Vol.2021-HPC-181 No.5
2021/9/27

Lagrange補間の係数計算のある簡単な数値的トリックについて

村上 弘[1,a)]

概要：Lagrange 補間式の係数を求める際に，浮動小数点数の性質を用いた以下のような近似計算法が可能である．使用する補間の分点 x_j，j=1, 2, . . . , n は互いに良く分離しているとする．そのとき関数 $f(x)$ に対する Lagrange の補間多項式は $p(x) = \sum_{j=1}^{n} w_j(\ell(x)/(x-x_j))f_j$ で与えられる．ここで $\ell(x) \equiv \prod_{k=1}^{n}(x-x_k)$，$f_j \equiv f(x_j)$ である．そうして $1/w_j = \ell'(x_j)$ であるが，この値を分点 x_j の値を極めて僅かにずらした点 y_j での値 $\ell'(y_i)$ に置き換える．すると，$\ell'(y_j) \approx \ell(y_j)/(y_j - x_j)$ である．この右辺の分母は分子に因子として含まれている．よって近似の仮定の下では，$s_j = \ell(y_j) = \prod_{k=1}^{n}(y_j - x_k)$ を求めて $w_j = (y_j - x_j)/s_j$ とすれば良い．この方法の利点は，係数 w_j を求める計算の中で，添字に依存する条件分岐やあるいは添字に依存したループ分割が回避できることである．そうして，補間多項式の $x = z$ に於ける値 $p(z)$ は，z がどの補間点にも一致や極端な近接がなければ，$p(z) = \ell(z)\sum_{j=1}^{n} w_j/(z - x_j)f_j$ として計算できる．

キーワード：ラグランジュ補間，浮動小数点数計算

1. はじめに

Lagrange 補間式の係数を求める際に，浮動小数点数の性質を用いた以下のような近似計算法が可能である．

いま指定された（相互に一致や極端な近接も無い）補間の分点を x_k，k=1, 2, . . . , n とし，それらを零点として持つ多項式 $\ell(x)$ を以下の式 (1) とする．

$$\ell(x) \equiv \prod_{k=1}^{n}(x - x_k)\,. \tag{1}$$

そのとき関数 $f(x)$ に対する Lagrange の補間多項式 $p(x)$ は，$f_j \equiv f(x_j)$ とするとき，以下の式 (2) で与えられる．

$$p(x) \equiv \sum_{j=1}^{n} w_j \frac{\ell(x)}{x - x_j} f_j\,, \text{ただし } w_j = 1/\ell'(x_j)\,. \tag{2}$$

1.1 通常の方法による係数 w_j の計算

通常の方法では $\ell'(x_j)$ の値を式 (3) により計算する．

$$\ell'(x_j) = \prod_{k\,(k\neq j)} (x_j - x_k)\,. \tag{3}$$

式 (3) に忠実に従って，補間の分点 x_k，k=1, 2, . . . , n が指定されたときに係数 $w_j = 1/\ell'(x_j)$，j=1, 2, . . . , n を計算する方法を `Fortran90` の記法で書いたものを示す（**図 1**）．必要な四則演算の回数は，乗算 $n^2 - n$ 回，減算が $n^2 - n$ 回，除算が n 回である．

また，内側のループ内での条件分岐をやめて，内側のループを 2 つに分けて書いたものを示す（**図 2**）．必要な四則演算の回数は変わらない．

さらに，式 $(x_j - x_k)$ の添字 j と k についての反対称性を利用した計算法を示す（**図 3**）．ただしこの場合にはコンパイラの最適化によっては丸め誤差の影響により結果は上のものと必ずしも完全に同じにはならないことがある．必要な演算の回数は乗算が $n^2 - n$ 回，減算が $n^2 - n$ 回（そのうちのちょうど半数は単なる符号反転である），除算が n 回である．さらにこのコードを変形して `J` のループの 2 番目と 3 番目を融合させることもできる．

図中で `Fortran90` の記法により示した計算方法は極めて素直に書かれたものであるが，どれもコンパイラの最適化の能力により共通部分式や，ループ不変式を認識し，コードでは配列要素として書かれているものを一時変数に置き換えて実行時にレジスタ上に値を置くことなどができるものである．

これら通常の計算方法は，内側のループ内に条件分岐を含むか，あるいは内側のループをループ長の変化する 2 つのループに分断してしまうことなどにより，演算をパイプ

[1] 東京都立大学 数理科学専攻
Department of Mathematical Sciences, Tokyo Metropolitan University, Hachioji, Tokyo 192–0397, Japan
[a)] mrkmhrsh@tmu.ac.jp

```
DO  J = 1, N
  W(J) = 1
  DO  K = 1, N
    IF (J /= K) THEN
      W(J) = W(J) * (X(J) - X(K))
    ENDIF
  ENDDO
  W(J) = 1 / W(J)
ENDDO
```

図 1 通常の計算法（基本形）

```
DO  J = 1, N
  W(J) = 1
  DO  K = 1, J-1
    W(J) = W(J) * (X(J) - X(K))
  ENDDO
  DO  K = J+1, N
    W(J) = W(J) * (X(J) - X(K))
  ENDDO
  W(J) = 1 / W(J)
ENDDO
```

図 2 通常の計算法（変形 A）

```
DO  J = 1, N
  W(J) = 1
ENDDO
DO  J = 1, N-1
  DO  K = J+1, N
    W(J) = W(J) * (X(J) - X(K))
    W(K) = - W(K) * (X(J) - X(K))
  ENDDO
ENDDO
DO  J = 1, N
  W(J) = 1 / W(J)
ENDDO
```

図 3 通常の計算法（変形 B）

ライン処理で行うのには少し性質が悪い．そこで，本研究では計算法を見直すことにより性質の改良を試みる．

1.2 係数 w_j の今回の計算法

計算は倍精度や単精度などの浮動小数点数とその演算を用いて普通に行うものとする．補間の分点 x_j，j=1, 2, . . . , n などの値は本質的に丸め誤差を含む近似値なので，計算結果にも誤差を許容して厳密性を要求しないことにするならば，以下のような計算方法を用いることができる．

いま添字 j=1, 2, . . . , n について補間の分点 x_j の値をごく僅かだけ（一般的には分点ごとに独立に）それぞれずらしたものを y_j とすれば，多項式 $\ell(x)$ の Taylor 展開から式 (4) が導かれる（肩のダッシュは微分を表す）．

$$\ell(y_j) = \ell(x_j) + \ell'(x_j)(y_j - x_j) + \tfrac{1}{2}\ell''(x_j)(y_j - x_j)^2 + O((y_j - x_j)^3)\,. \tag{4}$$

すると $\ell(x_j) = 0$ であることを用いて式 (5) が得られる．

$$\ell'(x_j) = \frac{\ell(y_j)}{y_j - x_j} - \frac{1}{2}\ell''(x_j)(y_j - x_j) + O((y_j - x_j)^2)\,. \tag{5}$$

よって項 $-\frac{1}{2}\ell''(x_j)(y_j - x_j)$ およびその後の展開項を無視する近似により式 (6) が得られる．

$$\ell'(x_j) \approx \frac{\ell(y_j)}{y_j - x_j} = \frac{\prod_{k=1}^{n}(y_j - x_k)}{y_j - x_j}\,. \tag{6}$$

式 (6) の右辺の分子と分母を別々に計算する際に，分子の式を計算するときの乗算でオーバーフローやアンダーフローが発生しないと仮定すれば，（仮定した近似の下で）浮動小数点数による演算の性質から，w_j の値は式 (7) により計算できる．

$$\begin{cases} s_j \leftarrow \ell(y_j) \equiv \displaystyle\prod_{k=1}^{n}(y_j - x_k)\,, \\ w_j \leftarrow \dfrac{y_j - x_j}{s_j}\,. \end{cases} \tag{7}$$

微小な数値である $(y_j - x_j)$ は，（それが零でさえなければ）桁落ちでどれだけ精度が低下していてもかまわない．なぜならば同じ値が s_j の計算式にも乗算の因子として含まれており，s_j の計算の過程ではアンダーフローは起きないと仮定しているので，乗じた因子はその後の分子と分母の割算でほぼ正確に除かれる．たとえば，y_j の数値は浮動小数点数として x_j の数値に隣接するほど近くても良い．

補間の分点 x_k と y_k，k=1, 2, . . . , n を与えて，それらから係数 w_j，j=1, 2, . . . , n を求める計算法の例を，Fortran90 の記法により以下に示す（**図 4**）．必要な演算の回数は，乗算 n^2 回，減算が $n^2 + n$ 回，除算が n 回である．（なお，このコードを少し変更して配列要素 Y(J) の値が必要になったらすぐその場で X(J) の値から作るようにするなら，配列 Y 自体を持つことは必要でなくなる．）

```
DO  J = 1, N
  W(J) = 1
  DO  K = 1, N
    W(J) = W(J) * (Y(J) - X(K))
  ENDDO
  W(J) = (Y(J) - X(J)) / W(J)
ENDDO
```

図 4 今回の計算法

係数 w_j を求める今回の計算法の利点は，内側のループに条件分岐が無いこと，あるいは内側のループの分割をせず

に常に同じループ長 n で計算ができることである．短所は，近似が導入されていることと，n 個の値 y_k，k=1, 2, . . ., n を作成して配列に保持することが必要になることである．

1.3 補間多項式の値の計算法

関数 $f(x)$ の Lagrange 補間多項式 $p(x)$ の $x = z$ に於ける値は，z がどの補間点にも一致または極端に近接していなければ，式 (8) を計算することで求められる [1].

$$p(z) \leftarrow \ell(z) \sum_{j=1}^{n} \frac{w_j}{z - x_j} f_j \,. \tag{8}$$

いま要素数 n の Fortran90 の配列 X，W，F にそれぞれ補間の分点 x_j，係数 w_j，関数値 $f_j = f(x_j)$ が格納されているときに，引数の値 z に対する補間多項式の値 $p = p(z)$ の計算法の例を Fortran90 の記法で示す（**図 5**）．この計算に必要な四則演算の回数は，加算 n 回，減算 n 回，乗算 $2n + 1$ 回，除算 n 回である．（複数の引数の値 z_ℓ に対する多項式の値 $p(z_\ell)$ を評価することが必要になることがよくあるが，その場合は Z と P と A を配列にして処理を書けばよい）.

```
P = 0
A = 1
DO  J = 1, N
  P = P + W(J) / (Z - X(J)) * F(J)
  A = A * (Z - X(J))
ENDDO
P = P * A
```

図 5 補間多項式の値 $p(z)$ の計算法

さらに別の形式で表した補間多項式の計算法もよく知られている．それは次の形の式 (9)（重心形式，barycentric form）によるものである [1].

$$p(z) \leftarrow \frac{\displaystyle\sum_{j=1}^{n} \frac{w_j}{z - x_j} f_j}{\displaystyle\sum_{j=1}^{n} \frac{w_j}{z - x_j}} \,. \tag{9}$$

これは，式 (8) において，定数関数 1 を補間した式を考えてそれにより関数 $f(x)$ の補間式 (8) を割ることにより共通の多項式 $\ell(z)$ を除去したものとして容易に導かれる．

ただし本報告では，係数 w_j が決まった後の補間多項式の値の計算法の比較や実際の計算の実験例などについては省略する．

1.4 実験例

計算に用いたシステムは，東京大学情報基盤センターの Oakbridge-CX の 1 ノード（CPU は Dual で Intel Xeon 8280（2.7GHz，28cores），共有メモリは 192GiB，DUAL CPU の 28 コアがすべてフル稼働した場合の倍精度のピーク演算性能は 4.8TFLOPS）である．プログラムのソースコードの記述には Fortran90 を用いた．計算に用いた数値と演算は IEEE 754 の倍精度浮動小数点（2 進，64bit）である．コンパイラは Intel Fortran（version 19.1.3.304）で，コンパイラのオプションには ”-Ofast” を指定した．今回の実験ではスレッド並列化はまったく行っておらず，1CPU の 1 コアだけによる計算である．

Lagrange 補間式に必要な係数 w_j，j=1, 2, . . ., n を計算する経過時間を測定する．今回の実験で採用した補間の分点は，$x_k = \cos\{(k-1)\pi/(n-1)\}$，$k$=1, 2, . . ., n とした（これらは閉区間 $[-1, 1]$ 内の $(n-1)$ 次 Chebyshev 多項式の極値点全体である）．そうして簡単のため，y_k の値はどれも対応する x_k の値に同一の値 ϵ を加えて作った．ここで ϵ は Fortran90 の倍精度に対するマシンイプシロン EPSILON(1.D0) で，その値は約 2.22×10^{-16} である．

通常の計算法による経過時間の表とグラフをそれぞれ**表 1** と**図 6** に示す．通常の計算法（基本形）による経過時間と今回の計算法による経過時間を比較するための表とグラフをそれぞれ**表 2** と**図 7** に示す（基本形の数値が 2 つの表あるいはグラフの間で異なっているのは，それぞれ異なる実行で得られた測定結果だからである）．同表からは，通常の計算法（基本形）に対して今回の計算法を用いることで，経過時間の短縮による計算の速度向上比として 1.4 倍〜2.25 倍程度が得られていることがわかる．経過時間の値はどれも各計算方法を素朴にコーディングした Fortran90 のサブルーチンを 10^7 回繰り返して呼び出すのに掛かった経過時間を呼び出した回数で割って求めた 1 回あたりの平均値である．

同様に，同じソースコードを gfortran バージョン 4.8.5 でオプション ”-Ofast” でコンパイルして実行した例も示す（**表 3**, **表 4**, **図 8**，**図 9**）．　こちらの場合も，通常の計算法（基本形）に対して今回の計算法を用いることで，計算の速度向上比として 1.1 倍〜2.1 倍程度が得られていることが表からわかる．

表 1　通常の計算法による経過時間（秒）（ifort）

n	基本形	変形 A	変形 B
5	2.81E-8	2.38E-8	2.11E-8
6	3.62E-8	2.96E-8	2.74E-8
7	4.31E-8	3.68E-8	3.61E-8
8	4.76E-8	4.54E-8	4.43E-8
9	5.54E-8	5.36E-8	5.79E-8
10	5.92E-8	5.73E-8	5.65E-8
11	7.19E-8	6.85E-8	7.06E-8
12	8.60E-8	8.34E-8	7.69E-8
13	9.10E-8	9.13E-8	8.60E-8
14	9.81E-8	1.03E-7	8.82E-8
15	1.14E-7	1.17E-7	1.06E-7
16	1.23E-7	1.32E-7	1.09E-7
17	1.47E-7	1.59E-7	1.37E-7
18	1.47E-7	1.57E-7	1.24E-7
19	1.76E-7	1.92E-7	1.58E-7
20	1.75E-7	1.88E-7	1.41E-7
21	2.02E-7	2.20E-7	1.70E-7
22	2.19E-7	2.35E-7	1.70E-7
23	2.32E-7	2.48E-7	1.89E-7
24	2.44E-7	2.76E-7	1.97E-7
25	2.65E-7	2.86E-7	2.24E-7

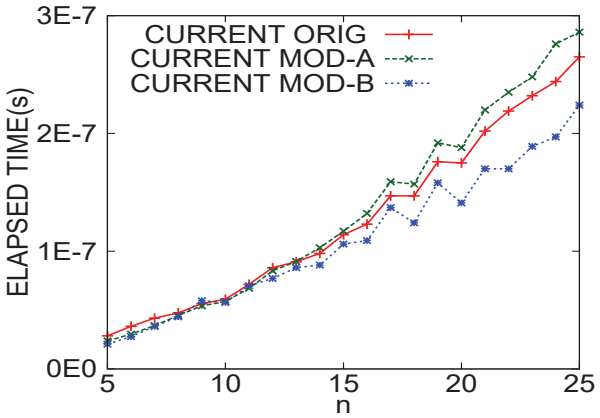

図 6　通常の方法による経過時間（ifort）

表 2　通常の方法と今回の方法による経過時間（秒）（ifort）

n	通常（基本形）	今回	速度向上比
5	3.08E-8	2.04E-8	1.51
6	4.04E-8	2.32E-8	1.74
7	4.66E-8	2.67E-8	1.75
8	5.27E-8	2.96E-8	1.78
9	5.89E-8	3.27E-8	1.80
10	6.26E-8	3.99E-8	1.57
11	7.18E-8	4.51E-8	1.59
12	7.87E-8	5.34E-8	1.47
13	9.05E-8	6.13E-8	1.48
14	9.85E-8	6.94E-8	1.42
15	1.18E-7	8.38E-8	1.41
16	1.24E-7	6.04E-8	2.05
17	1.39E-7	7.02E-8	1.98
18	1.48E-7	7.99E-8	1.85
19	1.65E-7	9.23E-8	1.79
20	1.76E-7	1.06E-7	1.66
21	1.95E-7	1.18E-7	1.65
22	2.10E-7	1.32E-7	1.59
23	2.47E-7	1.62E-7	1.52
24	2.45E-7	1.09E-7	2.25
25	2.67E-7	1.35E-7	1.98

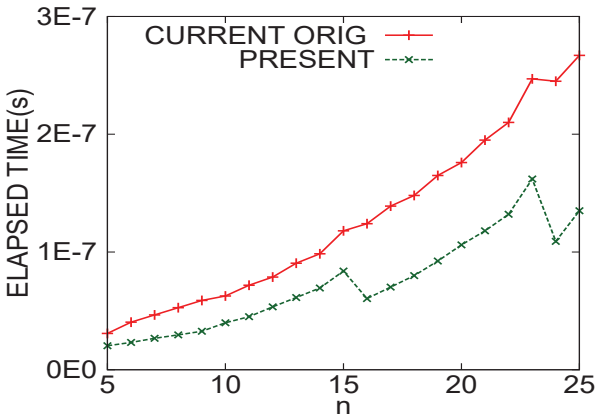

図 7　通常の方法と今回の方法による経過時間（ifort）

表 3 通常の計算法による経過時間（秒）（gfortran）

n	基本形	変形 A	変形 B
5	2.17E-8	2.27E-8	2.22E-8
6	2.76E-8	2.81E-8	2.53E-8
7	3.81E-8	3.46E-8	3.11E-8
8	4.46E-8	4.33E-8	3.68E-8
9	5.47E-8	5.35E-8	4.46E-8
10	6.57E-8	6.42E-8	5.29E-8
11	7.77E-8	7.80E-8	6.40E-8
12	9.74E-8	8.68E-8	7.63E-8
13	1.05E-7	1.01E-7	8.91E-8
14	1.20E-7	1.14E-7	1.04E-7
15	1.38E-7	1.28E-7	1.20E-7
16	1.56E-7	1.41E-7	1.38E-7
17	1.74E-7	1.58E-7	1.57E-7
18	2.01E-7	1.72E-7	1.79E-7
19	2.60E-7	2.00E-7	2.14E-7
20	3.47E-7	2.04E-7	2.29E-7
21	2.89E-7	2.30E-7	2.68E-7
22	3.02E-7	2.39E-7	2.84E-7
23	3.36E-7	2.62E-7	3.14E-7
24	3.74E-7	2.80E-7	3.44E-7
25	4.23E-7	3.02E-7	3.96E-7

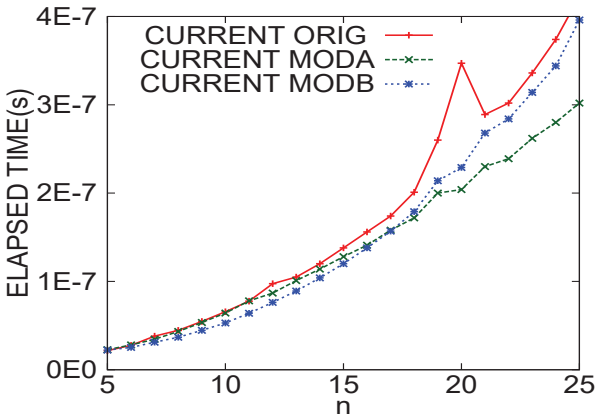

図 8 通常の方法による経過時間（gfortran）

表 4 通常の方法と今回の方法による経過時間（秒）（gfortran）

n	通常（基本形）	今回	速度向上比
5	1.78E-8	1.50E-8	1.19
6	2.33E-8	1.65E-8	1.41
7	3.06E-8	2.53E-8	1.21
8	3.88E-8	3.50E-8	1.11
9	4.84E-8	3.69E-8	1.31
10	5.84E-8	4.26E-8	1.37
11	6.97E-8	5.55E-8	1.26
12	8.74E-8	6.34E-8	1.38
13	1.03E-7	7.29E-8	1.41
14	1.22E-7	7.94E-8	1.54
15	1.33E-7	9.09E-8	1.46
16	1.58E-7	9.88E-8	1.60
17	1.77E-7	1.12E-7	1.58
18	1.96E-7	1.21E-7	1.62
19	2.44E-7	1.35E-7	1.81
20	3.05E-7	1.46E-7	2.09
21	2.87E-7	1.62E-7	1.77
22	3.03E-7	1.75E-7	1.73
23	3.38E-7	1.95E-7	1.73
24	3.77E-7	2.07E-7	1.82
25	4.15E-7	2.27E-7	1.83

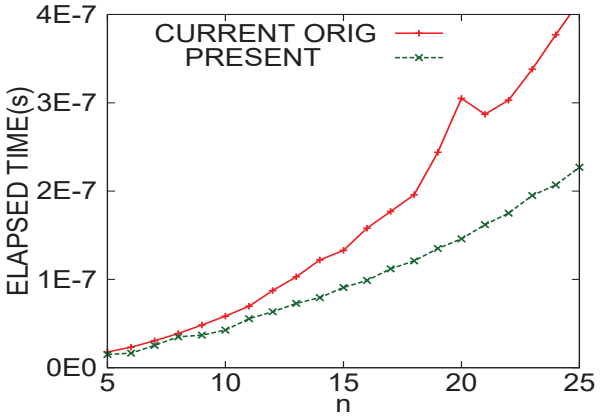

図 9 通常の方法と今回の方法による経過時間（gfortran）

表 5 今回の近似方法による係数 **w** の誤差の例（ifort）

n	最大絶対誤差	最大相対誤差
5	1.3E-15	6.7E-16
6	2.9E-15	9.0E-16
7	7.1E-15	1.3E-15
8	1.7E-14	1.8E-15
9	3.9E-14	2.4E-15
10	9.1E-14	3.2E-15
11	1.8E-13	3.5E-15
12	4.3E-13	4.7E-15
13	9.4E-13	5.5E-15
14	1.0E-12	6.1E-15
15	4.3E-12	7.4E-15
16	9.1E-12	8.3E-15
17	2.0E-11	1.0E-14
18	4.2E-11	1.1E-14
19	8.8E-11	1.2E-14
20	1.9E-10	1.4E-14
21	3.9E-10	1.5E-14
22	8.1E-10	1.6E-14
23	1.7E-09	1.8E-14
24	3.6E-09	2.0E-14
25	7.4E-09	2.1E-14

表 6 今回の近似方法による係数 **w** の誤差の例（gfortran）

n	最大絶対誤差	最大相対誤差
5	1.3E-15	6.7E-16
6	2.9E-15	9.0E-16
7	8.0E-15	1.5E-15
8	1.7E-14	1.8E-15
9	3.9E-14	2.4E-15
10	8.7E-14	3.1E-15
11	1.9E-13	3.7E-15
12	4.1E-13	4.4E-15
13	9.1E-13	5.3E-15
14	1.9E-12	6.1E-15
15	4.3E-12	7.3E-15
16	9.0E-12	8.2E-15
17	2.0E-11	9.7E-15
18	4.2E-11	1.1E-14
19	8.7E-11	1.2E-14
20	1.9E-10	1.4E-14
21	3.9E-10	1.5E-14
22	8.3E-10	1.7E-14
23	1.7E-09	1.8E-14
24	3.6E-09	2.0E-14
25	7.4E-09	2.1E-14

今回の方法では，計算を規則正しく行うために，係数 w_j，j=1, 2, . . ., n を求める計算式の段階で x_j に摂動を加えて y_j に置き換えるという近似を導入しているので，それによる誤差が入る．いまの実験例では，x_j は $(n-1)$ 次の Chebyshev 多項式の極値点であり，$y_j = x_j + \epsilon$ である．

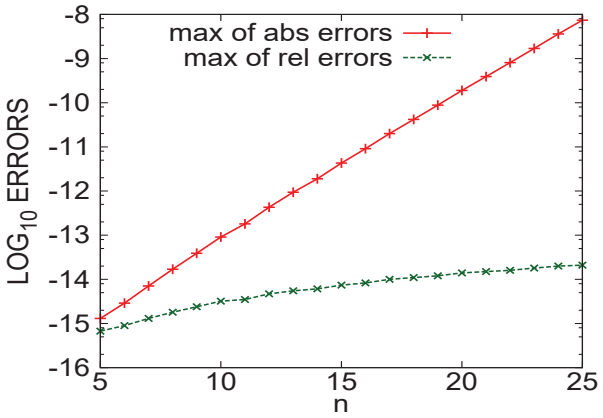

図 10 今回の近似方法による係数 **w** の誤差の例（ifort）

ここで ϵ は Fortran90 の倍精度のマシンイプシロンの値である．この場合について，通常の方法（基本形）で計算された係数を w_j とし，今回の方法により計算された係数を $\tilde{w}_j$ とするときに，係数の絶対誤差の最大値である「最大絶対誤差」$e_{\max} \equiv \max_j |w_j - \tilde{w}_j|$ と係数の相対誤差の最大値である「最大相対誤差」$e_{\max}/\max_j |w_j|$ を求めたものをそれぞれ表とグラフに示す（**表 5**，**図 10**，**表 6**）．表の値やグラフから，n が増加するとき，最大絶対誤差の増加に比べて最大相対誤差の増加は比較的緩やかであることがわかる（ただし誤差の値は分点 x_k の分布にも依存する）．

1.5 まとめ

与えられた一般的な補間の分点 x_k，k=1, 2, . . ., n に対して，Lagrange 補間の計算に必要な係数（今回の w_j，j=1, 2, . . ., n）を n^2 に比例する手間で作る処理を行う際に，その計算方法に変更を少し加えることで，計算機内部での処理をより円滑にする改良を試みた．今回の計算式の変更により生じる誤差は，浮動小数点数による数値計算では数値や演算での丸め誤差を伴うので，それが小さいならば許容できるものと考えられる．

計算機システム（CPU が Intel Xeon 8280 のシステム）を実際に用いて 1 コア 1 スレッドで計算を行い経過時間を測定してみたところ，今回の方法を採用することにより，通常の方法に比べて Intel の Fortran90 コンパイラを使用した場合には 1.4 倍〜2.0 倍程度の計算速度の向上が得られることを確認した．

参考文献

[1] Jean-Paul Berrut and Lloyd N. Trefethen: ”Barycentric Langrange Interpolation”, *SIAM Review*, Vol.46, No.3 (2004), pp.501–517.