第12章　補外法を用いた数値積分への応用

$s_c \in \mathbb{R}$ へ収束する数列 $s_0, s_1, ..., s_n, ...$ $(s_i \in \mathbb{R})$ の一般項 s_i が，ある多項式関数 $S(x)$ を用いて

$$s_i = S(x_i) = s_c + c_1 x_i + c_2 x_i^2 + \cdots + c_m x_i^m + \cdots \tag{12.1}$$

と表現できるものとする。

(12.1) 式より，収束値は $s_c = S(0)$ と表現できるため，$S(x)$ を多項式補間によって構成できれば，補間多項式によってより s_c に近い値を得ることが期待できる。

今，無限大 に発散する補助数列 $w_0, w_1,...,w_n, ...$ $(w_i > 0)$ を用いて $x_i = h/w_i$ とし，$\{x_i\}$ を 0 に収束するように設定する。補間点として $(x_0, S(x_0))$, $(x_1, S(x_1))$, ..., $(x_m, S(x_m))$ を取り，$x = 0$ における補間多項式の値を Neville のアルゴリズムによって求めることを考える。

初期系列を $T_{11} := S(x_0)$, $T_{21} := S(x_1)$, ..., $T_{m+1,1} := S(x_m)$ とし，$x = 0$ における m 次補間多項式の値 $p_m(0)$ を出すための漸化式は

$$\begin{aligned} T_{ij} &:= \frac{(0 - x_{i-j+1})T_{i,j-1} - (0 - x_i)T_{i-1,j-1}}{x_i - x_{i-j+1}} \\ &= \frac{x_i T_{i-1,j-1} - x_{i-j+1} T_{i,j-1}}{x_i - x_{i-j+1}} \qquad (12.2) \\ &= T_{i,j-1} + \frac{T_{i,j-1} - T_{i-1,j-1}}{\frac{x_{i-j+1}}{x_i} - 1} \\ &= T_{i,j-1} + \frac{T_{i,j-1} - T_{i-1,j-1}}{\frac{w_i}{w_{i-j+1}} - 1} \qquad (12.3) \end{aligned}$$

と変形できる。実際に計算する際には，(12.3) 式より (12.3) の方が有限桁計算においては良いとされている。

このように，$T_{m+1,m+1}$ を求める操作は，補間点 $x_0, x_1, ..., x_m$ $(x_i > 0)$ の外部 $x = 0$ における値を求めていることになるため，**補外 (extrapolation)** と呼ばれている。

こうして得られた $T_{m+1,m+1} = p_m(0)$ は，もし 0 と補間点 x_0, x_1, ..., x_m を含む区間

表 12.1: Richardson の補外表

初期系列						
T_{11}						
	$\searrow$					
T_{21}	$\rightarrow$	T_{22}				
$\vdots$		$\vdots$	$\ddots$			
T_{m1}	$\rightarrow$	T_{m2}	$\cdots$	T_{mm}		
	$\searrow$				$\searrow$	
$T_{m+1,1}$	$\rightarrow$	$T_{m+1,2}$	$\cdots$	$T_{m+1,m}$	$\rightarrow$	$T_{m+1,m+1}$

I において $S(x)$ が $m+1$ 回連続微分可能である時,

$$|T_{m+1,m+1} - S(0)| = (-1)^{m+1} \frac{S^{(m+1)}(\xi)}{(m+1)!} \prod_{i=0}^{m} x_i \tag{12.4}$$

を満足する $\xi \in I$ が存在することになる。従って,

$$|T_{m+1,m+1} - S(0)| = O\left(\prod_{i=0}^{m} x_i\right) = O(|h|^{m+1}) \tag{12.5}$$

となり，$O(|h|)$ で収束する $\{s_i\}$ よりも高速に，$m+1$ 次の速さで収束することが期待できる。

もし $S(x)$ が (12.1) ではなく，定数 $\alpha \geq 1$ に対して

$$s_i = S(x_i) = s_c + c_1 x_i^{\alpha} + c_2 x_i^{2\alpha} + \cdots + c_m x_i^{m\alpha} + \cdots \tag{12.6}$$

という形で表現できるのであれば，(12.3) は

$$T_{ij} := T_{i,j-1} + \frac{T_{i,j-1} - T_{i-1,j-1}}{\left(\frac{w_i}{w_{i-j+1}}\right)^{\alpha} - 1} \tag{12.7}$$

となる。この時，収束の速さは

$$|T_{m+1,m+1} - S(0)| = O(|h|^{(m+1)\alpha}) \tag{12.8}$$

であることが期待される。

実際に補外計算を行う際には最大段数 (行数)$L \geq 2$ を決めておき，まず $T_{11} = S(h/w_0)$ と $T_{21} = S(h/w_1)$ を求め,

$$\begin{array}{ccc} T_{11} & & \\ & \searrow & \\ T_{21} & \rightarrow & \underline{T_{22}} \end{array}$$

の T_{22} の計算において収束判定を行う。もし収束していなければ $T_{31} = S(h/w_2)$ を求めて

$$\begin{array}{ccccc} T_{11} & & & & \\ & & & & \\ T_{21} & & T_{22} & & \\ & \searrow & & \searrow & \\ T_{31} & \rightarrow & \underline{T_{32}} & \rightarrow & \underline{T_{33}} \end{array}$$

のように，T_{32}, T_{33} の計算を収束判定を行いつつ実施する。このように，なるべく計算時間が最も必要な (場合の多い) 初期系列の段数を少なく抑えることで，補外計算全体の効率化を図ることが出来る。

補助数列 $\{w_i\}$ としては，BNCpack では次の 2 つを用いた補外法が実装されている。

Romberg 数列: $1, 2, 4, 8, ..., 2^i, ...$

調和 (harmonic) 数列: $1, 2, 3, 4, ..., i + 1, ...$

ここでは，5.2 節の台形則を初期系列 T_{i1} の値として採用し，この収束を更に加速することを考える。このような補外を行った数値積分を **Romberg 型数値積分**と呼ぶ。

12.1 逐次計算プログラム

次の定積分を求める Romberg 型数値積分のプログラムをここでは示す。

$$\int_0^1 \exp(x)\, dx = e - 1$$

倍精度計算の場合は次のようになる。

extrap.c

```
 1 :#include <stdio.h>
 2 :#include <stdlib.h>
 3 :#include <math.h>
 4 :
 5 :#include "bnc.h"
 6 :
 7 :/* Integration Interval */
 8 :void dinterval(double *a, double *b)
 9 :{
10 :    *a = 0.0;
11 :    *b = 1.0;
12 :}
13 :
14 :/* test function : f(x) = exp(x) */
15 :double df(double x)
16 :{
17 :    return exp(x);
18 :}
19 :
20 :int main(int argc, char *argv[])
21 :{
22 :    long int num_div;
23 :    double da, db, dans;
24 :    double stime, etime;
25 :
26 :
27 :/* double */
28 :    dinterval(&da, &db);
29 :
30 :    printf("IEEE double\n");
31 :    printf("Romberg Integral[%25.17e, %25.17e]\n", da, db);
32 :
33 :    for(num_div = 8; num_div <= 128; num_div *= 2)
34 :    {
35 ://        printf(" %10d , %25.17e , ", num_div, dromberg_integral
   (da, db, df, num_div));
36 :        stime = get_secv();
37 :        //dans = dmromberg_integral(da, db, df, num_div);
38 :        dans = dmharmonic_integral(da, db, df, num_div);
39 :        etime = get_secv();
40 :        printf(" %10d , %25.17e, %10.3e", num_div, dans, etime -
   stime);
41 ://        printf("Y(DOUBLE):\n"); print_dmatrix(dymat);
42 ://        printf("YDIFF    :\n"); print_dmatrix(dydiffmat);
43 :        printf("\n");
```

```
 44 :     }
 45 :
 46 :     return EXIT_SUCCESS;
 47 : }
```

多倍長計算の場合は次のようになる。

extrap-gmp.c

```
  1 : #include <stdio.h>
  2 : #include <stdlib.h>
  3 : #include <math.h>
  4 :
  5 : #define USE_GMP
  6 : #define USE_MPFR
  7 : #include "bnc.h"
  8 :
  9 : /* Integration Interval */
 10 : void mpf_interval(mpf_t a, mpf_t b)
 11 : {
 12 :     mpf_set_ui(a, 0UL);
 13 :     mpf_set_ui(b, 1UL);
 14 : }
 15 :
 16 : /* test function : f(x) = exp(x) */
 17 : void mpf_f(mpf_t ret, mpf_t x)
 18 : {
 19 :     mpf_exp(ret, x);
 20 : }
 21 :
 22 : int main(int argc, char *argv[])
 23 : {
 24 :     long int num_div;
 25 :     double stime, etime;
 26 :     mpf_t mpf_a, mpf_b, mpf_ans;
 27 :
 28 :
 29 : #define DPREC 50
 30 :     set_bnc_default_prec_decimal(DPREC);
 31 :
 32 :     mpf_init(mpf_ans);
 33 :     mpf_init(mpf_a); mpf_init(mpf_b);
 34 :     mpf_interval(mpf_a, mpf_b);
 35 :
 36 :     printf("MPF(%d bits)\n", get_bnc_default_prec());
 37 :     printf("Romberg Integral[");
```

```
38 :
39 :     mpf_out_str(stdout, 10, 0, mpf_a); printf(", ");
40 :     mpf_out_str(stdout, 10, 0, mpf_b); printf("]\n");
41 :     for(num_div = 8; num_div <= 128; num_div *= 2)
42 :     {
43 :         stime = get_secv();
44 ://         mpf_mromberg_integral(mpf_ans, mpf_a, mpf_b, mpf_f, num
   _div);
45 :         mpf_mharmonic_integral(mpf_ans, mpf_a, mpf_b, mpf_f, num_
   div);
46 ://         mpf_mharmonic1_integral(mpf_ans, mpf_a, mpf_b, mpf_f, n
   um_div);
47 ://         mpf_mharmonic2_integral(mpf_ans, mpf_a, mpf_b, mpf_f, n
   um_div);
48 ://         mpf_mharmonic3_integral(mpf_ans, mpf_a, mpf_b, mpf_f, n
   um_div);
49 :         etime = get_secv();
50 :
51 ://         printf("Y(MPF%d):\n", get_bnc_default_prec()); print_mp
   fmatrix(mpfymat);
52 ://         printf("YDIFF    :\n"); print_mpfmatrix(mpfydiffmat);
53 :         printf("%5d, ",num_div); mpf_out_str(stdout, 10, 0, mpf_a
   ns);
54 :         printf(", %10.3e", etime - stime);
55 :
56 :         printf("\n");
57 :     }
58 :
59 :     mpf_clear(mpf_ans);
60 :     mpf_clear(mpf_a);
61 :     mpf_clear(mpf_b);
62 :
63 :     return EXIT_SUCCESS;
64 : }
```

12.2　並列計算プログラム

前述の逐次計算プログラムを並列化すると，以下のようなプログラムになる。

mpi-extrap.c

```
1 : #include <stdio.h>
2 : #include <stdlib.h>
3 : #include <math.h>
```

```
 4 :
 5 : #include "mpi.h"
 6 :
 7 : #include "mpi_bnc.h"
 8 :
 9 : /* Integration Interval */
10 : void dinterval(double *a, double *b)
11 : {
12 :     *a = 0.0;
13 :     *b = 1.0;
14 : }
15 :
16 : /* test function : f(x) = exp(x) */
17 : double df(double x)
18 : {
19 :     return exp(x);
20 : }
21 :
22 : int main(int argc, char *argv[])
23 : {
24 :     long int num_div;
25 :     double da, db, dans;
26 :     double stime, etime;
27 :     int myrank, num_procs;
28 :
29 : /* for MPI */
30 :     MPI_Init(&argc, &argv);
31 :     MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
32 :     MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
33 :
34 : /* double */
35 :     dinterval(&da, &db);
36 :
37 :     if(myrank == 0)
38 :     {
39 :     printf("IEEE double\n");
40 :
41 :     printf("Romberg Integral[%25.17e, %25.17e]\n", da, db);
42 :     }
43 :
44 :     for(num_div = 8; num_div <= 128; num_div *= 2)
45 :     {
46 : //        printf(" %10d , %25.17e , ", num_div, dromberg_integral
   (da, db, df, num_div));
47 :         stime = get_secv();
48 :         //_mpi_dmromberg_integral(&dans, da, db, df, num_div, MPI
```

```
   _COMM_WORLD);
 49 :         _mpi_dmharmonic_integral(&dans, da, db, df, num_div, MPI_
   COMM_WORLD);
 50 :         etime = get_secv();
 51 :         if(myrank == 0)
 52 :         {
 53 :         printf(" %10d , %25.17e, %10.3e", num_div, dans, etime -
   stime);
 54 : //         printf("Y(DOUBLE):\n"); print_dmatrix(dymat);
 55 : //         printf("YDIFF    :\n"); print_dmatrix(dydiffmat);
 56 :         printf("\n");
 57 :         }
 58 :     }
 59 :
 60 :     MPI_Finalize();
 61 :
 62 :     return EXIT_SUCCESS;
 63 : }
```

多倍長計算プログラムを並列化すると，以下のようになる。

mpi-extrap-gmp.c

```
  1 : #include <stdio.h>
  2 : #include <stdlib.h>
  3 : #include <math.h>
  4 :
  5 : #include "mpi.h"
  6 :
  7 : #define USE_GMP
  8 : #define USE_MPFR
  9 : #include "mpi_bnc.h"
 10 :
 11 : /* Integration Interval */
 12 : void mpf_interval(mpf_t a, mpf_t b)
 13 : {
 14 :     mpf_set_ui(a, 0UL);
 15 :     mpf_set_ui(b, 1UL);
 16 : }
 17 :
 18 : /* test function : f(x) = exp(x) */
 19 : void mpf_f(mpf_t ret, mpf_t x)
 20 : {
 21 :     mpf_exp(ret, x);
 22 : }
 23 :
```

```
 24 : int main(int argc, char *argv[])
 25 : {
 26 :     long int num_div;
 27 :     double stime, etime;
 28 :     mpf_t mpf_a, mpf_b, mpf_ans;
 29 :     int myrank, num_procs;
 30 :
 31 : /* for MPI */
 32 :     MPI_Init(&argc, &argv);
 33 :     MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
 34 :     MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
 35 :
 36 : #define DPREC 50
 37 : //    set_bnc_default_prec(128);
 38 :     _mpi_set_bnc_default_prec_decimal(DPREC, MPI_COMM_WORLD);
 39 :     commit_mpf(&(MPI_MPF), ceil(DPREC/log10(2.0)), MPI_COMM_WORLD
    );
 40 :     create_mpf_op(&(MPI_MPF_SUM), _mpi_mpf_add, MPI_COMM_WORLD);
 41 :
 42 :     mpf_init(mpf_ans);
 43 :     mpf_init(mpf_a); mpf_init(mpf_b);
 44 :     mpf_interval(mpf_a, mpf_b);
 45 :
 46 :     if(myrank == 0)
 47 :     {
 48 :     printf("MPF(%d bits)\n", get_bnc_default_prec());
 49 :     printf("Romberg Integral[");
 50 :
 51 :     mpf_out_str(stdout, 10, 0, mpf_a); printf(", ");
 52 :     mpf_out_str(stdout, 10, 0, mpf_b); printf("]\n");
 53 :     }
 54 :     for(num_div = 8; num_div <= 128; num_div *= 2)
 55 :     {
 56 : //        mpf_trapezoidal_fs(mpf_ans, mpf_a, mpf_b, mpf_f, num_di
    v);
 57 : //        mpf_romberg_integral(mpf_ans, mpf_a, mpf_b, mpf_f, num_
    div);
 58 :         stime = MPI_Wtime();
 59 : //        mpf_mromberg_integral(mpf_ans, mpf_a, mpf_b, mpf_f, num
    _div);
 60 :         _mpi_mpf_mharmonic_integral(mpf_ans, mpf_a, mpf_b, mpf_f,
     num_div, MPI_COMM_WORLD);
 61 : //        mpf_mharmonic1_integral(mpf_ans, mpf_a, mpf_b, mpf_f, n
    um_div);
 62 : //        mpf_mharmonic2_integral(mpf_ans, mpf_a, mpf_b, mpf_f, n
    um_div);
```

```
63 ://        mpf_mharmonic3_integral(mpf_ans, mpf_a, mpf_b, mpf_f, n
   um_div);
64 :         etime = MPI_Wtime();
65 :         if(myrank == 0)
66 :         {
67 ://        printf("Y(MPF%d):\n", get_bnc_default_prec()); print_mp
   fmatrix(mpfymat);
68 ://        printf("YDIFF    :\n"); print_mpfmatrix(mpfydiffmat);
69 :         printf("%5d, ",num_div); mpf_out_str(stdout, 10, 0, mpf_a
   ns);
70 :         printf(", %10.3e", etime - stime);
71 :
72 :         printf("\n");
73 :         }
74 :     }
75 :
76 :     mpf_clear(mpf_ans);
77 :     mpf_clear(mpf_a);
78 :     mpf_clear(mpf_b);
79 :
80 :     MPI_Finalize();
81 :
82 :     return EXIT_SUCCESS;
83 :}
```

演習問題

次の定積分を求める補外法のプログラムを作り，実際に計算せよ。また並列化したときの計算時間の推移も調べよ。

1. $$\int_0^1 x^2 dx = \frac{1}{3}$$

2. $$\int_0^1 \frac{1}{1+x^2}\, dx = \frac{\pi}{4}$$

3. $$\int_0^1 \frac{\cos x}{\sqrt{1-x^2}} dx \approx 1.2019697\cdots$$