数理解析研究所講究録
1514巻 2006年 122-124

多項式の実数解を求める方法について — 再訪 —

平野 照比古
HILANO TERUHIKO
神奈川工科大学情報学部
FACULTY OF INFORMATICS, KANAGAWA INSTITUTE OF TECHNOLOGY*

1 アルゴリズム概観

$f(x)$ を有理数を係数とする多項式とする. この多項式の実解を与えられた精度で求める方法として次のようなアルゴリズムを提案した.[4]

1. $f(x)$ から次の性質を持つ多項式列 $f_1(x), f_2(x), \ldots, f_n(x)$ を作る.
   - $f(x)$ と $f_1(x)$ は同じ実解を持つ.
   - $f_k(x)$ $(k = 1, 2, \ldots, n)$ は square-free である.
   - $f_k(x)$ は $f_{k+1}(x)$ の隣り合う実解の間で単調である.
   - $f_n(x)$ は定数関数である.
2. $f_{k+1}(x)$ の隣り合う実解の間で $f_k(x)$ の符号変化を調べることで $f_k(x)$ の実解があるか判定できる.

この多項式列は $f(x)$ を無平方化し, その導関数を無平方化して順次計算する.

また, このアルゴリズムのインプリメントの方針は次のとおりである.

- 扱うデータはすべて整数のみである.
- 各多項式の各実根は区間数の形で与える.
- 導関数を計算した後で, その多項式のコンテンツを取り除く.
- $f_{k+1}(x)$ の実根の位置における $f_k(x)$ の符号が確定しない場合には各実根の精度を自動的に上げることで判定する.

普通, 多項式の実解を求める手段としては Sturm の方法が有名であるが, Sturm の方法では多項式列を構成するときに係数が大きくなりすぎる場合がある. [2, pp.426–428] ここで提案した方法では多項式列は導関数から作られるものであり, 得られた導関数から毎回コンテンツを取り除くことで上の多項式列がすべて square-free のときは係数の膨張は高々 $2^{\text{もとの多項式の次数}}$ に抑えられるのでこの点では Sturm 列より有利であることがわかる.

また, Pari[1] には polroots という関数がある. この関数は複素解までを高速に解く. これとの比較についてはこの報告の後半部で述べる.

*hilano@ic.kanagawa-it.ac.jp

2 インプリメントしたシステムと今回の経緯

このアルゴリズムによるプログラムの開発は次のような経緯をたどった.

- 初期は Asir で開発した.
- 次に, pari の gp モードで開発をした.
- Pari に組み込まれている polroots の処理速度に対抗するため Pari の library mode(C から関数を呼ぶ)[1] で作成
- 今回の問題は当初 Asir 版で 30 分程度で解いた.
- その後, Pari の library mode で作成することにした.
- 以前の開発時期から Pari のバージョンアップがあり, この間のガーベッジコレクションの関数が変更になったのでその部分を中心にプログラムを修正した.
- これに基づいて 64 ビット版でも動くものが開発された.(FreeBSD/Opteron)

3 今回の問題とその解法の経緯について

3.1 今回の問題について

今回の問題の詳しい経緯については [3] を参照してほしい. そこで現れた多項式は次のような性質を持っていた.

- 486 次の多項式である.
- 多項式の長さは文字列で 500KB 程度 (平均 200 桁) であった.
- 根の存在範囲はそれほど大きくない.
- 途中の多項式列での実根の数は 160 個程度であった.
- 直前の多項式は 39 個である.
- 求める実根は 8 個である.

3.2 今回の改良点

- Pari のガーベッジコレクションで利用する関数を変更した.
- 根の精度を上げるときに多項式を $f(x/10)$ に置き換えたものを整数係数にしていたが、この多項式のコンテンツを除くようにした.(この問題では高次の係数が 10 のべきで割れるので効果的であった.)

この問題は Pari/Opteron 版で 40 秒程度で解くことができた.
なお, Pari の polroots で計算させたところ次のようであった.

- 1800 秒ぐらいの時間と 500MB 程度のメモリを使用してすべての根 (複素根を含む) が求まった.

- Pari が出した根はかなりな精度が出ていた. これは根を分離するために思った以上の高精度計算を必要とするためのようである. その結果, 計算時間とメモリーの使用量が多くなったと考えられる.
- 筆者の作成したプログラムが処理速度で勝ったはじめての例であった.

4 今後の課題

2005 年の春先に次のような問題が提供された.

- 次数 440 次で奇数次の項がなし.
- 多項式の長さは文字列で文字列にして 1.5MB 程度になった.
- 実根の数 6 個 うち 2 つは 3500 桁以上の整数部分を持つ. 残りの 4 個は絶対値が 1 に近い.

与えられたまま計算するとメモリーは 1GB 以上必要でで計算時間は 150 時間以上であった. この計算では 3000 桁以上のあたりでの多項式の評価を必要とするので, そのためにメモリーや計算時間が必要となった.

しかしながら, 根の逆数を求めるようにすると 1 分程度で計算が終了する. これは逆数を取ることで次のようなことがおこっているからと考えられる.

- 絶対値が 1 以下の解が 0 の近くと ± 1 の近くにあるだけなので, 解の分離が簡単にできた.(0 をはさんで対称の位置に絶対値が非常に近い解が存在するが, その二つの解は 0 により分離できた.)
- したがって, 3000 桁の位置での多項式の値を計算する必要がなくなった.

逆数の解が求まったとしても有効数字一桁を出すためにはやはり 3000 桁の多倍長計算を必要とする. どの解も 0 ではない限り有効数字を最低でも一桁与えるようにするためにはアルゴリズムの改良が必要である.

対策としては次のようなものが考えられる.

- 根をすべて求めるのではなく, 区間に分けて求めること.

  これについてはインプリメントはされているがすべての場合に正しくは動かない. 区間の端で求められた多項式列が 0 にならなければ正しく動作する.

- 指数的な区間の幅で根を求める.

  根の存在範囲が大きい場合に, 区間を分割して求めることにした場合, その分割方針として, たとえば, $10^{10}, 10^{20}, 10^{30}, \ldots$ のように分割する. この理由としてはたとえば 1000 桁あたりの根を求めるのに 1000 桁のまま行うことははじめから 1000 桁の精度を持って計算している可能性があるからである.

参 考 文 献

[1] Batut K., Belabas K., Bernardi D., Cohen H., Olivier M. User's Guide to the PARI library, http://pari.math.u-bordeaux.fr

[2] Knuth D. *The Art of Computer Programming* Vol.2,

[3] Kimura K., Hilano T., Yokoyama K. 関孝和の問題を解く, 数式処理 11(2005)pp.35–42

[4] 齋藤 友克, 竹島 卓, 平野 照比古, グレーブナー基底の計算実践編 Risa/Asir で解く, 東京大学出版会, 2003