

高精度計算による代数方程式の解法

平山 弘 今泉 喜則 佐藤 創太郎 菅 望

システムデザイン工学科

Numerical Method for Algebraic Equation by high-precision Calculation

Hiroshi HIRAYAMA, Yoshinori IMAIZUMI, Soutarou SATO and Nozomu SUGA

Abstract

It is very important to solve algebraic equations numerically. Although many methods for these equations have been developed by many resercher, it is very difficult to solve them precisely. In this paper, it is shown that these equations can be solved easily by high precision calcultion.

Key Words: high precision, multiple precision , C++ language, algebraic equation

1. はじめに

n 次の代数方程式 $f(x) = 0$ に対して、これまで多くの研究が行われている。最もよく知られた古い方法としてNewton法がある。この方法は、1 個の近似解 x_0 が知られているとき、 $f(x)$ を $x = x_0$ でTaylor展開し、一次の項までで近似する。

$$f(x) = f(x_0) + f'(x_0)(x - x_0) \quad (1.1)$$

この一次式を解くことによって、さらにより良い近似根を得る。

一般に $y = f(x)$ の逆関数 $y = g(x)$ の $x = y_0$ におけるTaylor展開は、

$$y = g(x) = g(y_0) + \frac{g'(y_0)}{1!}(x - y_0) + \frac{g''(y_0)}{2!}(x - y_0)^2 + \dots \quad (1.2)$$

ここで、

$$y_0 = f(x_0) \quad (1.3)$$

$$g'(x) = \frac{dy}{dx} = \frac{1}{\frac{dx}{dy}} = \frac{1}{f'(y)} \quad (1.4)$$

$$g''(x) = \frac{d^2y}{dx^2} = \frac{-f''(y)}{\{f'(y)\}^3} \quad (1.5)$$

$$g'''(x) = \frac{d^3y}{dx^3} = \frac{3\{f''(y)\}^2 - f'(y)f'''(y)}{\{f'(y)\}^5} \quad (1.6)$$

となる。これを使って、高次の公式を作成することができる。3 次の公式はHally法と呼ばれる。五十嵐[3]は、この種の任意次数の公式を提案している。

Durand-Kerner法やAberth法の全根を同時に求める反復公式と知られている。それぞれNewton法やHally法の反復公式の $f'(x)$ や $f''(x)$ の値を、近似根を使った式に置き換えることによって得られる公式である。

$f(x)$ を分子 M 次、分母 N 次の有理関数に展開する (Pade展開)。

$$f(x)Q(x) - P(x) = O(x^{M+N+1}) \quad (1.7)$$

Nourein[4]は、分子を一次式にし、分母を高次の式にすることによって、高次の公式を得ている。

以上の公式は、単根の場合、高次収束する公式であるが、重根等がある場合、1 次収束しかしないことが知られている。

櫻井等[5]は、 $f(x)/f'(x)$ をPade展開することによって、多重根でも高次収束する公式を提案している。これは、 $f(x)/f'(x)$ を計算するすると、重根は、分子分母の共通因子となり、約分されすべて単根となるためである。

代数方程式には上のように多くの解法が提案されているが、20~30次を越える代数方程式は解くことができない場合が多い。

代数方程式の問題点は、計算精度が足りないため代数方程式を精度良く解くことができない点にある。倍精度で計算し、倍精度で計算結果を得ら

れない問題が存在することが知られている。このことは次のような方程式を考察することによって容易にわかる。

$$(x-1.23)^6 = 0 \quad (1.8)$$

この方程式の解は、明らかに解 $x=1.23$ であるが、(1.8) 式を展開し、計算精度15桁の倍精度で計算するならば、 $x=1.231$ も解である。したがって、この方程式の解の計算精度は3桁と非常に小さくなる。このように、次数が高くなると代数方程式は、精度よく計算するのが、非常に困難になる。これを回避する方法はいろいろ提案されているが、確実にできる方法は計算精度を上げることである。

本論文では、この問題を解決するために、最大で約6000万桁の精度で計算できる可変長高精度計算ルーチン[2]を作成し、計算精度が十分にあるならば代数方程式は問題なく解けるかどうかを調べた。予測される問題点として高精度で計算したとき計算時間が非常に多くかかる問題があるが、それがどの程度になるのかを調べた。計算方法として、収束が証明されている平野法[1]を使った。

具体例として、ガウス型数値積分公式の分点や重みの計算を行った。ガウス型数値積分公式の分点や重みの計算は、代数方程式の重要な応用問題であり、精度が要求される問題である。通常代数方程式の問題と異なり、倍精度のガウス型数値積分公式を作るならば、最後の桁まで正しい値が要求されることである。

重み関数が積分区間で正であるならば、直交多項式を生成させ、直交多項式の漸化式を利用した計算方法を適用することによって、ある程度精度低下を防ぐことができる。しかし、次数が高くなるにつれて、この方法でも十分な精度で計算することができなくなる。

2 代数方程式の解法

代数方程式の根は、絶対値が小さい方から求めている。このように計算すると減次のとき、誤差が小さくなることが知られているので、この方法を使った。大きい絶対値を持つ根を計算する時には、係数を逆に並べた逆数を根にもつ方程式を使って、その方程式の減次を行って誤差が少なくなるようにしている。

平野法では、計算の途中で、方程式の一部を取り2項方程式として解き、解の絶対値が最も小さい場合を選び出す演算がある。この計算は、選び出す段階では、あまり精度を必要としないので、この段階では、精度を落として計算し、次の段階に進むとき精度を上げるようにしている。これによって、計算の速度を上げることができる。このようにすることによって、150桁で計算しているプ

ログラムで、根を選び出す段階を20桁で計算することによって、30次程度の方程式で約半分の時間で解くことができるようになった。選び出す段階の計算は通常の倍精度計算でも行うこともできるが、精度だけでなく指数部分の大きさも大きくなる場合が多いので、精度の低い多倍長数を利用した。このような最適化は代数方程式の次数が高くなれば、さらにその効果が現れると思われる。

プログラムはできるだけ適用範囲を広げるために、入力として高精度の複素数の係数を入力する形式にした。出力も高精度複素数で出力するように作成した。実際の計算では入力の実数である場合が多いので実数にすれば、高速化することができる可能性がある。

Newton法は局所的収束性については実用上十分な非線形方程式の解法であるが、大域的収束性は保証されず、初期値に依存する。しかし、代数方程式に限ると、Newton法を修正した方法である平野法(平野の改良Newton法)を使えば、大域的収束性が保証されている。

複素数を係数とする n 次多項式 $f(z)$ の零点を求めることを考える。

$$f(z) = a_0 z^n + a_1 z^{n-1} + \cdots + a_n \quad (2.1)$$

初期値 z_0 から始めて、次のように近似根を修正していくとする。

$$z_{m+1} = z_m + \Delta z_m \quad (2.2)$$

式(2.1)を近似根 z_m の周りで次のように展開する。

$$f(z_m + \xi) = c_0 + c_1 \xi + \cdots + c_n \xi^n \quad (2.3)$$

上式を0に近づける ξ を求めるわけである。 $|\xi|$ が十分小さいとして、上式を1次式で近似すると、

$$c_0 + c_1 \xi = 0, \quad \xi = -\frac{c_0}{c_1} \quad (2.4)$$

となり、通常のNewton法になる。平野法では次の n 個の二項近似を考える。

$$c_0 + c_1 \xi = 0, \quad \xi = -\frac{c_0}{c_1} \quad (2.5)$$

$$c_0 + c_2 \xi^2 = 0, \quad \xi = \left(-\frac{c_0}{c_2} \right)^{\frac{1}{2}} \quad (2.6)$$

...

$$c_0 + c_n \xi^n = 0, \quad \xi = \left(-\frac{c_0}{c_n} \right)^{\frac{1}{n}} \quad (2.7)$$

この中で、絶対値の小さい ξ を修正量とする。

この後、関数値を単調に減少させるために減速という操作を行う。

数値計算では次のような流れとなる。なお、減

速のパラメータ β , λ の範囲は, $0 < \beta < 1$, $\lambda > 1$ である。

1. 式(2.3)のべき級数の係数 c_k ($k = 0, 1, \dots, n$) を組立除法で計算する。
2. $\xi_k = \left(-\mu \frac{c_0}{c_k} \right)^{\frac{1}{k}}$ ($k = 1, \dots, n$)
3. $|\xi_k|$ が最小の k ($1 \leq k \leq n$) を l とする。
4. $|f(z_m + \xi_l)| \leq (1 - (1 - \beta)\mu)|c_0|$ であれば, $\Delta z_m = \xi_l$ とする。

そうでなければ, $\mu = \frac{\mu}{\lambda}$ として, 2. に戻る。

この方法をくり返し, 近似根における関数値が十分小さくなったら, そのときの近似根を根として終える。

1 つの根 (α) が求まれば, 減次

$$f(z) = \frac{f(z)}{z - \alpha} \text{ して次の根を求める。}$$

3 数値例

ここで扱う数値例は, すべて Pentium III 933MHz 上で実行した。使ったコンパイラは Boland C++ 5.0 である。

例 1. ウイルキンソンの問題

次の方程式を展開しその解を求めよ。ただし, 展開された係数は倍精度で計算与えるものとする。

$$\prod_{k=1}^{20} (x - k) = 0 \quad (3.1)$$

この方程式の解は 1 から 20 までの整数値になるが, 係数が 15 桁に丸められるため, 誤差が生じる。以下にこの数値計算結果を示す。この計算は 10 進数 200 桁で計算したものである。計算時間は 6.1 秒であった。この時の収束条件は, 解を代入したとき, その値が 10^{-100} より小さくなった時, 収束したものとして反復を止めるようになっている。計算時間には途中の若干の出力を含むがこれをなくしても計算時間にあまり影響を与えないと思われる。それを 40 桁の精度で解いた結果を示す。

```

1.0000000000 0000131530 1639459899 0653696923 5540749325 186004243
2.0000000000 0095964407 6155535492 3659583005 4919622853 096723649
2.9999999998 6639955134 7144660451 8753448478 5876465433 963068359
4.0000000049 5944066373 3101634385 7813850215 6470858325 159507680
4.9999999147 3414288695 4572990327 8962470432 5088334542 601351646
6.0000008457 1660734935 4838430005 7722119443 6222156442 831891372
6.9999945554 4845213517 7549160135 6146971485 3213662042 192511247
8.0000244325 6893858785 5917444778 0522585878 3022646000 360387931
8.9999200118 6834800982 1276735028 3421231619 9643766708 431451401
10.000196964 9053688150 1099647779 6229386944 8298458678 269591815
10.9996284302 4064360444 9327051087 0270919299 3863246829 97721883
12.0005437436 3591164235 9616043027 7517281041 6066635107 55288615
12.9993807345 5789735837 6764317696 5285837223 0539001376 61606450
14.0005479886 7380047134 2555387245 8414018068 6647740314 95684832
14.9996265821 7054832524 3412151948 4218553659 7223533994 94252559
16.0001920830 3847318082 7245855403 5938780226 3705164417 95333603
16.9999277346 1773180983 7468382172 7959276626 7222203171 90444338
18.0000187517 0604149346 2942089854 9204964149 1786952274 52724469
18.9999969977 4389137612 9607168417 9953657284 7218908638 02090595
20.0000002235 4640177933 7869024844 1286885489 2743765417 02971088

```

係数が変化すると, どの程度解が変化するかを調べた。その結果を図 1 に示した。縦軸は係数の次数, 横軸は 1~20 までの根である。すなわち,

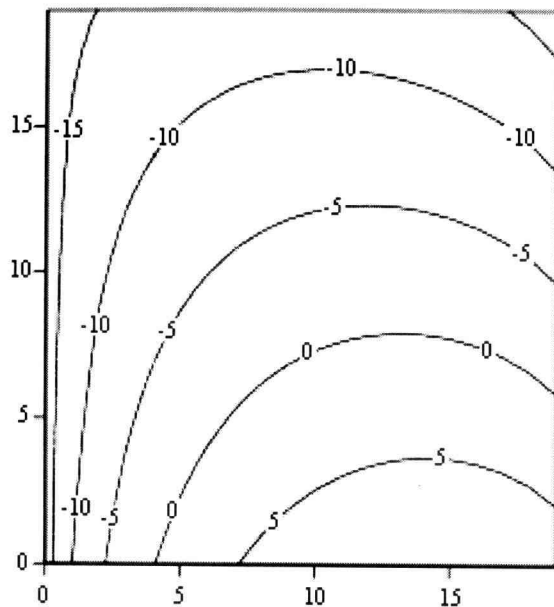
$$\log_{10} \left| \frac{\partial r_i}{\partial c_j} \right|$$

をプロットしたものである。 i は横軸, j は縦軸

である。 c_i は $i-1$ 次の係数で, r_j は j を値にもつ根である。15 の値を持つ根は, 低次の係数の変化の影響をよく受けることを示している。係数が少し変化すると 10^5 倍以上の変化を受けることを示している。実際の測定では, 10^9 倍程度の影響を受けた。倍精度で 15 桁程度の精度しかない計算機では, 最後の 1 桁が 1 ずれると, 解はその 10^9 倍ずれるこ

とを示している。これは、典型的な例であるが、代数方程式は係数が少しずれただけで解が大きく影響することがわかる。

図1では、特定の方程式(3.1)の係数と根の変化の計算であるが、一般に高次の方程式では係数の変化が根の大きな変化となる。これが高次代数方程式の解法の困難さを表している。



b

図1 係数と根の影響

分点座標

1 7.0539889691 9887533666 8900458421 5095869360 6298353100 6537693247 574441750e-002
 2 3.7212681800 1611443794 2413887611 4663667402 8210156184 1234158265 441997834e-001
 3 9.1658210248 3273564667 7162770741 8318720560 4198042966 2805750011 627793207e-001
 (途中省略)

18 4.7619994047 3465021399 4162715285 1121113143 9703425749 2874479289 596122956e+001
 19 5.5810795750 0638988907 5077344449 7235628385 3112483335 3939332085 467291837e+001
 20 6.6524416525 6157538186 4031879146 0665979634 9130056970 8145901753 669229260e+001

Weight Factors = w_i

1 1.6874680185 1113862149 2238996894 8079260392 7970557464 5163680349 929542070e-001
 2 2.9125436200 6068281716 7953238122 2649010672 6819466688 9449708476 233443710e-001
 3 2.6668610286 7001288549 5208689978 8224113897 4898758070 2294339990 673771918e-001
 (途中省略)

18 1.5395221405 8234355346 3833196674 0244446312 9686997276 3889628013 021223726e-020
 19 5.2864427255 6915782880 2735876827 9932945003 7262751298 0720749476 004613717e-024
 20 1.6564566124 9902329590 7819085291 0559845009 2072076320 2192396558 672987564e-028

例3. 複素数を係数とする方程式

$$\sum_{k=0}^{20} (200 - k^2)x^k = 0 \quad (3.3)$$

この方程式を計算精度128桁で解くために必要な

例2. ガウスラゲールの積分公式

$$\int_0^\infty e^{-x} f(x) dx \approx \sum_{i=1}^n \omega_i f(x_i) \quad (3.2)$$

この場合、モーメントは

$$M_j = \int_a^b W(x) x^j dx = j! \quad j = 0, 1, \dots, k \quad (3.2)$$

である。 $n = 20$ のとき、次のように精度80桁の数値を簡単に求められる。解はすべて実数であるが、絶対値が小さい解と大きな解が共存する方程式である。この方程式は、20次と比較的低次の方程式なので、数値のアンダーフローやオーバーフローは起きないが100次程度の高次方程式になるとアンダーフローやオーバーフローが簡単に起こる。特に重みはアンダーフローが生じ、通常の計算機で使われる浮動小数点数の範囲外となり、コンパイルも出来なくなる。

ガウス型積分公式は、高精度を特徴とする積分公式なので、最後の桁まで正しい数値であることが必要である。これは、高精度計算の代数方程式の解法プログラムがあって初めて可能な計算である。

計算時間は、6.8秒で、根がすべて実数で次数が同じ場合である例1とほぼ同じであった。高精度計算では、実際に虚数部分がなければ、実数の計算とほぼ同じ時間になるので、このような結果になったものと思われる。この計算結果の一部を以下

に示す。

```

0      -1.00455876016513091297970346536572548551448544942437421739328
        0.0000000000000000000000000000000000000000000000000000000e+00
1      2.2885629531261511492639710510988548879018347743985550474e-01
        9.7812486125463125617542766684123189393159654262779391974e-01
        6.7497617971139290879231777396033878312308721699446866096e-01
        (途中省略)
7      9.1786644276553456474720468668243284538430615911814735680e-01
        -2.643661268882090592753536701078623025800751204756137130e-116
8      -9.5973544341075309009526938295338161041845812661968172921e-01
        -2.9672492146121844029402467170195773015774512151109774361e-01
        (途中省略)
18     5.0880106572827259897319812702398792278632454120201208883e-01
        8.6613689377510903168524121462345257660369707341500492058e-01
19     -3.6424193529362647460646898380370200633829729489947648759e-01
        -9.3619219752861626728664286513431945279268402424794660412e-01

```

左側の数値が解の番号で、番号の右側が実数部でその下が虚数部分である。この問題は2個の実数解と18個の複素数解が得られる。実数解の1つは、計算結果としては複素数解であり、虚数部が -2.64×10^{-116} であった。この解の虚数部は計算精度の限界程度の小さな数値なので容易に実数解と判断されるが、0と容易に判断できないような場合、どのようにしてこの解を実数解と判断すべきか等の問題が起こる。

4 まとめ

高次方程式の解くための問題点は、精度不足になり計算が破綻することである。これを解決するために可変高精度ルーチンを使って解いてみた。高精度計算ルーチン使うことによって、より広い範囲の方程式を解くことができるようになった。計算時間については、実際の計算時間を示したが、この時間が実用的かどうかは、ユーザーに判断を任せたいと思う。

しかしながら、当然予測できたことであるが、通常の倍精度などの低精度計算で起こる収束しない等の現象が、数百桁で計算しても100次程度の方程式に対して計算すると同じような現象起こり、計算が破綻することがある。無制限に計算精度を上げることは、非常に多くの資源を使うので、問題が与えられたとき、何桁の精度で計算する必要があるかを決定することが必要がある。

5 参考文献

- [1] 室田一雄, " 平野の変形Newton法の大域的収束性", 情報処理学会論文誌, Vol. 21, pp. 469-474, 1980
- [2] 平山 弘, " C++言語による高精度計算パッケージの開発", 日本応用数学会, Vol. 5, No.3, pp. 123-134, 1995
- [3] 五十嵐 正夫, "代数方程式の大域的解法の初期値に対する一注意", 情報処理学会論文誌, Vol. 30, No.11, pp. 1526-1528, 1989
- [4] Nourein, M., "Root Determination by Use of Pade Approximants", BIT, Vol.16, pp.291-297, 1977
- [5] 櫻井、鳥居、杉浦, "Pade近似による代数方程式の反復解法", 情報処理学会論文誌, Vol.31, No.4, pp.517-522, 1990