第11章　DKA法の実例

本章では，実際に多倍長計算が必要な実例を一つ取り上げ，ネチネチとその過程を追いかけることにしたい。今までは特に考えなくIEEE754倍精度と多倍長精度の例を示してきたが，計算時間を考えれば，なるべく多倍長計算はしたくない，ということになる。にもかかわらず，それが必要な事例もある，ということを本章で味わっていただきたい。

11.1　DKA法のアルゴリズム

一変数n次多項式

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0 \tag{11.1}$$

に対して次の代数方程式

$$p(x) = 0 \tag{11.2}$$

を解くことにする。

ここでは$p_n(x)$を

$$q_n(x) = x^n + c_{n-1} x^{n-1} + \cdots + c_1 x + c_0$$

と変形し

$$q_n(x) = 0 \tag{11.3}$$

という代数方程式について考えることにする。ここで

$$c_i = \frac{a_i}{a_n} \ (i = 0, 1, ..., n-1)$$

である。

代数方程式(11.3)に対してn個の解$\alpha_1, \alpha_2, ..., \alpha_n$と係数の関係式を書き下してみ

ると

$$\left\{\begin{array}{rcl} (-1)^1 \sum_{i=1}^{n} \alpha_i - c_{n-1} &=& 0 \\ (-1)^2 \sum_{i_1<i_2}^{n} \alpha_{i_1}\alpha_{i_2} - c_{n-2} &=& 0 \\ (-1)^3 \sum_{i_1<i_2<i_3}^{n} \alpha_{i_1}\alpha_{i_2}\alpha_{i_3} - c_{n-3} &=& 0 \\ & \vdots & \\ (-1)^{n-1} \sum_{i_1<i_2<\cdots<i_{n-1}}^{n} \alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_{n-1}} - c_1 &=& 0 \\ (-1)^n \alpha_1\alpha_2\cdots\alpha_n - c_0 &=& 0 \end{array}\right. \tag{11.4}$$

となる。この n 次元非線型方程式 (11.4) の解は代数方程式 (11.3) の解でもある。これに Newton 法を適用する方法を DKA 法 (Durand-Kerner-Aberth 法)[1] と呼ぶ。

1. 初期値 $z_i^{(0)}$ $(i = 1, 2, ..., n)$ を設定する。
2. $k = 0, 1, 2, ...$ に対して次の反復を繰り返す。

$$z_i^{(k+1)} := z_i^{(k)} - \frac{q_n(z_i^{(k)})}{\prod_{j=1, j\neq i}^{n}(z_i^{(k)} - z_j^{(k)})} \tag{11.5}$$

この際，初期値としては Aberth が提案した

$$z_i^{(0)} := -\frac{c_{n-1}}{n} + r\exp\left(\frac{2(i-1)\pi}{n} + \sqrt{-1}\frac{3}{2n}\right) \tag{11.6}$$

が良いとされている。ここで r は全ての根が複素平面上の円盤

$$\left|z + \frac{c_{n-1}}{n}\right| \leq r$$

に存在するように取る。例えば伊理 [11] は実係数代数方程式

$$x^n - |c_{n-2}|x^{n-2} - |c_{n-3}|x^{n-3} - \cdots - |c_1|x - |c_0| = 0 \tag{11.7}$$

の正の実数根を r とすることを提案している。

[1] 一松 [10] によれば，Durand が代数方程式用に Newton 法を改良し，Kerner が対称式と係数の関係式からの解釈を与え，Aberth が初期値を提案した，という流れになるらしい。

11.2 逐次計算プログラム

テスト用として，代数方程式の左辺の多項式 (11.1) を次のようにする。

$$\begin{aligned}\prod_{i=1}^{20}(x-i) = {} & x^{20} - 210x^{19} + 20615x^{18} - 1256850x^{17} + 53327946x^{16} - 1672280820x^{15} \\ & + 40171771630x^{14} - 756111184500x^{13} + 11310276995381x^{12} \\ & - 135585182899530x^{11} + 1307535010540395x^{10} - 10142299865511450x^{9} \\ & + 63030812099294896x^{8} - 311333643161390640x^{7} \\ & + 1206647803780373360x^{6} - 3599979517947607200x^{5} \\ & + 8037811822645051776x^{4} - 12870931245150988800x^{3} \\ & + 13803759753640704000x^{2} - 8752948036761600000x^{1} \\ & + 2432902008176640000\end{aligned} \tag{11.8}$$

一見そうは見えないが，この解はかなりの近接状態になっており，多倍長計算でなければ十分な精度が得られない問題として知られている。

この多項式の係数を”polycoef20.dat”ファイルに格納し，逐次計算の DKA 法で解を求めるプログラムは以下のようになる。

dka.c

```
 1 : #include <stdio.h>
 2 : #include <stdlib.h>
 3 : #include <math.h>
 4 :
 5 : #include "bnc.h"
 6 :
 7 : #define MAX_LENGTH 1024
 8 :
 9 : #define DEG 20
10 :
11 : int main(int argc, char *argv[])
12 : {
13 :     long int i, dtimes;
14 :     FILE *dcoef;
15 :
16 :     CDArray cdans, cdinit;
17 :     DPoly df;
18 :     double dabs_eps, drel_eps;
19 :     double start, endtime;
```

```
 20 :
 21 :    /* init */
 22 :    dabs_eps = 1.0e-100;
 23 :    drel_eps = 1.0e-7;
 24 :    df = init_dpoly(MAX_LENGTH);
 25 :
 26 :    cdans = init_cdarray(DEG);
 27 :    cdinit = init_cdarray(DEG);
 28 :
 29 :    /* Input coefficients */
 30 :    dcoef = fopen("polycoef20.dat", "r");
 31 :    fread_dpolycoef(dcoef, df, DEG);
 32 :    fclose(dcoef);
 33 :
 34 :    print_dpoly(df);
 35 :
 36 :    /* set Aberth's initial value */
 37 :    ddka_init(cdinit, df);
 38 ://    print_cdarray(cdinit);
 39 :
 40 :    /* DKA method */
 41 :    start = get_secv();
 42 :    dtimes = ddka(cdans, cdinit, df, 1000, dabs_eps, drel_eps);
 43 :    endtime = get_secv() - start;
 44 :
 45 :    /* print answer */
 46 :    printf("Iterative times: %d\n", dtimes);
 47 :    print_cdarray(cdans);
 48 :
 49 :    /* clear */
 50 :    free_dpoly(df);
 51 :    free_cdarray(cdans);
 52 :    free_cdarray(cdinit);
 53 :
 54 :    printf("double_DKA     : %f sec\n", endtime);
 55 :
 56 :    return EXIT_SUCCESS;
 57 :}
```

多倍長計算では次のようになる。

dka-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 : #define MAX_LENGTH 1024
10 :
11 : #define MPFDEG 20
12 :
13 : int main(int argc, char *argv[])
14 : {
15 :     long int i, mpftimes;
16 :     FILE * mpfcoef;
17 :
18 :     double start, endtime;
19 :     CMPFArray cmpfans, cmpfinit;
20 :     MPFPoly mpff;
21 :     mpf_t mpfabs_eps, mpfrel_eps;
22 :
23 : #define PREC 128
24 :     set_bnc_default_prec(PREC);
25 :
26 :     /* init */
27 :     mpf_init_set_d(mpfabs_eps, 1.0e-100);
28 :     mpf_init_set_d(mpfrel_eps, 1.0e-7);
29 :
30 :     mpff = init_mpfpoly(MAX_LENGTH);
31 :     cmpfans = init_cmpfarray(MPFDEG);
32 :     cmpfinit = init_cmpfarray(MPFDEG);
33 :
34 :     cmpfans = init_cmpfarray(MPFDEG);
35 :     cmpfinit = init_cmpfarray(MPFDEG);
36 :
37 :
38 :     mpfcoef = fopen("polycoef20.dat", "r");
39 :     fread_mpfpolycoef(mpfcoef, mpff, MPFDEG);
40 :     fclose(mpfcoef);
41 :
42 :     print_mpfpoly(mpff);
43 :
44 :     /* set Aberth's initial value */
45 :     mpf_dka_init(cmpfinit, mpff);
46 : //    print_cmpfarray(local_cmpfinit);
47 :
48 :     /* DKA method */
49 :     start = get_secv();
```

```
 50 :     mpftimes = mpf_dka(cmpfans, cmpfinit, mpff, 1000, mpfabs_eps,
    mpfrel_eps);
 51 :     endtime = get_secv() - start;
 52 :
 53 :     /* print answer */
 54 :     printf("Iterative times: %d\n", mpftimes);
 55 :     print_cmpfarray(cmpfans);
 56 :
 57 :     /* clear */
 58 :     free_mpfpoly(mpff);
 59 :     free_cmpfarray(cmpfans);
 60 :     free_cmpfarray(cmpfinit);
 61 :
 62 :     printf("mpf_DKA(%dbits): %f sec\n", PREC, endtime);
 63 :
 64 :     return EXIT_SUCCESS;
 65 : }
```

11.3　並列計算プログラム

同じ問題に対して，並列版の DKA 法によるプログラムは以下のようになる。この場合，各 PE では (11.5) 式の左辺を計算し，それが終わると新たに求められた $z^{(}k+1)_i$ $(i = 1, 2, ..., n)$ を Allgather して全ての PE で共有し，次の計算に備えることで実現できる。

mpi-dka.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 : #define MAX_LENGTH 1024
 10 :
 11 : #define DEG 20
 12 :
 13 : int main(int argc, char *argv[])
 14 : {
 15 :     long int i, dtimes, mpftimes;
 16 :     FILE * dcoef, * mpfcoef;
 17 :
```

```
18 :     long int local_dim, dd_dim[MPI_GMP_MAXPROCS];
19 :     CDArray cdans, cdinit, local_cdans, local_cdinit;
20 :     DPoly df;
21 :     double dabs_eps, drel_eps;
22 :     double startwtime[2], endwtime[2];
23 :     int myrank, num_procs;
24 :
25 : /* for MPI */
26 :     MPI_Init(&argc, &argv);
27 :     MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
28 :     MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
29 :
30 : /* double */
31 :
32 :     /* init */
33 :     dabs_eps = 1.0e-100;
34 :     drel_eps = 1.0e-7;
35 :     df = init_dpoly(MAX_LENGTH);
36 :
37 :     local_dim = _mpi_divide_dim(dd_dim, DEG, num_procs);
38 :     if(myrank == 0)
39 :     {
40 : //         for(i = 0; i < DEG; i++)
41 : //             printf("%d d_dim[%d]: %d\n", local_dim, i, dd_dim[i
   ]);
42 :     }
43 :     local_cdans = init_cdarray(local_dim);
44 :     local_cdinit = init_cdarray(local_dim);
45 :
46 :     cdans = init_cdarray(DEG);
47 :     cdinit = init_cdarray(DEG);
48 :     if(myrank == 0)
49 :     {
50 :         dcoef = fopen("polycoef20.dat", "r");
51 :         fread_dpolycoef(dcoef, df, DEG);
52 :         fclose(dcoef);
53 :     }
54 :
55 :     /* Bcast dpoly */
56 :     _mpi_bcast_dpoly(df, MPI_COMM_WORLD);
57 :     if(myrank == 0)
58 :     {
59 :         printf("rank: %d\n", myrank);
60 :         print_dpoly(df);
61 :     }
62 :
```

```
 63 :     /* set Aberth's initial value */
 64 :     _mpi_ddka_init(local_cdinit, df, MPI_COMM_WORLD);
 65 : //    print_cdarray(local_cdinit);
 66 :
 67 :     /* DKA method */
 68 :     if(myrank == 0) startwtime[0] = MPI_Wtime();
 69 :     _mpi_ddka(&dtimes, cdans, local_cdans, cdinit, local_cdinit,
    df, 1000, dabs_eps, drel_eps, MPI_COMM_WORLD);
 70 :     if(myrank == 0) endwtime[0] = MPI_Wtime();
 71 :
 72 :     /* print answer */
 73 :     if(myrank == 0) printf("Iterative times: %d\n", dtimes);
 74 :     _mpi_collect_cdarray(cdans, dd_dim, local_cdans, MPI_COMM_WOR
    LD);
 75 :     if(myrank == 0) print_cdarray(cdans);
 76 :
 77 :     /* clear */
 78 :     free_dpoly(df);
 79 :     free_cdarray(cdans);
 80 :     free_cdarray(cdinit);
 81 :
 82 :     MPI_Finalize();
 83 :     if(myrank == 0)
 84 :     {
 85 :         printf("double_DKA     : %f sec\n", endwtime[0] - startwt
    ime[0]);
 86 :     }
 87 :
 88 :     return EXIT_SUCCESS;
 89 : }
```

多倍長計算の場合は次のようになる。

mpi-dka-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 : #define MAX_LENGTH 1024
```

```
12 :
13 :#define DEG 20
14 :
15 :int main(int argc, char *argv[])
16 :{
17 :    long int i, dtimes, mpftimes;
18 :    FILE * dcoef, * mpfcoef;
19 :
20 :    long int local_dim, dd_dim[MPI_GMP_MAXPROCS];
21 :    double startwtime[2], endwtime[2];
22 :    CMPFArray cmpfans, cmpfinit, local_cmpfans, local_cmpfinit;
23 :    MPFPoly mpff;
24 :    mpf_t mpfabs_eps, mpfrel_eps;
25 :    int myrank, num_procs;
26 :
27 :/* for MPI */
28 :    MPI_Init(&argc, &argv);
29 :    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
30 :    MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
31 :
32 :#define PREC 128
33 ://#define PREC 256
34 ://#define PREC 512
35 ://#define PREC 1024
36 :#define MPFDEG 20
37 ://    set_bnc_default_prec(PREC);
38 :    _mpi_set_bnc_default_prec(PREC, MPI_COMM_WORLD);
39 :    commit_mpf(&(MPI_MPF), PREC, MPI_COMM_WORLD);
40 :    commit_mpi_mpfcmplx(&(MPI_BNC_MPFCMPLX), PREC, MPI_COMM_WORLD
   );
41 :
42 :    /* init */
43 :    mpf_init_set_d(mpfabs_eps, 1.0e-100);
44 :    mpf_init_set_d(mpfrel_eps, 1.0e-7);
45 :
46 :    mpff = init_mpfpoly(MAX_LENGTH);
47 :    cmpfans = init_cmpfarray(MPFDEG);
48 :    cmpfinit = init_cmpfarray(MPFDEG);
49 :
50 :    local_dim = _mpi_divide_dim(dd_dim, MPFDEG, num_procs);
51 :    if(myrank == 0)
52 :    {
53 ://        for(i = 0; i < num_procs; i++)
54 ://            printf("%d d_dim[%d]: %d\n", local_dim, i, dd_dim[i
   ]);
55 :    }
```

```
56 :     local_cmpfans = init_cmpfarray(local_dim);
57 :     local_cmpfinit = init_cmpfarray(local_dim);
58 :
59 :     cmpfans = init_cmpfarray(MPFDEG);
60 :     cmpfinit = init_cmpfarray(MPFDEG);
61 :     if(myrank == 0)
62 :     {
63 :         mpfcoef = fopen("polycoef20.dat", "r");
64 :         fread_mpfpolycoef(mpfcoef, mpff, DEG);
65 :         print_mpfpoly(mpff);fflush(stdout);
66 :         fclose(mpfcoef);
67 :     }
68 ://    goto end;
69 :
70 :     /* Bcast mpfpoly */
71 :     _mpi_bcast_mpfpoly(mpff, MPI_COMM_WORLD);
72 :     if(myrank == 0)
73 :     {
74 :         printf("rank: %d\n", myrank);
75 :         print_mpfpoly(mpff);fflush(stdout);
76 :     }
77 :
78 :     /* set Aberth's initial value */
79 :     _mpi_mpf_dka_init(local_cmpfinit, mpff, MPI_COMM_WORLD);
80 ://    print_cmpfarray(local_cmpfinit);
81 :
82 :     /* DKA method */
83 :     if(myrank == 0) startwtime[1] = MPI_Wtime();
84 :     _mpi_mpf_dka(&mpftimes, cmpfans, local_cmpfans, cmpfinit, loc
   al_cmpfinit, mpff, 1000, mpfabs_eps, mpfrel_eps, MPI_COMM_WORLD);

85 :     if(myrank == 0) endwtime[1] = MPI_Wtime();
86 :
87 :     /* print answer */
88 :     if(myrank == 0) printf("Iterative times: %d\n", mpftimes);
89 :     _mpi_collect_cmpfarray(cmpfans, dd_dim, local_cmpfans, MPI_CO
   MM_WORLD);
90 ://    print_cmpfarray(local_cmpfans);
91 :     if(myrank == 0) print_cmpfarray(cmpfans);
92 :
93 :     /* clear */
94 :     free_mpfpoly(mpff);
95 :     free_cmpfarray(local_cmpfans);
96 :     free_cmpfarray(local_cmpfinit);
97 :     free_cmpfarray(cmpfans);
98 :     free_cmpfarray(cmpfinit);
```

```
 99 :
100 :     free_mpi_mpfcmplx(&(MPI_BNC_MPFCMPLX));
101 :     free_mpf(&(MPI_MPF));
102 :
103 : end:
104 :     MPI_Finalize();
105 :     if(myrank == 0)
106 :     {
107 :         printf("mpf_DKA(%dbits): %f sec\n", PREC, endwtime[1] - s
    tartwtime[1]);
108 :     }
109 :
110 :     return EXIT_SUCCESS;
111 :
112 : }
113 :
```

演習問題

1. (11.8) の係数を計算するプログラムを作れ。

2. 上記のプログラムを使って，

$$\prod_{i=1}^{30}(x-i)$$

   の係数を計算せよ。

3. 上記の多項式を $p(x)$ とする代数方程式の解を求めよ。この場合，全ての解が 10 桁以上の制度を持つようにするには，何桁の多倍長計算を行えばよいかも検討せよ。