社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－ARC－172 (34)
2007－HPC－109 (34)
2007／3／2

古典的精度推定法に基づいた Gauss型積分公式の分点計算について

幸谷智紀*

Gauss 型積分公式は直交多項式によって規定されるものであり，その分点は直交多項式の零点となる。この分点の精度は直接定積分の精度に影響を与えるため，高精度である必要がある。この零点を求める方法として，多倍長計算と Newton 法に基づいた山下の方法と，実対称行列の固有値問題に基づいた Golub& Welsch の方法の二つが 1960 年代に提案されている。本稿ではこの２つの方法に，いわゆる「古典的精度推定法」を適用し，ユーザが指定した精度を持つ分点を求めることができることを示す。

On the Computations of Abscissas of Gauss Quadrature Rules based on Classsical Precision Estimation

Tomonori Kouya*

Gauss Quadrature rules are defined by the corresponding orthogonal polynomials. so its abscissas are zero points of these polynomials. The zero points must be accurate because the accuracy of these zero points effects on one of the definite integrals computed by the Gauss quadrature rules. Two means to obtain the abscissas have been proposed in 1960's. One is proposed by Yamashita, based on Newton method and multiple precision arithmetic. The other is proposed by Golub and Welsch, based on eigenvalue problem of real symmetric matrix. In this paper, we apply so-called "classical precision estimation" to the two means, and demonstrate that our proposed method can obtain the abscissas with user-given precision.

1. 初めに

現在の数値計算の多くは有限桁の浮動小数点数演算と近似理論に基づいたものであり，それ故に数値計算によって得られた結果は丸め誤差や理論誤差 (打ち切り誤差) の影響を免れ得ない。特に，数値計算がコンピュータ上で高速に行われるようになると，途中結果を逐一チェックすることが難しくなった。そこで，数値結果に含まれる誤差の大きさを事後的に判断する手法がいくつか考案されている。特に近年では精度保証を行うための有用な技術や理論が開発されているが，これも誤差を判断するための手法の一つといえる。

それとは別に，かなり昔より「数値計算の常識」として浸透している誤差判定の手法がある。それがうまくまとめられている邦書としては伊理 [6] の第 4 章「たった一回の計算なんて」を挙げておく。

それによれば，数値計算結果に含まれる誤差の判定は，条件を変えて複数回の計算を行い，その結果を比較することによって可能となる，とある。具体的に言うと，理論誤差の判定は，例えば常微分方程式の初期値問題の数値計算の場合は，刻みを変えて計算することで誤差の order を知ることができるし，計算桁数を変えて計算することで，長い桁数で計算した結果を元に，短い桁数で計算した結果に含まれる丸め誤差の大きさを判定することができるようになる。このような Try & Error を繰り返して誤差を推定する手法を，本稿では「古典的精度推定法」と呼ぶことにする。

このような古典的精度推定法に基づく論証は，多数の数値計算に関する論文に見ることができる。つまり，執筆者も査読者もこの手法に対しては疑義を持たず，信頼性の高い誤差判定法として常識化していると言う事ができる。

そこで我々はこの手法を，アルゴリズムとして自動化した上で，ユーザが指定した精度を持つ数値結果を得る目的で使用できることを示す。本稿ではその適用例として，Gauss 型積分法の分点計算に適用した結果について報告する。これは三項漸化式で帰納的に表現される直交多項式の零点計算に相当するものであるが，1960 年代には 2 つの方式で計算することが提案されている。一つは代数方程式の零点を Newton 法で計算するもので，日本では山下眞一郎が ALGOL プログラムと共に，分点と重みの数表を掲載した論文を執筆している。もう一つは，G.H.Golub と J.H.Welsch が提案したもので，三項漸化式を対称な実 3 重対角行列として表現し，その固有値問題として解く手法である。結論から言うと，直交多項式の分点数が上がるにつれて分点の近接度が増すため，前者は多倍長計算を用いなければ Newton 法の収束すらおぼつかなくなるのに対し，後者は固有値問題としては相当高い次数でも良条件であるため，使用した浮動小数点数の桁数に近い精度を持つ分点が得

*静岡理工科大学
*Shizuoka Institute of Science and Technology

られるので，前者に比べて数値的に安定した手法であると言える。しかし多倍長計算環境下であれば，前者でも強引に解く事は可能であり，悪条件問題でも本稿で述べる手法が適用可能であることを示す実例としては望ましい。

そこで，本稿ではこの2つの手法によって，ユーザが指定する精度を持つ Gauss-Lengendre 積分公式の分点計算を行った結果について報告する。

2. 古典的誤差推定法に基づく精度推定法

本稿では，例えば伊理 [6] に述べられているように，

理論 (打ち切り) 誤差の推定 ・・・ 丸め誤差より優越している地点の誤差を理論誤差の推定値 $T(x)$ として使用

丸め誤差の推定 ・・・ 同じアルゴリズムを実行し，長い桁数 L で計算した結果 x^L を真値として用い，それより短い桁数 S による結果 x^S に含まれる丸め誤差を

$$R(x^S) = |x^L - x^S| \tag{1}$$

として $R(x^S)$ を推定値とする

という，「数値計算の常識」となっている方法を「古典的誤差推定法」と呼ぶことにする。本稿ではこれに基づいて精度の推定を行う。

このうち，丸め誤差の推定法は問題やアルゴリズムによらず使用可能であるが，理論誤差の推定法はそれに応じたものが必要となる。今回取り上げるのは代数方程式の零点を求めるための Newton 法と固有値問題を解くための QR 分解法であるので，どちらも近似値が数列 $x_0, x_1, ..., x_n, ...$ $(x_k \in \mathbb{R})$ という形で得られるタイプのアルゴリズムになっている。そこで，

- 各近似値 x_i の丸め誤差の評価値 $R(x_i)$ が得られている
- 数列は単調に収束している

と仮定できれば，各近似値 x_k に含まれる理論誤差の**評価値の候補** $T(x_k)$ を

$$T(x_k) = |x_{k+1} - x_k| \tag{2}$$

とし，

$$R(x_k) < T(x_k)$$

であれば，これを理論誤差として，x_k に含まれる誤差全体の評価値 $E(x_k)$ としても使用する。もしこれが成立しなければ，理論誤差は丸め誤差より小さいと判断し，$E(x_k)$ としては $R(x_k)$ を使用する。

このようにして x_k に含まれる誤差の評価値 $E(x_k)$ が得られたら，これを x_k の絶対誤差として扱い，これに基づいて x_k の精度の判断を行う。

重要なことは，これはあくまで精度の推定であって，解法の収束判定とは別の次元の話であるということである。従って，収束判定の基準はアルゴリズムごとに別途考える必要がある。

3. Gauss 型積分公式の分点計算

Gauss 型積分公式の分点 $\alpha_1, \alpha_2, ..., \alpha_N$ は，三項漸化式 (3)

$$p_j(x) = (a_j x + b_j)p_{j-1}(x) - c_j p_{j-2}(x) \ (j = 1, 2, ..., N) \tag{3}$$

に基づいて定義される N 次直交多項式 $p_N(x)$ の零点 $p_N(\alpha_i) = 0 (i = 1, 2, ..., N)$ として表現できる。ここで，$a_j, b_j, c_j \in \mathbb{R}$ は直交多項式ごとに定まる定数であり，Legendre 多項式の場合は $p_{-1}(x) = 0, p_0(x) = 1, a_j = (2j-1)/j, b_j = 0, c_j = (j-1)/j$ である。また，分点に対応する重み $w_i (i = 1, 2, ..., N)$ は，$p_j(x)$ の最高次の係数を μ_j, $\lambda_j = \int_a^b (p_j(x))^2 dx$ とする時，

$$w_i = \left(\frac{\mu_{N-1}}{\lambda_{N-1}\mu_N} p_{N-1}(\alpha_i) p'_N(\alpha_i) \right)^{-1} \tag{4}$$

となる。これによって，Gauss 型の数値積分は，重み関数を $w(x)$ とすると

$$\int_a^b w(x) f(x) dx \approx \sum_{i=0}^{N} w_i f(\alpha_i)$$

と計算される。Gauss-Legendre 積分公式の場合は，$[a, b] = [-1, 1]$ である。

3.1 山下の方法

山下眞一郎 [10] は，多倍長計算が可能な ALGOL 処理系を用いて Newton 法によって Legendre 多項式の零点を求め，重みを (4) に基づいた形で求めている。この時，初期値は直交多項式ごとに適切なものを指定し，減次は行なっていない。このようにして，山下は 2 点から 35 点までの Gauss-Legendre 積分公式の分点と重みを 10 進 20 桁の数表として提示している。使用した桁数は 30 点公式で 10 進 45 桁，Newton 法は 4〜5 回で収束していると山下は述べている。

この方法の欠点は，分点数 = 直交多項式の次数が増えるにつれて，零点の密集度が増し，極めて悪条件の代数方程式問題となることである。数表作成には多倍長計算が必要であるが，それ以上に，山下の提示した初期値 (10 進 3 桁程度は正しい) を与えても，倍精度計算では 10 次程度でも Newton 法が収束しないという状況になる。また，収束したとしても精度は極端に落ちてしまうことになる。しかし，全てを実数計算で行えることと，並列化がきわめて容易であるという利点もある。

今回は後述する Golub& Welsch の方法による IEEE754 倍精度計算の結果を初期値とし，Newton

法の計算は全て多倍長計算で行った。その際，最大反復回数は100回とし，この回数を越える反復を必要とする場合は桁数が足りないと判断，自動的に桁数を増やして再計算するようにしてある。

精度推定は次のようにして行っている。なおこの手順は全てプログラムによって自動的に行われるようになっている。

まず，ユーザが指定した精度桁数(10進)を U である時，精度桁数の増分量のデフォルト値が D であれば，増分量 C を

$$C = \max(D, U/10) \tag{5}$$

とする。これを元に，実際に計算する桁数 S を

$$S = U + C \tag{6}$$

とし，丸め誤差を評価するために必要な精度桁数 L を

$$L = S + C \tag{7}$$

として，S より C 桁だけ余分に桁数を取るようにする。更に，収束判定に必要な相対許容度 ε_R と絶対許容度 ε_A を

$$\varepsilon_R = 10^{-U} \tag{8}$$

$$\varepsilon_A = 10^{-2U} \tag{9}$$

とする。

その上で，前述したようにIEEE754倍精度計算で求めた初期値 x_0 から出発し，S 桁計算と L 桁計算によるNewton法を実行し，近似値 x_k^S と x_k^L を求める。従って(1)式から $R(x_k^S) = |x_k^L - x_k^S|$ が丸め誤差の評価値として得られる。また，理論誤差の評価値 $T(x_k^S)$ は(2)式に基づき，$T(x_k^S) = |x_{k+1}^S - x_k^S|$ として求める。

収束は

$$|x_k^S| \le \varepsilon |x_k^S - x_{k-1}^S| + \varepsilon_A$$

を満足した時点とする。この時の反復回数を k_s とする。ここでこの時点における理論誤差の評価値 $T(x_{k_s}^S)$ を得るために，更にもう一度だけ反復を行い，$x_{k_s}^S$ の絶対誤差の評価値 $E(x_{k_s}^S) = \max(R(x_{k_s}^S), T(x_{k_s}^S))$ を確定する。

もしこの時点で $E(x_{k_s}^S) < 10^{-U}$ を満足していれば，ユーザ指定の精度を持つ近似値として受け入れ，そうでなければさらに C 桁追加してもう一度計算を行う。

これとは別に，もし指定された最大反復回数に達しても収束しない場合は，C を2倍して再計算を行うようにした。

3.2 Golub&Welschの方法

Golubと Welsch[2]は，1次から N 次までの(3)を

$$x\begin{bmatrix} p_0(x) \\ p_1(x) \\ \vdots \\ p_{N-2}(x) \\ p_{N-1}(x) \end{bmatrix} = \begin{bmatrix} -\frac{b_1}{a_1} & \frac{1}{a_1} & & & \\ \frac{c_2}{a_2} & -\frac{b_2}{a_2} & \frac{1}{a_2} & & \\ & \ddots & \ddots & \ddots & \\ & & \frac{c_{N-1}}{a_{N-1}} & -\frac{b_{N-1}}{a_{N-1}} & \frac{1}{a_{N-1}} \\ & & & \frac{c_N}{a_N} & -\frac{b_N}{a_N} \end{bmatrix} \times \begin{bmatrix} p_0(x) \\ p_1(x) \\ \vdots \\ p_{N-2}(x) \\ p_{N-1}(x) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ \frac{1}{a_N} p_N(x) \end{bmatrix} \tag{10}$$

と表現し，これを改めて

$$x\mathbf{p}(x) = T\mathbf{p}(x) + \frac{1}{a_N} p_N(x)\mathbf{e}_N \quad (\mathbf{e}_i\text{は単位ベクトル})$$

と置き，分点 $x = \alpha_i$ においては

$$T\mathbf{p}(\alpha_i) = \alpha_i \mathbf{p}(\alpha_i) \tag{11}$$

となることを利用し，(11)を固有値問題のアルゴリズムを使用して解き，分点(固有値)を求めることを提案している。

現実の分点は全て実数であるが，三重対角行列 T は非対称行列になることがあるので，

$$d_{i+1} = \sqrt{\frac{a_{i+1}}{a_i c_{i+1}}}\, d_i$$

を対角成分とする対角行列 D を用いて

$$J = DTD^{-1}$$

$$= \begin{bmatrix} -\frac{b_1}{a_1} & \sqrt{\frac{c_2}{a_1 a_2}} & & & \\ \sqrt{\frac{c_2}{a_1 a_2}} & -\frac{b_2}{a_2} & \sqrt{\frac{c_3}{a_2 a_1}} & & \\ & \ddots & \ddots & \ddots & \\ & & \sqrt{\frac{c_{N-1}}{a_{N-2} a_{N-1}}} & -\frac{b_{N-1}}{a_{N-1}} & \sqrt{\frac{c_N}{a_{N-1} a_N}} \\ & & & \sqrt{\frac{c_N}{a_{N-1} a_N}} & -\frac{b_N}{a_N} \end{bmatrix} \tag{12}$$

と対称化し，この J を使用することも同時に提案している。ここで，重みはユークリッドノルムによって正規化された J の α_i に対応する固有ベクトル $\mathbf{q}_i$ の第一成分 $q_1^{(i)}$ を用いて

$$w_i = (q_1^{(i)})^2 \int_a^b w(x)dx \tag{13}$$

を計算することによって得られる。

今回はこの J に対して Wilkinson シフトを用いた実対称行列用の QR 分解法 [3][4][5] を実行して分点を求めている。また，重みは分点 α_i を求めた後，連立一次方程式 $(J-\alpha_i I)\mathbf{q}_i = 0$ を解いて正規化し，(13) によって求めている。

本解法の利点は，代数方程式のような近接根が引き起こす問題が発生せず，固有値問題としては極めて良条件であるため，ほぼ計算桁数と同程度の精度を持つ分点を求めることが出来る点にある。しかし，QR 分解の並列化が難しく，また記憶領域も三重対角行列を記憶しておくため，山下の方法に比べると多く必要になる。

精度推定は，前述した山下の方法によるものと同様に行う。ユーザが指定した精度桁を U とした時，C を (5)，S を (6)，L を (7)，ε_R と ε_A を (8) と (9) と指定し，丸め誤差の評価のために S 桁計算で求めた行列 J^S と，L 桁計算で求めた行列 J^L それぞれに Wilkinson シフトつき QR 分解法を適用する。丸め誤差の評価値及び理論誤差の評価値はそれぞれ特定の対角成分についてのみ同様に行う。J^S の副対角成分の値を見て収束を判断し，この時点でもう一度反復を行って停止し，精度の推定を行って問題なければ減次して次の固有値の計算を行っている。

4. 数値実験

数値実験を行った環境は以下の通りである。

使用ハードウェア AMD Athlon64 X2 3800+, Fedora Core 4 x86_64

使用ソフトウェア gcc-4.1.1, GMP 4.2.1[1], MPFR 2.2.1[9], BNCpack 0.6c[7]

Newton 法及び QR 分解法は，BNCpack が提供している，MPFR/GMP による多倍長演算をベースとしたデータ型及び基本線型計算を用いて実装した。MPFR は変数ごとに仮数部の桁数を指定できるため，BNCpack のデータ型も全て変数ごとに指定できるようになっている。また，BNCpack で提供している多倍長計算関数は，内部で実行される全ての計算を返り値の桁数で実行するように実装されており，丸め誤差の評価値を容易に得ることができる。

得られた分点の精度を検証するため，次の定積分を Gauss 型積分公式で計算し，その相対誤差を計測した。

$$\int_0^{\pi/2} \cos x \, dx = 1$$

丸め誤差の影響を排除するため，全て 10 進 2000 桁計算を行っている。この結果を表 1，2 に示す。なお，重み w_i は (4) で計算できるが，次数が増えると直交多項式の値が桁落ちにより不正確になる。そのため，山下の方法，Golub & Welsch の方法のどちらで求めた分点でも，重みは (13) に基づいて計算してある。

表 1: Gauss 型積分による検証結果 ($\log_{10}$(相対誤差))

	山下の方法		
分点数	50 桁	100 桁	1000 桁
16	−49	−49	−49
32	−51	−101	−115
64	−52	−101	−268
128	−51	−100	−611
256	−51	−101	−1002
512	−51	−101	−1003
1024	−51	−101	−1001

表 2: Gauss 型積分による検証結果 ($\log_{10}$(相対誤差))

	Golub & Welsch		
分点数	50 桁	100 桁	1000 桁
16	−49	−49	−49
32	−52	−102	−115
64	−52	−102	−268
128	−53	−103	−611
256	−53	−102	−1002
512	−52	−102	−1003
1024	−52	−102	−1003

表 1 及び表 2 の結果より，分点及び重みは全て要求桁数以上の精度を保っていることが分かる。

最後に，1024 次の場合における山下の方法による理論誤差及び丸め誤差の評価値のグラフを図 1 に示す。

ユーザの要求桁数が低いにもかかわらず，直交多項式の計算における桁落ちが激しい (400 桁近い) ため，$U = 50$ の場合は $S \approx 695$，$U = 100$ の場合は $S \approx 745$ の精度が必要となる。しかし，要求桁数が上がるにつれて U と S の差は縮まっていき，$U = 1000$ の時は $S \approx 1701$，$U = 5000$ の時は $S = U + C$ で収束している。

なお，Golub & Welsch の方法の場合は，全ての精度桁において，丸め誤差の評価値が ε_R とほぼ一致し，理論誤差の評価値は 0 となった。

5. 結論と今後の課題

以上の検証結果より，我々の提案する古典的誤差推定法に基づいた Gauss 型積分公式の計算は，悪条件問題となる山下の方法でも，良条件である Golub & Welsch の方法でも有効に働くことが確認できた。この結果を踏まえ，次のような展開を行っていきたい。

5.1 古典的誤差推定法の適用範囲の拡大

今回は数値積分によって得られた結果を検証することができる問題を選択したが，我々の提案する手法はどのような問題にも，理論誤差の評価が適切に

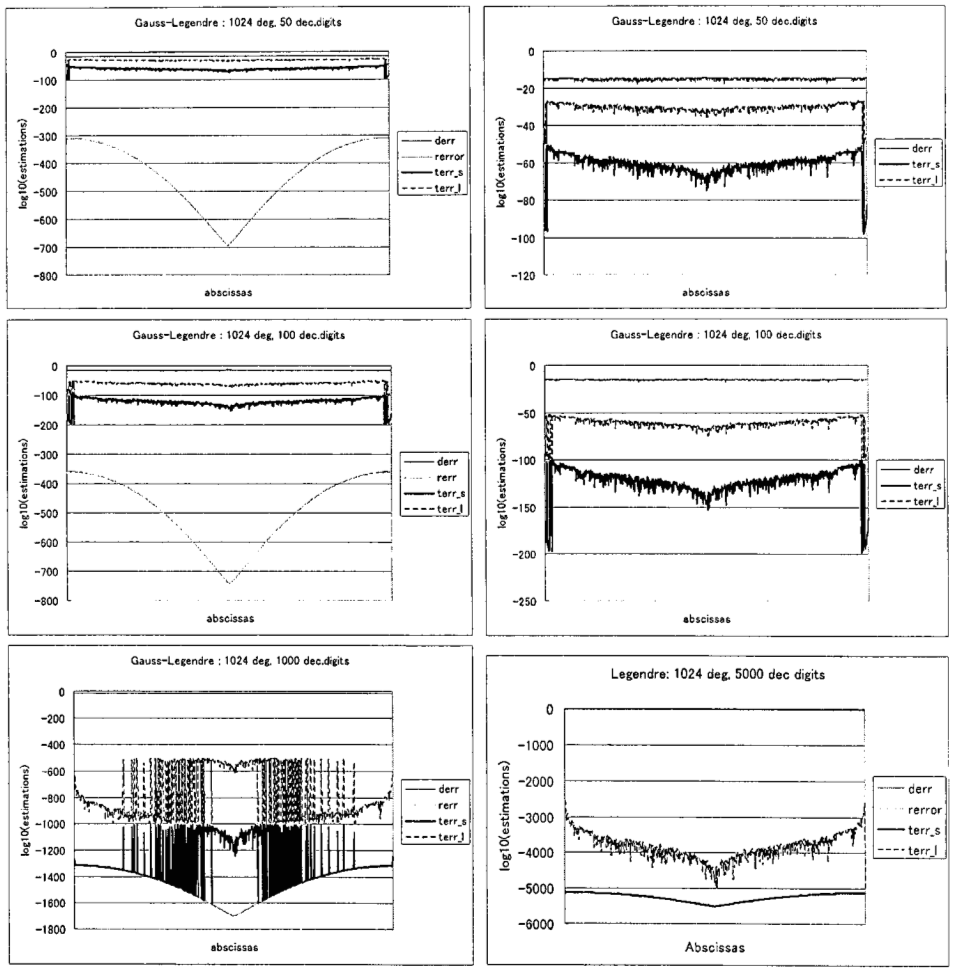

図 1: 山下の方法による分点計算: 1024 次 10 進 50 桁 (上), 100 桁 (中), 1000 桁 (左下), 5000 桁 (右下)

なされていれば適用可能であると予想できる。よって，零点計算や固有値計算以外のアルゴリズムにも逐次適用し，数値的な検証を行っていきたい。

また，丸め誤差の評価法については，今回はよりシャープな評価が得られる安全な方法を選択したが，同じ桁数で異なる丸めモードで計算した結果を比較して丸め誤差を評価する方法もある [8]。これとの比較検討も行っていきたい。

5.2 任意精度 Gauss 型積分公式の計算プログラムの開発

今回は Legendre 多項式の場合のみを計算したが，同様の方法により，他の直交多項式の場合も要求桁数の分点と重みを求めることは可能であると思われるので，複数の直交多項式に対応した Gauss 型積分公式の導出プログラムを開発していきたい。

その際には，より高速かつ重みについても分点同様の誤差推定が可能になるような工夫を行う必要がある。高速化については，特に丸め誤差推定に必要な計算の並列化が課題となろう。

参考文献

[1] Swox AB. GNU MP. `http://swox.com/gmp/`.

[2] G.H.Golub and J.H.Welsch. Calculation of gauss quadrature rules. *Mathematics of Computations*, Vol. 23, pp. 221–230, 1969.

[3] G.H.Golub and C.F.van Loan. *Matrix Computations (3rd ed.)*. Johns Hopkins University Press, 1996.

[4] G.W.Stewart. *Matrix Algorithms Volume I: Basic Decompositions*. SIAM, 1998.

[5] G.W.Stewart. *Matrix Algorithms Volume II: Eigensystems*. SIAM, 2001.

[6] 伊理正夫, 藤野和建. 数値計算の常識. 共立出版, 1985.

[7] Tomonori Kouya. BNCpack. `http://na-inet.jp/na/bnc/`.

[8] 幸谷智紀, 永坂秀子. IEEE754 規格を利用した丸め誤差の測定法について. 日本応用数理学会論文誌, Vol. 7, No. 1, pp. 79 – 89, 1997.

[9] MPFR Project. The MPFR library. `http://www.mpfr.org/`.

[10] 山下眞一郎. Gauss の数値積分公式の分点と重率の決定. 情報処理, Vol. 5, pp. 206–215, 1964.