスツルム法による高次代数方程式の実根の分離

村上 弘
HIROSHI MURAKAMI
首都大学東京 数理情報科学専攻
DEPARTMENT OF MATHEMATICS AND INFORMATION SCIENCES,
Tokyo Metropolitan University *

要約：実係数高次代数方程式の区間内の実根をスツルム法を用いて浮動小数点演算により求める．多項式の展開基底には単項式ではなくてチェビシェフ多項式を用いる．チェビシェフ基底は標準区間 $[-1,1]$ 内で線形独立性が高いので丸め誤差の実根の計算値に与える影響を減らせる．N 次多項式の係数摂動に対する $[-1,1]$ 内の根の条件数は，単項式基底では $O((\sqrt{2}+1)^N)$，チェビシェフ基底では $O(N)$ となることが知られている [9]．浮動小数点演算を用いた計算実験の例を示し，また記憶量，演算量を低減する方法も示す．

1 スツルム列

N 次多項式 F に対して，$F_0{=}F$，$F_1{=}F'$ と，符号が逆の剰余：

$$F_k := -\text{remainder}(F_{k-2}, F_{k-1}),\ \text{for } k{\geq}2$$

で多項式の列を生成するとき，零になる前までの多項式の列 $[F_0, F_1, \ldots, F_\ell]$，$(\ell{\leq}N)$ を F のスツルム列という．F_ℓ は定数倍を除き $\text{GCD}(F, F')$ に等しい．

スツルムの定理（文献 [1]） $V(x)$:スツルム列の x に於ける符号変化数．符号変化数は，列 $[F_0(x), F_1(x), \ldots, F_\ell(x)]$ の値を（値 0 を飛ばして）順番にみて数える．区間 $(a,b]$ に於ける F の実根の個数は $V(a){-}V(b)$ に等しい．（重複根は 1 個と数える）．但し，端点 a が $F(x)$ の k 重根 $(k>1)$ の場合は，符号変化数は，スツルム列を $(x{-}a)^{k-1}$ で割った後に数える．端点 b についても同様．（この注意は端点が重複根でなければ不要）．

このスツルムの定理を用いると区間内の実根がカウントできる．区間縮小法を用いれば実根の分離（isolate）が可能である．

2 チェビシェフ多項式 $T_n(x)$

チェビシェフ多項式 T_n は $T_0(x)=1$，$T_1(x)=x$，$T_{n+1}(x)=2xT_n(x)-T_{n-1}(x)$ で定義される n 次多項式で，区間 $[-1,1]$ 内での一様近似に最も便利な直交多項式であり，性質 $T_n(\cos\theta)=\cos(n\theta)$，$\int_{-1}^{1} T_j(x)T_k(x)\frac{dx}{\sqrt{1-x^2}} = \delta_{j,k}(1+\delta_{j,0})\frac{\pi}{2}$，などを持つ．最初の 4 次までの多項式のグラフを図 1 に示す．

*mrkmhrsh@tmu.ac.jp

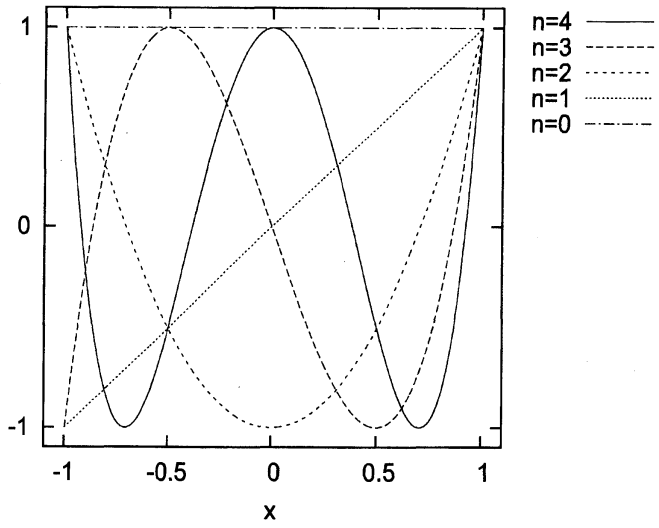


図 1: チェビシェフ多項式.

2.1 チェビシェフ展開

実根の探索区間 $t \in [\alpha, \beta]$ を線形変換で，標準区間 $x \in [-1, 1]$ に移した N 次多項式を $F(x)$. $F(x)$ のチェビシェフ展開：$F(x) = \sum_{k=0}^{N} c_k T_k(x) = c_0 T_0(x) + c_1 T_1(x) + \cdots + c_N T_N(x)$ の係数 c_k は，$N+1$ 個の分点 $x_\ell = -\cos\frac{(\ell+\frac{1}{2})\pi}{N+1}$，$\ell = 0, \ldots, N$ での $F(x)$ の値から求まる．$N+1$ 個の分点での値から $N+1$ 個の展開係数を求める演算量は素朴な方法では $O(N^2)$ だが，高速コサイン変換を用いると $O(N \log_2 N)$ になる [12][10]. 以降で，多項式はチェビシェフ展開係数で与えられるとする.

2.2 展開係数で与えられた多項式の値の計算

任意分点 x でのチェビシェフ展開の値の計算：$F(x) = \sum_{k=0}^{N} c_k T_k(x) = c_0 T_0(x) + c_1 T_1(x) + \cdots + c_N T_N(x)$ は，Clenshaw の算法 (図 2 の **Algorithm 1**) を用いて演算量 $O(N)$ で計算できる.

図 2: Algorithm 1：Clenshaw の算法.

```
Q2 ← 0 ; Q1 ← 0 ; Y ← 2 * x ;
FOR j ← N STEP (-1) UNTIL 1 DO
    Q0 ← Y * Q1 - Q2 + c_j ;
    Q2 ← Q1 ; Q1 ← Q0
OD ;
Q0 ← x * Q1 + (c_0 - Q2) ;
RETURN Q0
```

2.3 導関数の展開係数

チェビシェフ多項式 $T_k(x)$ の導関数は $kU_{k-1}(x)$. $U_\ell(x)$ は次数 ℓ の第二種チェビシェフ多項式. $U_\ell(x)$ による展開は，関係式 $U_k(x)$=$2T_k(x)+U_{k-2}(x)$, $U_1(x)$=$2T_1(x)$, $U_0(x)$=$T_0(x)$ を用いて，通常のチェビシェフ展開に変換. N 次の多項式 $F(x)$ のチェビシェフ展開係数 c_k, k=0,...,N から，その導関数 $G(x)=F'(x)$ のチェビシェフ展開係数 d_k, k=0,...,$N-1$ が，図3の **Algorithm 2** を用いて演算量 $O(N)$ で計算できる.

図 3: Algorithm 2：多項式の導関数のチェビシェフ係数.

FOR k ← 1 TO N DO d_{k-1} ← k * c_k OD ;
FOR k ← $N-1$ STEP (-1) UNTIL 2
  DO d_{k-2} ← $d_{k-2}+d_k$; d_k ← 2 * d_k OD ;
d_1 ← 2 * d_1

2.4 チェビシェフ展開形による多項式剰余の計算

次数 m の多項式 $A(x)$ を 次数 n の多項式 $B(x)$ で割り，$(m-n)$ 次の商 $Q(x)$ と**高々**$(n-1)$ 次の剰余多項式 $R(x)$ を求める. スツルム列の生成では，**殆んどの場合に**次数が1だけ違う割算. 割算には積公式を利用：

$$T_i(x)\cdot T_j(x)=\frac{1}{2}\left(T_{i+j}(x)+T_{|i-j|}(x)\right).$$

割算の手順を以下に Fortran90 記法で示す（図4の **Algorithm 3**）. a(0:m) は $A(x)$ の係数の配列，b(0:n) は $B(x)$ の係数の配列で，算法は剰余 $R(x)$ の係数を a(0:n-1) の場所に，商 $Q(x)$ の係数を a(n:m) の場所にそれぞれ重ね書きする. 作業配列 h(0:m) の使用は分かり易さのためで，算法から除くことも可能である.

図 4: Algorithm 3：チェビシェフ係数による多項式の割算.

```
DO  k = m, n, -1
   i = k - n
   h(0:k) = 0.0
   DO  j = 0, n
      T = 0.5 * b(j)
      h(i+j) = h(i+j) + T
      h(ABS(i-j)) = h(ABS(i-j)) + T
   ENDDO
   a(k) = a(k)/h(k)
   a(0:k-1) = a(0:k-1) - a(k) * h(0:k-1)
ENDDO
```

3 スツルム二分法による実根の分離と精度改良

任意区間内の実根がカウントできるので，区間縮小法で元の区間内の実根を分離（**isolate**）できる．分離後は根を包む区間幅を狭めて精度改良（**refine**）もできる．N を多項式の次数，r を元の区間の中にある実根の数，分離区間幅の元の区間幅に対する比の下限（許容値）を ε とする．分点 1 個について符号変化数 V を計算する演算量は素朴な方法では $O(N^2)$．スツルム二分法で実根を分離（**isolate**）するための符号変化数 V の評価回数は $O(r+\log_2(r/\varepsilon))$ 程度，実根の分離後の近似精度の向上（**refine**）の計算量は，「多項式 F の値の符号だけを用いる二分法」では $O(r\,\log_2(r/\varepsilon)\cdot N)$ となる．

3.1 実根の分離区間の精密化（refinement）

実根を 1 個含む分離区間の精密化には，収束の速い「挟み撃ち法（割線法+異符号）」が利用可能である．但し，浮動小数点演算では丸め誤差の影響に注意が必要である．数値計算的には二分法が最も頑丈である．

3.2 剰余の再規格化について

話を単純にするため，スツルム列生成の際の剰余の再規格化は述べてこなかった．符号変換数 $V(z)$ はスツルム列の各多項式に正数を乗じても変わらない．数値計算で多項式を再規格化するのは，浮動小数点演算のアンダーフローで仮数部の下位が失われることによる無駄な精度低下を避けるためである．再規格化自身による丸め誤差の導入を避けるために，浮動小数点数の基数（IEEE-754 では 2）を b とし，乗数の形を $\gamma_k = b^e$（但し指数 e は整数）に制限する．そうして元の剰余列の計算式を次のように変更する：

$$F_k := -\gamma_k \cdot \mathrm{remainder}(F_{k-2}, F_{k-1}),\quad \text{for } k \geq 2.$$

指数 e は多項式 F_k の絶対値最大の係数が 1 付近となるようにとる．

4 数値実験の例

計算システムの仕様　実験に用いた計算システムの仕様を表 1 に示す．（もしも計算精度を倍精度から四倍精度にすると，intel Fortran では経過時間が約 30 倍程度となる．）

表 1: 計算システムの仕様

CPU	Intel Core i7 920 (2.66GHz,8MB L3) (1 コアのみ使用).
メモリ	DDR3-1333 PC3-10600 2GB × 6=12GB (triple channel).
コンパイラ	Intel Fortran v11.1 for intel64 (オプション`-fast`).
浮動小数点演算	IEEE 64-bit 倍精度.
OS	Fedora10 for intel64.

例題のチェビシェフ係数　例題の N 次多項式 $F(x)$ のチェビシェフ係数 c_k，k=0,...,N の値は，再現が容易な数式により，以下のように与えた：

$$c_k = \begin{cases} \dfrac{r_k}{\sqrt{k+1}} \ (\text{for } k < N), \\ \\ 10^{-12} \quad (\text{for } k = N) \end{cases}$$

ここで $r_k = \cos\{(k+1)^2\}$ は乱数の代用である.

図 5: 区間 $[-1, 1]$ 内の多項式のグラフ. 100 次多項式（区間内の実根 34 個）.

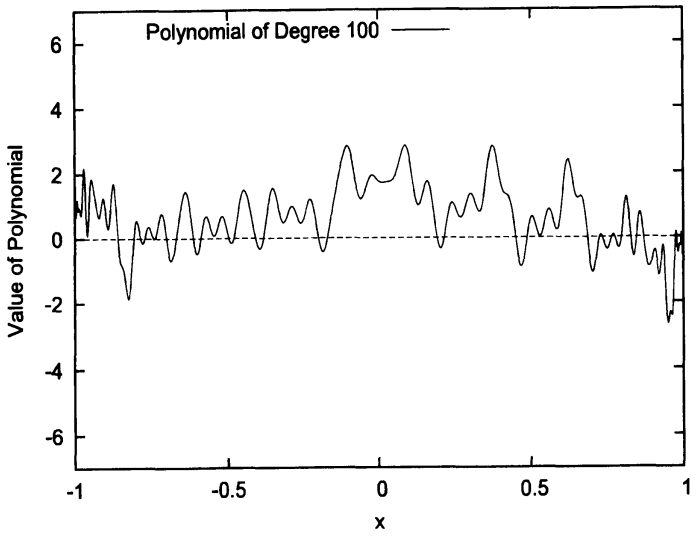


図 6: 区間 $[-1, 1]$ 内の多項式のグラフ. 300 次多項式（区間内の実根 86 個）.

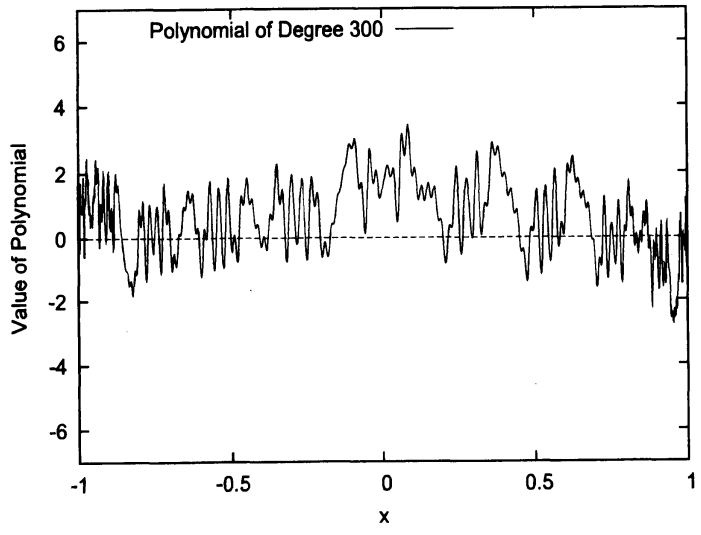


図 7: 区間 $[-1,1]$ 内の多項式のグラフ. 1000 次多項式（区間内の実根 184 個）.

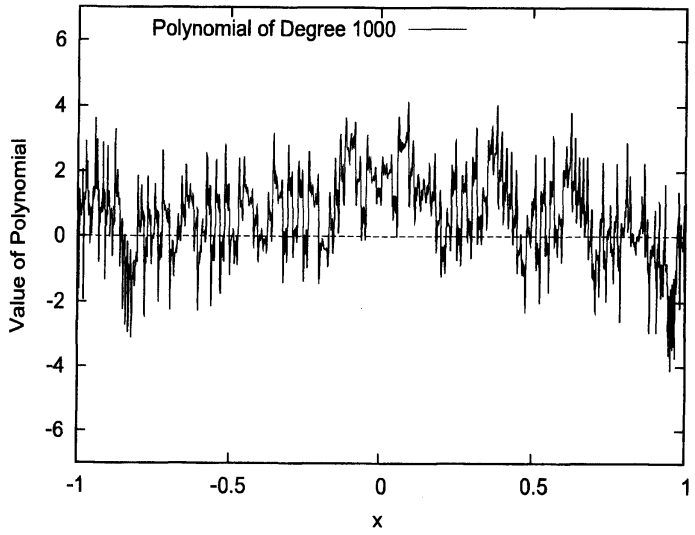
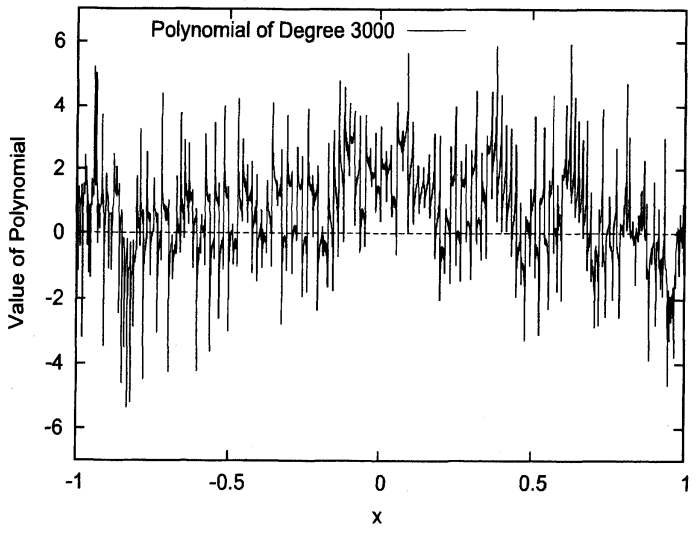


図 8: 区間 $[-1,1]$ 内の多項式のグラフ. 3000 次多項式（区間内の実根 388 個）.



4.1 「素朴な計算法」

スツルムの方法で素朴に計算すると，スツルム列を生成し保持するのに記憶量 $O(N^2)$，演算量 $O(N^2)$ が必要となり，その後の 1 個の分点 z について符号変化数 $V(z)$ を求めるたびに記憶参照量 $O(N^2)$，演算量 $O(N^2)$ が必要となる．そのため N が大きく実根の個数が多い場合には記憶量も演算量も多くて計算が重い．

実験結果（素朴法） 素朴な方法による，区間 $[-1,1]$ 内の実根のカウントの経過時間を表 2 に，また実根の isolation と精度 10^{-8} までの refinement の経過時間を表 3 に示す．

N を次数，r を区間内の実根の個数とすると，スツルム列を作る経過時間，一個の区間 $[-1,1]$ 内の実根をカウントする経過時間はそれぞれ N^2 に比例し，isolation の経過時間は rN^2 に比例している．生成されたスツルム列全体を参照する isolation に比べて，多項式 F だけを参照する refinement は記憶参照も演算量も共に軽い．

表 2: 区間 $[-1,1]$ 内の実根のカウントの経過時間（素朴法）．

次数 N	スツルム列生成 経過時間（秒）	実根のカウント 経過時間（秒）	カウントされた 実根の個数
100	6.3E-5	6.1E-5	34
300	3.9E-4	4.8E-4	86
1000	3.8E-3	5.2E-3	184
3000	3.7E-2	4.7E-2	388
10000	4.6E-1	5.2E-1	1355
30000	4.8E+0	4.8E+0	6145

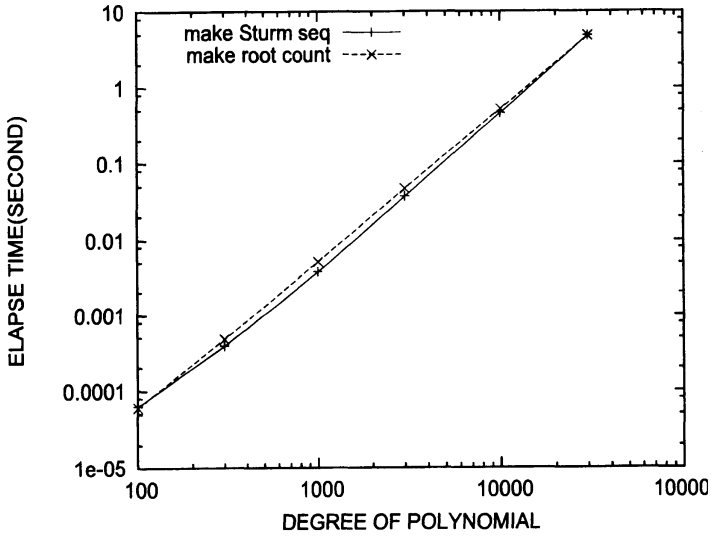


図 9: 区間 $[-1,1]$ 内の実根のカウントの経過時間（両対数軸）（素朴法）．

4.2 「記憶量を低減する方法」

分点 z での符号変化数 $V(z)$ を求めるためにスツルム列を計算するが，使い捨てにして保持しないことで，必要な作業記憶量を $O(N)$ に下げることができて，1 個の分点 z で $V(z)$ を求めるための作業用記憶量が $O(N)$，演算量は $O(N^2)$ となる．必要な作業記憶量は減るが，スツルム列全体の係数を毎回計算するため，素朴法よりも演算量が増えるのが欠点である．

剰余列の三項関係を用いてスツルム列の多項式を順番に生成し，その中で多項式が生成されたらすぐに分点 z での符号を求めて $V(z)$ に反映させる．生成された多項式は後に続く 2 個の多項式の生成に用いた後は廃棄が可能である．これにより必要な作業記憶量を $O(N)$ に低減する．

表 3: 実根の isolation と精度 10^{-8} までの refinement の経過時間（素朴法）.

次数 N	スツルム列の生成 経過時間（秒）	実根の 個数	isolation 経過時間（秒）	refinement 経過時間（秒）
100	6.3E-5	34	3.2E-3	3.6E-4
300	3.9E-4	86	5.9E-2	2.6E-3
1000	3.8E-3	184	1.5E+0	1.7E-2
3000	3.7E-2	388	2.9E+1	9.7E-2
10000	4.6E-1	1355	1.2E+3	9.9E-1

実際の計算では多項式 2 個分の記憶を用意し，順次生成されるスツルム列の多項式を交互に重ね書きする．多項式 F_i が生成されたら直ちに z での符号を求め，符号変化があれば $V(z)$ のカウントを増す．

この方法では F_i の p 個の分点の組 $z_1, z_2, \ldots, z_p$ での符号の組を並行して求めることで，符号変化数 $V(z_1), V(z_2), \ldots, V(z_p)$ を同時並行に求めることが可能であり，そのようにすればスツルム列の再生成が p 個の間で兼用できて演算量が節約できる．

実験結果（記憶を低減する方法） 記憶を低減する方法を用いて，区間 $[-1, 1]$ 内の実根のカウントの経過時間を表 4 に（注：カウントする際には同時にスツルム列も計算している），また実根の isolation と精度 10^{-8} までの refinement を行った経過時間を表 5 に示す．

表 4: 区間 $[-1, 1]$ 内の実根のカウントの経過時間（記憶量低減法）.

次数 N	実根のカウント 経過時間（秒）	カウントされた 実根の個数
100	1.6E-4	34
300	1.2E-3	86
1000	1.2E-2	184
3000	1.1E-1	388
10000	1.3E+0	1355
30000	1.2E+1	6145
100000	1.4E+2	22952

表 5: isolation と refinement の経過時間（記憶量低減法）.

次数 N	実根の 個数	isolation 経過時間（秒）	refinement 経過時間（秒）
100	34	7.9E-3	3.6E-4
300	86	1.2E-1	2.6E-3
1000	184	2.8E+0	1.7E-2
3000	388	5.6E+1	9.7E-2
10000	1355	2.4E+3	9.9E-1

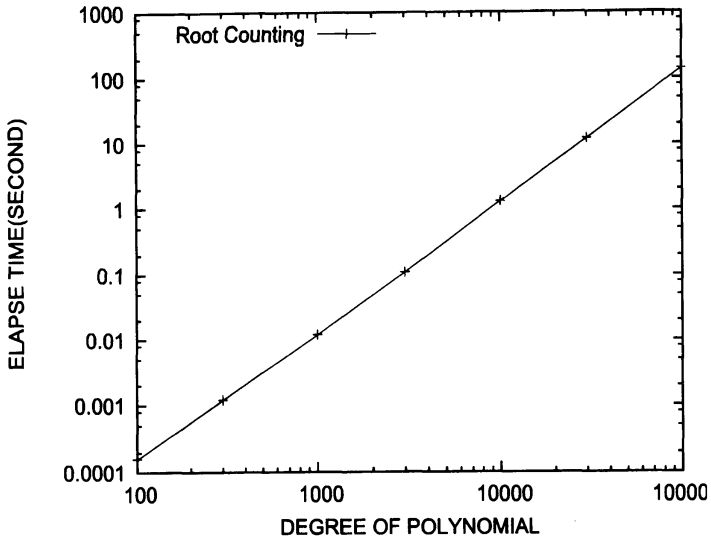


図 10: 区間 $[-1,1]$ 内の実根のカウントの経過時間（両対数軸）（記憶量低減法）.

4.3 「記憶量と演算量の両方を低減する方法」

スツルム列の生成時に，商多項式と規格化乗数だけを保持する：

$$F_i \Rightarrow Q_i F_{i+1} - \widehat{F}_{i+2}\ ;\ \ F_{i+2} := \gamma_{i+2} \cdot \widehat{F}_{i+2}\ .$$

商多項式 Q_i は通常（殆んどの場合は 1 次の）低次多項式で，商多項式全ての次数の和は N 以下である.

分点 z での値 $F_i(z)$ と $F_{i+1}(z)$ と γ_{i+2} から，値 $F_{i+2}(z)$ が求まる：

$$F_{i+2}(z) := -\gamma_{i+2}\ (F_i(z) - Q_i(z)F_{i+1}(z))\ .$$

このことから，$F_0{=}F$，$F_1{=}F'$ と規格化乗数の列 $\gamma_2,\ldots,\gamma_\ell$，及び商多項式の列 $Q_0, Q_1, \ldots, Q_{\ell-2}$ だけを保持すれば，任意の分点でのスツルム列の値や符号の列が求まる.

商多項式の列と規格化乗数を保持するのに必要な記憶量は $O(N)$ で，それらが保持されていれば，任意の分点 z でのスツルム列の符号変化数 $V(z)$ を求める演算量は $O(N)$ になる.

まずスツルム列を多項式 2 個分の記憶場所を用いて記憶量 $O(N)$，演算量 $O(N^2)$ で計算する．その際に，スツルム列の多項式 F_ℓ は（最初の $F_0{=}F$, $F_1{=}F'$ 以外は）保持せずに，規格乗数の列 $\gamma_2, \gamma_3, \ldots, \gamma_\ell$ と商多項式の列 $Q_0, Q_1, \ldots, Q_{\ell-2}$ だけを合計記憶量 $O(N)$ で保持する．$F_0 = F$，$F_1 = F'$ と商多項式の列，乗数の列を用いると，任意区間での実根のカウントが記憶参照量 $O(N)$，演算量 $O(N)$ でできる．区間内の全ての実根を（相対許容幅 ε）で分離する演算量は $O\left((r + \log_2 \frac{1}{r\varepsilon}) \cdot N\right)$ で高速になる.

但し，素朴法のようにスツルム列を多項式として求めてその符号を計算するのとは演算の過程が異なるため，数値誤差の影響により符号変化数の計算結果が一致しないことがあり得る.

実験結果（記憶と演算の低減法）　記憶と演算の低減法による，区間 $[-1,1]$ 内の実根のカウントの経過時間を表 6 に，また実根の isolation と精度 10^{-8} までの refinement の経過時間を表 7 に示す.

表 6: 区間 $[-1,1]$ 内の実根のカウントの経過時間（記憶と演算の低減法）.

次数 N	商多項式列生成 経過時間（秒）	実根のカウント 経過時間（秒）	カウントされた 実根の個数
100	4.9E-5	4.1E-6	34
300	2.4E-4	1.3E-5	86
1000	2.1E-3	4.3E-5	184
3000	2.2E-2	1.2E-4	388
10000	2.8E-1	4.0E-4	1355
30000	3.1E+0	1.3E-3	6145
100000	3.9E+1	4.4E-3	22953

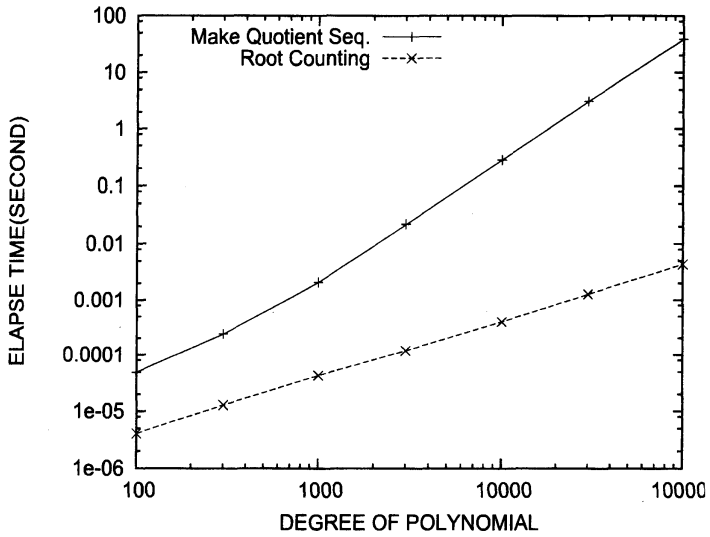


図 11: 実根のカウントの経過時間（両対数軸）（記憶と演算の低減法）.

表 7: 実根の isolation と精度 10^{-8} までの refinement の経過時間（記憶と演算の低減法）.

次数 N	実根の 個数	isolation 経過時間（秒）	refinement 経過時間（秒）
100	34	1.1E-3	3.6E-4
300	86	4.4E-3	2.6E-3
1000	184	2.6E-2	1.7E-2
3000	388	1.5E-1	9.7E-2
10000	1355	1.7E+0	9.9E-1
30000	6145	1.9E+1	1.2E+1
100000	22953	2.5E+2	1.4E+2

5 「素朴法」と「記憶と演算を低減する方法」の比較

「素朴法」と「記憶と演算を低減する方法」のそれぞれによる区間 $[-1,1]$ 内の実根のカウントの経過時間の比較を表 8 に示す．また区間 $[-1,1]$ 内の実根の isolation の経過時間の比較を表 9 に示す（refinement の方法は両方で同じなので経過時間は共通である）．

表 8: 区間 $[-1,1]$ 内の実根のカウントの経過時間（「素朴法」と「記憶と演算の低減法」の比較）．

	素朴法		記憶と演算の低減法	
次数 N	スツルム列生成 経過時間（秒）	カウント 経過時間（秒）	商多項式列生成 経過時間（秒）	カウント 経過時間（秒）
100	6.3E-5	6.1E-5	4.9E-5	4.1E-6
300	3.9E-4	4.8E-4	2.4E-4	1.3E-5
1000	3.8E-3	5.2E-3	2.1E-3	4.3E-5
3000	3.7E-2	4.7E-2	2.2E-2	1.2E-4
10000	4.6E-1	5.2E-1	2.8E-1	4.0E-4
30000	4.8E+0	4.8E+0	3.1E+0	1.3E-3

表 9: 実根の isolation と精度 10^{-8} までの refinement の経過時間（「素朴法」と「記憶と演算の低減法」の比較）．

		素朴法	記憶と演算の低減法	
次数 N	実根の 個数	isolation 経過時間（秒）	isolation 経過時間（秒）	refinement 経過時間（秒）
100	34	3.2E-3	1.1E-3	3.6E-4
300	86	5.9E-2	4.4E-3	2.6E-3
1000	184	1.5E+0	2.6E-2	1.7E-2
3000	388	2.9E+1	1.5E-1	9.7E-2
10000	1355	1.2E+3	1.7E+0	9.9E-1

6 今後の課題（算法の安定性や誤差の振舞い）

多項式 $F_i(x)$ を多項式 $F_{i+1}(x)$ で割って商と剰余を求める計算：$F_i(x) \Rightarrow Q_i(x)\,F_{i+1}(x) - F_{i+2}(x)$ に於いて，通常は剰余 $F_{i+2}(x)$ の次数は $F_{i+1}(x)$ の次数よりも 1 だけ小さいが，そうならない場合も生じ得る．誤差を含む近似演算では，剰余の最高次係数が 0 に等しいかが決定ができず，次数が確定しないことがある．そのような場合に，例えば $F_{i+2}(x)$ の次数が $F_{i+1}(x)$ の次数よりも常に 1 だけ小さいと見なして計算を続け，最高次係数が微小になるとその後の計算が不安定になる．あるいは，計算を「安定化」させるため，微小な項を 0 に置き換えると，結果はどれだけ変化するか．係数の微小な項を 0 に置き換える場合，算法は後退誤差の意味で安定であろうか?

浮動小数点演算を用いて数学的に確実な結果を得るには，文献 [2] の区間演算や丸め制御付き演算（Round-Up, Round-Down）などにより近似値の数学的に正しい上限下限を計算する．演算精度の不足により途中の符号判定や零判定が不確実であればそれを検出して誤りを回避できるので，必要に応じて多倍長浮動小数点演算をこのような誤差評価と併用することになる．

7 まとめ

スツルム法により高次多項式の実根の分離を浮動小数点演算でなるべく高精度に行うためにチェビシェフ展開形を用いた.

素朴なスツルム法では必要記憶量は $O(N^2)$ で，スツルム列を最初に1回だけ構成するための演算量が $O(N^2)$，分点1個あたりの符号変化数 $V(z)$ を求める演算量が $O(N^2)$ である．これに対して，記憶と演算を低減する方法では，スツルム列を求める際の商多項式列を保存し，スツルム列自身は保存しないことで必要記憶量 $O(N)$ であり，最初に商多項式列を1回だけ構成するための演算量は $O(N^2)$ だが，その後の分点1個あたりの符号変化数 $V(z)$ を求める演算量は $O(N)$ となる（この技法は有理数演算の場合でも記憶量や演算量を減らすために有用であろう）.

数学的に確実な結果を浮動小数点演算を用いて得るための方法（区間演算，丸め制御演算，可変多倍長浮動小数点演算）の実験は今後の課題である.

参 考 文 献

[1] 高木貞治:「代数学講義(改訂新版)」, 第3章:'スツルムの問題, 根の計算', 共立出版 (1965).

[2] 大石進一:「精度保証付き数値計算」, コロナ社 (2000).

[3] Battles, Z. and Trefethen, L. N.: An Extension of Matlab to Continuous Functions and Operators, SIAM J. Sci. Comput., **v25**, No.5 (2004), pp.1743–1770.

[4] Boyd, J.P.: A Chebyshev Polynomial Interval-searching Method ("Lanczos Economization)" for Solving a Nonlinear Equation with Application to the Nonlinear Eigenvalue Problem, J.Comp.Phys., **v118** (1995), pp.1–8.

[5] Boyd, J.P.: Computing Zeros on a Real Interval Through Chebyshev Expansion and Polynomial Rootfinding, SIAM J. Numer. Anal., **v40**, No.5 (2002), pp.1666–1682.

[6] Boyd, J.E.: Computing Real Roots of a Polynomial in Chebyshev Series form Through Subdivision, Applied Numerical Math., **v56**, No.8 (2006), pp.1077–1091.

[7] Boyd, J.P.: Computing real roots of a polynomial in Chebyshev series form through subdivision with linear testing and cubic solves, Applied Math. Comput., **174** (2006), pp.1642–1658.

[8] Boyd, J.P. and Gally, D.H.: Numerical Experiments on the Accuracy of the Chebyshev-Frobenius Companion Matrix Method for Finding the Zeros of a Truncated Series of Chebyshev Polynomials, *J. Comput. and Appl. Math.*, **v205**, No.1 (2007), pp.281–295.

[9] Day, D. and Romero, L.: Roots of Polynomials Expressed in Terms of Orthogonal Polynomials, SIAM J. Numer. Anal., **v43**, No.5 (2005) pp.1969–1987.

[10] Hasegawa, T., Torii, T. and Sugiura, H.: An algorithm based on the FFT for a generalized Chebyshev interpolations, Math. Comp., **v54** (1990), pp.195–210.

[11] Jónsson, G.F. and Vavasis, S., Solving polynomials with small leading coefficients, SIAM J. Matrix Anal. Appl., **v26** (2004), pp.400–414.

[12] Sherlock, B.G. and Monro, D.M.: Algorithm 749:fast discrete cosine transform, ACM Trans. Math. Soft., **v21**, Issue 4 (1995), pp.372–378.