Brody-Flemings model における Kobayashi の解の計算について

— 積分法での解を求めた際の比較計算に関する報告 —

姫宮 利融

デンドライト凝固に伴う溶質の再分布について、かなり古くから考えられてきたスチュエイションとして Brody-Flemings Model がある。我々は、積分法でこの問題を解くひとつの方法を開発したが、解析解として求められている Kobayashi の解との比較を行なった。Kobayashi の解は従来計算を行なうためには多大な労力を必要とすると言われていたが、今回計算してみたところ、Sun SPARC staion 20 上で、ユーザ空間で 2.0 秒の CPU TIME、SPARC station IPC 上でもユーザ空間で 16.0 秒の CPU TIME という実用上満足のいく結果を得たので、その詳細を報告する。

1 緒言：Brody-Flemings model と Kobayashi の解

Brody と Flemings[1] は、デンドライト凝固に伴う溶質の再分布について、次のような問題設定を考えた。それは、溶質として侵入型元素（典型的には Fe 中の C,P,S）を考えると、固体内での拡散を無視しえず、Scheil のモデルでは不十分であるからである。

- 1 次元で考える
- 液相は完全混合
- 固相で有限な拡散が働く
- parabolic growth を仮定する。

$$X/\lambda = (t/t_f)^{(1/2)}$$

最後の仮定は、前の 3 つに比べて本質的な仮定ではないが、デンドライトの形状を考慮すると妥当なものであり、その後の研究者もこれに従っている。なお、Fig.1 を参照されたい。

この問題は以下のように数学的に記述される。

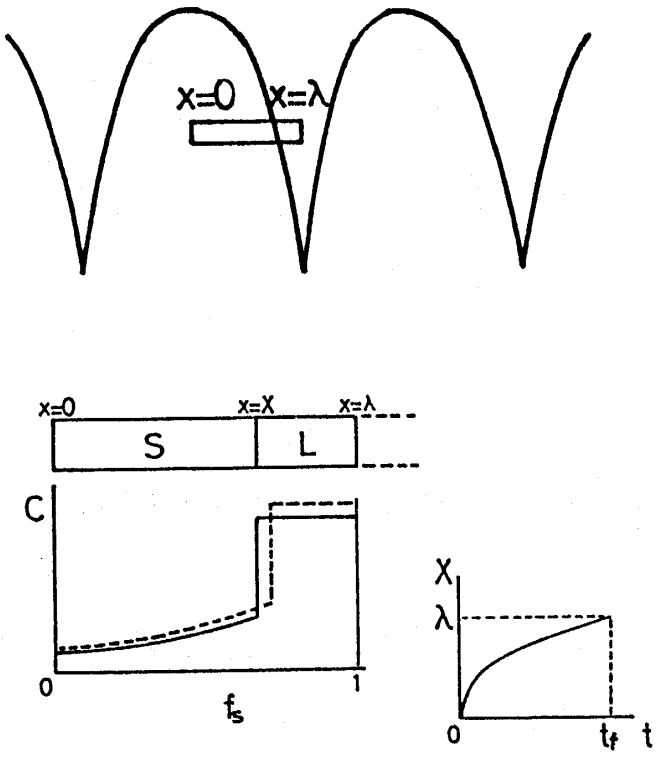

Fig.1 Brody-Flemings model

$$C_L(1-k)\frac{dX}{dt} = D_S + (\lambda - X)\frac{dC_L}{dt} \quad at\ x = X \quad (1)$$

$$C_S^* = kC_L \quad at\ x = X \quad (2)$$

$$\frac{\partial C_S}{\partial x} = 0 \quad at\ x = 0 \quad (3)$$

$$\frac{\partial C_S}{\partial t} = D_S\frac{\partial^2 C_S}{\partial x^2} \quad \text{in the solid} \quad (4)$$

本論文では、

$$\gamma \equiv \frac{D_S t_f}{\lambda^2} \quad (5)$$

という表記を使うことをお断りしておく。これは、Brody と Flemings では α と表記されていた無次元数である。

Brody と Flemings[1] は、固相内での拡散を溶質拡散層の厚みを使って近似し、

$$(C_L - C_S^*)\frac{dX}{dt} = (\lambda - X)\frac{dC_L}{dt} + \frac{\delta_S}{2}\frac{dC_S^*}{dt} \quad (6)$$

f_S と C_L の関係として

$$C_L = C_0[1 - f_S(1-2\gamma k)]^{\frac{k-1}{1-2\gamma k}} \quad (7)$$

を得たことは、古典的な業績である。

その後、Clyne-Kurz[2], Ohnaka[3] などの研究があったが、1988 年に Kobayashi[4] が解析解を提唱した。それは、

$$C_S(f_S, x^*) = kC_0\sum_{n=0}^{\infty}\zeta_n\frac{F\left(-\frac{n}{2},\frac{1}{2};-\frac{x^{*2}}{4\gamma}\right)}{F\left(-\frac{n}{2},\frac{1}{2};-\frac{1}{4\gamma}\right)}f_S^n \quad (8)$$

ここで、

$$x^* = \frac{x}{X} \quad (9)$$

$F(p,q;z)$ は Kummer の合流型超幾何関数を意味し、

$$F(p,q;z) = \sum_{r=0}^{\infty}\frac{(p)_r z^r}{(q)_r r!} \quad (10)$$

また、

$$\zeta_n = \prod_{m=0}^{n-1}\left[1 - k\frac{F\left(-\frac{m}{2},\frac{3}{2};-\frac{1}{4\gamma}\right)}{F\left(-\frac{m}{2},\frac{1}{2};-\frac{1}{4\gamma}\right)}\right] \quad (11)$$

である。

液相の溶質濃度については、

$$C_L = C_0\sum_{n=0}^{\infty}\zeta_n f_S^n \quad (12)$$

となる。

著者らは、デンドライト凝固に伴う溶質の再分布について、濃度プロフィールを単純な 2 次関数で表現し積分法を用いて常微分方程式を求めるという方法で、固相内の拡散が完全で液相で限定的な拡散が働く場合 [5]、固相と液相で共に限定的な拡散が働く場合 [6] について研究してきたが、今回、液相内が完全混合で限定的な固相内の拡散が働く場合について同じ手法を適用した [7]。

その際に、Kobayashi の解との比較を試みた。Kobayashi の解は Kummer の合流型超幾何関数を使って有限乗積 ζ_n を求め、それを使った無限級数を求めるものであり、実際に、$k = 0.14, \gamma = 0.1$ では、最終の液相濃度（言語矛盾であるが、$f_S = 1$ の時の C_L）を求めるためには、1157 項までの ζ_n を計算することを必要とした。そのため、Kobayashi の解は実用上からは、扱いにくいものとされてきた。Kobayashi 氏が、論文の Discussion で、"Further detailed analysis will need numerical methods, such as the finite difference method. In the numerical analysis, the exact solution is useful to examine the accuracy of numerical results." と述べている [4] のは、このことに考慮したものと推察する。しかしながら、１０年間の間に計算機の能力が飛躍的に向上したのか、今回 C 言語でプログラミングして計算したところ、Sun SPARC staion 20 上で、ユーザ空間で 2.0 秒の CPU TIME、SPARC station IPC 上でもユーザ空間で 16.0 秒の CPU TIME という実用上満足のいく結果を得た。もちろん、システム空間で消費した時間は共に 0.0 秒であった。したがって、今後は Kobayashi の解自体を直接溶質再分布の計算に使うことも十分実用的になると考える。そこで、計算の方法の詳細を報告する。

2 Kummer の合流型超幾何関数

Kummer の合流型超幾何関数は次式で表現される。

$$F(p,q;z)=\sum_{r=0}^{\infty}\frac{(p)_r z^r}{(q)_r r!}$$

ここで、$(p)_r$, $(q)_r$ は Pochhammer の記号であり、

$$(p)_r = p(p+1)\ldots(p+r-1),$$

$$(p)_0 = 1, \qquad etc. \qquad (13)$$

である。

無限級数の各項を直接計算しようとすると、メモリのあふれのために計算できないことが生ずる。

$$F(p,q;z)=\sum_{r=0}^{\infty}a_r \qquad (14)$$

とおくと、

$$a_0 = 1.0 \qquad (15)$$

$$a_r/a_{r-1} = (p+r-1)z/\{(q+r-1)r\} \qquad (16)$$

である。これを利用して Kummer の合流型超幾何関数を計算した。下に、テスト・プログラムのソースを示す。1/10000 の誤差を打ち切りにしてある。

Kummer_test.c

```
#include <math.h>
main()
{
        double Hg1F1(double, double, double);
        double p, q, z, Y;
        float pp, qq, zz;

        printf("INPUT p: ");
        scanf("%f", &pp);
        printf("INPUT q: ");
        scanf("%f", &qq);
        printf("INPUT z: ");
        scanf("%f", &zz);

        p = pp; q = qq; z = zz;
        Y=Hg1F1(p, q, z);
        printf("Y=%e\n", Y);
}


double Hg1F1(double p, double q, double z)
{
        double term, term_th1,term_th2, sum, product ;
        int r, m;

        term = 1.0;
        sum = 0.0;
        for(r=0; ; r++) {
```

```
                if (r == 0)
                                ;
                else {
                        product = (p+r-1) * z / ( (q+r-1) * r);
                        term = term * product;
                }
                if ( (r>20) && (fabs(term)<term_th1 || fabs(term)<term_th2) ) {
                        printf("打ち切り r=%d, term=%f\n", r, term);
                        sum += term;
                        break;
                } else {
                        sum += term;
                        term_th1 = fabs(0.00001 * term);
                        term_th2 = fabs(0.0001 * sum);
                }
        }
        return sum;
}
```

次に、プログラムの実行試験の結果を Mathematica での計算結果と対比させて示す。おおむね、良い結果が得られているが、$F(100.0, 1.5; 0.5)$ の計算結果は一致しなかった。打ち切り条件の設定がこの場合には適当でなかったのかもしれない。

```
hoku2% /usr/local/bin/math
Mathematica 2.2 for SPARC
Copyright 1988-93 Wolfram Research, Inc.
 -- SunView graphics initialized --
In[1]:= Hypergeometric1F1[0.0,1.5,0.0]

Out[1]= 1.



In[2]:= Hypergeometric1F1[-1.0,0.5,0.0]

Out[2]= 1.
```

```
sparc07% cc Kummer_test.c -lm



sparc07% a.out
INPUT p: 0.0
INPUT q: 1.5
INPUT z: 0.0
打ち切り r=21, term=0.000000
Y=1.000000e+00
sparc07% a.out
INPUT p: -1.0
INPUT q: 0.5
INPUT z: 0.0
打ち切り r=21, term=-0.000000
Y=1.000000e+00
```

```
In[3]:= Hypergeometric1F1[1000.0,1.5,0.5]

                  17
Out[3]= 3.73713 10



In[4]:= Hypergeometric1F1[10.0,1.5,0.5]

Out[4]= 11.0879



In[5]:= Hypergeometric1F1[1.0,1.5,0.5]

Out[5]= 1.41069



In[6]:= Hypergeometric1F1[-1.0,1.5,0.5]

Out[6]= 0.666667



In[7]:= Hypergeometric1F1[-100.0,1.5,0.5]

Out[7]= 0.0903469



In[8]:= Hypergeometric1F1[-1000.0,1.5,0.5]

Out[8]= 1.41819



In[10]:= Hypergeometric1F1[100.0,-1.5,-5.0]

Out[10]= -35.2663
/* no match */



In[11]:= Hypergeometric1F1[-1.0,-1.5,-5.0]

Out[11]= -2.33333
```

```
sparc07% a.out
INPUT p: 1000.0
INPUT q: 1.5
INPUT z: 0.5
打ち切り r=36, term=286539977
90635.238281
Y=3.736965e+17
sparc07% a.out
INPUT p: 10.0
INPUT q: 1.5
INPUT z: 0.5
打ち切り r=21, term=0.000000
Y=1.108791e+01
sparc07% a.out
INPUT p: 1.0
INPUT q: 1.5
INPUT z: 0.5
打ち切り r=21, term=0.000000
Y=1.410686e+00
sparc07% a.out
INPUT p: -1.0
INPUT q: 1.5
INPUT z: 0.5
打ち切り r=21, term=-0.000000
Y=6.666667e-01
sparc07% a.out
INPUT p: -100.0
INPUT q: 1.5
INPUT z: 0.5
打ち切り r=21, term=-0.000004
Y=9.034664e-02
sparc07% a.out
INPUT p: -1000.0
INPUT q: 1.5
INPUT z: 0.5
打ち切り r=61, term=-0.000029
Y=1.418190e+00
sparc07% a.out
INPUT p: 100.0
INPUT q: -1.5
INPUT z: -5.0
打ち切り r=72, term=7.604174
Y=1.275003e+05
/* no match */
sparc07% a.out
INPUT p: -1.0
INPUT q: -1.5
INPUT z: -5.0
打ち切り r=21, term=0.000000
Y=-2.333333e+00
```

```
In[12]:= Hypergeometric1F1[10.0, -1.5, -5.0]

Out[12]= 2.54994



In[13]:= Hypergeometric1F1[-10.0,-1.5,-5.0]

                    6
Out[13]= 2.09536 10



In[14]:= Hypergeometric1F1[-500.0,-1.5,-5.0]

                   45
Out[14]= 3.4438 10
```

```
sparc07% a.out
INPUT p: 10.0
INPUT q: -1.5
INPUT z: -5.0
打ち切り r=37, term=-0.000109
Y=2.549925e+00
sparc07% a.out
INPUT p: -10.0
INPUT q: -1.5
INPUT z: -5.0
打ち切り r=21, term=-0.000000
Y=2.095359e+06
sparc07% a.out
INPUT p: -500.0
INPUT q: -1.5
INPUT z: -5.0
打ち切り r=67, term=283634583
05094377383958208658604500923
1872.000000
Y=3.443543e+45
```

3 Kobayashi の解を計算するプログラム

main 関数の最初で、配列の形式で ζ_n を必要な項数だけ予め計算しておく。これは、C_L, C_S の計算の度毎に ζ_n が呼び出されるので何度も繰り返して同じ計算をするのを避けるためである。

ζ_n が必要な精度で C_L, C_S を計算するのに何項必要かは $f_S = 1$ のときの C_L の計算で必要な項数で決まる。それは、

$$C_L = C_0 \sum_{n=0}^{\infty} \zeta_n f_S^n \qquad when \;\; 0 < f_S < 1$$

であるが、$f_S = 1$ のとき、

$$C_L = C_0 \sum_{n=0}^{\infty} \zeta_n \tag{17}$$

であって、級数を計算して C_L がどれだけの精度で求まるかは、専ら $n \to \infty$ のとき ζ_n がどれだけの速さで 0 に収束するかに係っているからである。下に掲げるプログラムの例では 2000 項を予め計算することにしているが、これは標準エラー出力に出力される C_L の計算での打ち切りを見ながら適当に変えていけばよい。

なお、$\zeta_n = \zeta_n(k, \gamma)$ であって、

$$\zeta_0 = 1,$$

$$\zeta_n/\zeta_{n-1} = 1 - k\frac{F\left(-\frac{n-1}{2}, \frac{3}{2}; -\frac{1}{4\gamma}\right)}{F\left(-\frac{n-1}{2}, \frac{1}{2}; -\frac{1}{4\gamma}\right)} \tag{18}$$

であることを利用する。

main 関数の中では、f_S を変化させて、関数 C_L(), C_S() を呼び出して液相の溶質濃度と、固相内での溶質分布を計算する。C_L での「打ち切り 1」は正常な計算の打ち切り、「打ち切り 2」は不正常な計算の打ち切りである。このメッセージは標準エラー出力に出力される。 C_S(), Hg1F1() での打ち切りに関する出力はコメントアウトしてある。

kobayashi.c

```
/* Kobayashi's equation --- to calculate solute redistribution with     */
/* dendritic solidification accompanied with some diffusion in solid    */
/* and no diffusion in liquid                                           */

#include<stdio.h>
#include<math.h>
double Zeta[10000];

main()
{
        int i;
        float K, G;
        double k, g, fs, Cl, Cs, xx, x;
        double C_L(double, double, double);
        double C_S(double, double, double, double);
        double zeta_term(int, double, double);

        fprintf(stderr, "Distribution Coefficient : ");
                scanf("%f", &K);
        fprintf(stderr, "Parameter of Diffusion in Solid : ");
                scanf("%f", &G);

        k = K;
        g = 2 * G;
        printf("\n\n\n");
        printf("k = %f\tgamma = %f\n\n\n", k, G);

        Zeta[0] = 1.0;
        for (i=1; i<2000; i++)
                Zeta[i] = Zeta[i-1] * zeta_term(i-1, k, g);
        fprintf(stderr, "Cal Zeta done\n");

        fs = 0.01;
                Cl = C_L(fs, k, g);
                printf("\nfs = %f \t", fs);
                printf("C_L = %e \n", Cl);

                for (i = 0; i<=10; i++) {
                        xx = 0.1 * i;
                        Cs = C_S(fs, xx, k, g);
                        x = xx * fs;
                        printf("\tx = %f\t",x);
                        printf("C_S = %e\n", Cs);
                }
```

```
for(fs = 0.05; fs < 0.95001; fs += 0.05 ) {
        Cl = C_L(fs, k, g);
        printf("\nfs = %f \t", fs);
        printf("C_L = %e \n", Cl);


        if(fs <= 0.5) {
                for (i = 0; i<=10; i++) {
                        xx = 0.1 * i;
                        Cs = C_S(fs, xx, k, g);
                        x = xx * fs;
                        printf("\tx = %f\t",x);
                        printf("C_S = %e\n", Cs);
                }
        } else {
                for (i = 0; i<=20; i++) {
                        xx = 0.05 * i;
                        Cs = C_S(fs, xx, k, g);
                        x = xx * fs;
                        printf("\tx = %f\t",x);
                        printf("C_S = %e\n", Cs);
                }
        }
        }

        fs = 0.98;
        Cl = C_L(fs, k, g);
        printf("\nfs = %f \t", fs);
        printf("C_L = %e \n", Cl);
                for (i = 0; i<=20; i++) {
                        xx = 0.05 * i;
                        Cs = C_S(fs, xx, k, g);
                        x = xx * fs;
                        printf("\tx = %f\t",x);
                        printf("C_S = %e\n", Cs);
                }

        fs = 0.99;
        Cl = C_L(fs, k, g);
        printf("\nfs = %f \t", fs);
        printf("C_L = %e \n", Cl);
                for (i = 0; i<=20; i++) {
                        xx = 0.05 * i;
```

```
                        Cs = C_S(fs, xx, k, g);
                        x = xx * fs;
                        printf("\tx = %f\t",x);
                        printf("C_S = %e\n", Cs);
                }

        fs = 1.0;
        Cl = C_L(fs, k, g);
        printf("\nfs = %f \t", fs);
        printf("C_L = %e \n", Cl);
                for (i = 0; i<20; i++) {
                        xx = 0.05 * i;
                        Cs = C_S(fs, xx, k, g);
                        x = xx * fs;
                        printf("\tx = %f\t",x);
                        printf("C_S = %e\n", Cs);
                }
}


double C_L(double fs, double k, double g)
/* to calculate solute composition in the liquid *?
{
        static double term, term2, sum, sum_, term_th1, term_th2;
        static int n;


        sum = 0.0;
        for(n=0;; n++) {
                term = Zeta[n] * pow(fs, n);
                if ( (n < 2000 && n>100) &&
                                (term < term_th1 || term <term_th2 ) ) {
                        fprintf(stderr,
                                "in C_L() 打ち切り1 : fs=%f, n=%d\n", fs, n);
                        sum += term;
                        break;
                }
                else if ( (n>1900) && (fabs(term) > 0.95 * fabs(term2) ) ) {
                        fprintf(stderr,
                                "in C_L() 打ち切り2 : fs=%f, n=%d\n", fs, n);
                        fprintf(stderr,"\t%e\n", term);
                        sum += term;
                        sum_ = sum + term / ( 1.0 - term / term2) ;
                        fprintf(stderr, "fs = %f, sum_upper = %e\n", fs, sum_);
                        break;
```

```
                }
                else {
                        sum += term;
                        term2 = term;
                        term_th1 = term * 0.001;
                        term_th2 = sum * 0.0001;
                }
        }
        return sum;
}

double C_S(double fs, double x_, double k, double g)
/* to calculate solute distribution in the solid */
{
        static double term1,term2, sum, term_th1, term_th2;
        static double nn, gg, xg;
        static int n;
        double Hg1F1(double, double, double);

        sum = 0.0;
        for(n=0;; n++) {
                nn = -n/2;
                xg = - x_ * x_ / (2 * g);
                gg = -1/(2 * g);
                term1 = Zeta[n] * pow(fs, n);
                term2 = term1 * Hg1F1(nn, 0.5, xg) / Hg1F1(nn, 0.5, gg);
                if ( (n>40) && (term2 < term_th1 || term_th2) ) {
          /*      printf("in C_S() 打ち切り 1 : fs=%f, n=%d\n", fs, n); */
                        sum += term2;
                        break;
                }
                else {
                        sum += term2;
                        term_th1 = term2 * 0.001;
                        term_th2 = sum * 0.0001;
                }
        }
        return k*sum;
}


double zeta_term(int n, double k, double g)
{
        static int m;
        static double term;
```

```
        static double mm, gg;
        double Hg1F1(double, double, double);

                mm = -n/2;
                gg = -1/(2 * g);
                term = 1 - k * Hg1F1(mm, 1.5, gg) / Hg1F1(mm, 0.5, gg);
                return term;
}


double Hg1F1(double p, double q, double z)
{
        double term, term_th1,term_th2, sum, product ;
        int r, m;

        term = 1.0;
        sum = 0.0;
        for(r=0; ; r++) {
                if (r == 0)
                                ;
                else {
                        product = (p+r-1) * z / ( (q+r-1) * r);
                        term = term * product;
                }
                if ( (r>20) && (fabs(term)<term_th1 || fabs(term)<term_th2) ) {
         /*             printf("打ち切り r=%d, term=%f\n", r, term); */
                        sum += term;
                        break;
                } else {
                        sum += term;
                        term_th1 = fabs(0.00001 * term);
                        term_th2 = fabs(0.0001 * sum);
                }
        }
        return sum;
}
```

4 kobayashi.c の計算結果

sparc07% cc kobayashi.c -lm とコンパイルして
sparc07% time a.out >res0.1 とすると、次のような標準エラー出力が表示される。

```
Distribution Coefficient : 0.14
Parameter of Diffusion in Solid : 0.1
Cal Zeta done
in C_L() 打ち切り 1 : fs=0.010000, n=101
in C_L() 打ち切り 1 : fs=0.010000, n=101
in C_L() 打ち切り 1 : fs=0.050000, n=101
in C_L() 打ち切り 1 : fs=0.100000, n=101
in C_L() 打ち切り 1 : fs=0.150000, n=101
in C_L() 打ち切り 1 : fs=0.200000, n=101
in C_L() 打ち切り 1 : fs=0.250000, n=101
in C_L() 打ち切り 1 : fs=0.300000, n=101
in C_L() 打ち切り 1 : fs=0.350000, n=101
in C_L() 打ち切り 1 : fs=0.400000, n=101
in C_L() 打ち切り 1 : fs=0.450000, n=101
in C_L() 打ち切り 1 : fs=0.500000, n=101
in C_L() 打ち切り 1 : fs=0.550000, n=101
in C_L() 打ち切り 1 : fs=0.600000, n=101
in C_L() 打ち切り 1 : fs=0.650000, n=101
in C_L() 打ち切り 1 : fs=0.700000, n=101
in C_L() 打ち切り 1 : fs=0.750000, n=101
in C_L() 打ち切り 1 : fs=0.800000, n=101
in C_L() 打ち切り 1 : fs=0.850000, n=101
in C_L() 打ち切り 1 : fs=0.900000, n=101
in C_L() 打ち切り 1 : fs=0.950000, n=102
in C_L() 打ち切り 1 : fs=0.980000, n=200
in C_L() 打ち切り 1 : fs=0.990000, n=314
in C_L() 打ち切り 1 : fs=1.000000, n=1157
2.0u 0.0s 0:08 23% 0+0k 0+0io 0pf+0w
```

この例では、データの入力は標準入力から行なっているが、データ・ファイルを別に作り、リダイレクションを使ってもよい。プログラムでは "Cal Zeta done\n"の箇所に '^G' (BELL) が記入されていて、ζ_n の配列の計算が終了するとベル音と共に、メッセージが表示されるようにしてある。計算時間のほとんどを ζ_n の配列の計算に費やしているようである。

打ち切りのメッセージは C_L の計算を行なうのに、ζ_n の配列を何項まで必要としたかを表示している。この例では、$f_S = 0.9$ までは 101 項（打ち切りの最低限）であるが、$f_S = 0.95$ では 102 項、$f_S = 0.98$ では 200 項、$f_S = 0.99$ では 314 項、$f_S = 1$ では 1157 項を必要としたことを示している。プログラムにあるように、「打ち切り 1」は正常な打ち切り（収束した）を示し、「打ち切り 2」は不正常な打ち切りを示す。「打ち切り 2」となったときには、ζ_n の配列をもっと大きな項まで計算して、打ち切りまでに使う項数を大きくしてやる必要がある。

次に、出力結果（ファイル **res0.1**）の内容を途中を省略して紹介しておく。

```
k = 0.140000 gamma = 0.100000


fs = 0.010000  C_L = 1.008675e+00
x = 0.000000 C_S = 1.412057e-01
x = 0.001000 C_S = 1.412058e-01
x = 0.002000 C_S = 1.412061e-01
x = 0.003000 C_S = 1.412065e-01
x = 0.004000 C_S = 1.412071e-01
x = 0.005000 C_S = 1.412079e-01
x = 0.006000 C_S = 1.412089e-01
x = 0.007000 C_S = 1.412100e-01
x = 0.008000 C_S = 1.412113e-01
x = 0.009000 C_S = 1.412128e-01
x = 0.010000 C_S = 1.412145e-01

fs = 0.050000  C_L = 1.044940e+00
x = 0.000000 C_S = 1.460652e-01
x = 0.005000 C_S = 1.460675e-01
x = 0.010000 C_S = 1.460742e-01
--- More ---
fs = 0.950000  C_L = 1.030634e+01
x = 0.000000 C_S = 2.951320e-01
x = 0.047500 C_S = 2.958712e-01
x = 0.095000 C_S = 2.981081e-01
x = 0.142500 C_S = 3.019026e-01
x = 0.190000 C_S = 3.073581e-01
x = 0.237500 C_S = 3.146286e-01
x = 0.285000 C_S = 3.239279e-01
x = 0.332500 C_S = 3.355448e-01
x = 0.380000 C_S = 3.498632e-01
x = 0.427500 C_S = 3.673921e-01
x = 0.475000 C_S = 3.888071e-01
x = 0.522500 C_S = 4.150106e-01
x = 0.570000 C_S = 4.472189e-01
```

```
x = 0.617500 C_S = 4.870877e-01
x = 0.665000 C_S = 5.368989e-01
x = 0.712500 C_S = 5.998359e-01
x = 0.760000 C_S = 6.803973e-01
x = 0.807500 C_S = 7.850215e-01
x = 0.855000 C_S = 9.230372e-01
x = 0.902500 C_S = 1.108121e+00
x = 0.950000 C_S = 1.360541e+00

fs = 0.980000  C_L = 1.878514e+01
x = 0.000000 C_S = 3.033307e-01
x = 0.049000 C_S = 3.041875e-01
x = 0.098000 C_S = 3.067842e-01
x = 0.147000 C_S = 3.112018e-01
x = 0.196000 C_S = 3.175817e-01
x = 0.245000 C_S = 3.261362e-01
x = 0.294000 C_S = 3.371653e-01
x = 0.343000 C_S = 3.510807e-01
x = 0.392000 C_S = 3.684431e-01
x = 0.441000 C_S = 3.900141e-01
x = 0.490000 C_S = 4.168351e-01
x = 0.539000 C_S = 4.503421e-01
x = 0.588000 C_S = 4.925381e-01
x = 0.637000 C_S = 5.462529e-01
x = 0.686000 C_S = 6.155375e-01
x = 0.735000 C_S = 7.062703e-01
x = 0.784000 C_S = 8.270924e-01
x = 0.833000 C_S = 9.908644e-01
x = 0.882000 C_S = 1.216946e+00
x = 0.931000 C_S = 1.534781e+00
x = 0.980000 C_S = 1.989560e+00

fs = 0.990000  C_L = 2.778760e+01
x = 0.000000 C_S = 3.061844e-01
x = 0.049500 C_S = 3.070858e-01
x = 0.099000 C_S = 3.098193e-01
x = 0.148500 C_S = 3.144750e-01
x = 0.198000 C_S = 3.212105e-01
x = 0.247500 C_S = 3.302637e-01
x = 0.297000 C_S = 3.419723e-01
x = 0.346500 C_S = 3.568038e-01
x = 0.396000 C_S = 3.753996e-01
x = 0.445500 C_S = 3.986400e-01
x = 0.495000 C_S = 4.277417e-01
x = 0.544500 C_S = 4.644025e-01
x = 0.594000 C_S = 5.110206e-01
x = 0.643500 C_S = 5.710287e-01
x = 0.693000 C_S = 6.494077e-01
x = 0.742500 C_S = 7.534816e-01
x = 0.792000 C_S = 8.941598e-01
x = 0.841500 C_S = 1.087886e+00
x = 0.891000 C_S = 1.359714e+00
x = 0.940500 C_S = 1.748186e+00
x = 0.990000 C_S = 2.313090e+00

fs = 1.000000  C_L = 7.691582e+01
x = 0.000000 C_S = 3.091048e-01
x = 0.050000 C_S = 3.100540e-01
x = 0.100000 C_S = 3.129344e-01
x = 0.150000 C_S = 3.178463e-01
x = 0.200000 C_S = 3.249661e-01
x = 0.250000 C_S = 3.345612e-01
x = 0.300000 C_S = 3.470134e-01
x = 0.350000 C_S = 3.628557e-01
x = 0.400000 C_S = 3.828258e-01
x = 0.450000 C_S = 4.079471e-01
x = 0.500000 C_S = 4.396502e-01
x = 0.550000 C_S = 4.799563e-01
x = 0.600000 C_S = 5.317583e-01
x = 0.650000 C_S = 5.992525e-01
x = 0.700000 C_S = 6.886102e-01
x = 0.750000 C_S = 8.090281e-01
x = 0.800000 C_S = 9.743822e-01
x = 0.850000 C_S = 1.205847e+00
x = 0.900000 C_S = 1.536065e+00
x = 0.950000 C_S = 2.015794e+00
```

これらの計算結果を纏めたものの例を、論文末の図 (**Fig.2 − 5**) に示す。**Fig.2** は、溶質の分配係数 $k = 0.14$ で、固相内での拡散の度合いを示す無次元パラメータ $\gamma = 0.5$ の時の固相率の変化（凝固の進行）に伴う液相内の溶質濃度の変化を Scheil, Himemiya, Ohnaka, Clyne-Kurz, Brody-Flemings, lever rule による計算結果と比較して示した。**Fig.3** は $\gamma = 0.5$ のとき、凝固の進行に伴って固相内の溶質分布の変化する様子を示した。**Fig.4, 5** は、$k = 0.14, \gamma = 5.0$ での **Fig.2, Fig3** に対応する図である。なお、

$k = 0.14$ という数値は Al-Cu 系 2 元合金での Al 側での Cu の分配係数であり、計算のための数値としては仮想的なものであるが、[5],[6] との比較のためにこの数値で計算をしてみた。

5 計算が高速で行なわれるようになった理由と、プログラムの有効性

計算結果のところで見たように、ユーザ空間で 2.0 秒という CPU 時間は十分実用に耐えるものだと考える。しかも、プログラムを見ると想像できるようにその時間のほとんどは ζ_n の配列を計算するのに使われ、かつ ζ_n は漸化式を利用して計算するので必要な ζ_n の項数が増えていっても計算時間は項数に比例してしか増大しない。従って、たとえ、10000 項の ζ_n が必要とされるとしても、CPU は 1 分以内であろう。（$k = 0.14$, $\gamma = 0.1$ で ζ_n を 10000 項まで最初に計算してやってみたところ、ユーザ空間で 8.0 秒 — Sun SPARC station 20 上での値 — という結果を得た。）

当初、計算が困難であると見做していた Kobayashi の解がこのように高速で計算できるようになった理由はなんといってもこの間の計算機の CPU パワーが飛躍的に向上したことが挙げられる。つぎに、kobayashi.c に関して言えば、メモリの心配をしないで ζ_n の配列を確保して使ったこと、ζ_n と $F(p,q;z)$ の計算に漸化式を利用したことが挙げられる。

Kobayashi の解は解析的な導出の過程を理解するのが難しいが、今後、違いにこだわるとしたら数値的な計算法だけでなくそれ自身がもっと活用されるであろう。

プログラムの利用について

プログラム **Kummer_test.c,kobayashi.c** は自由に研究・教育にご活用ください。ただし、著者は、本論文で報告した以上の bug が潜んでいたとしてもそのことに責任は負いません。bug を発見してそれを潰した場合はご一報下さい。また、これらのプログラムを活用して研究を行ない、その成果を公表なさる場合には、プログラム **kobayashi.c** は文献 [4] に基づくものであり、解の導出は Sumio Kobayashi 氏に負うものであること、2 つのプログラムの原著者は、姫宮であることを明示して下さい

文献

[1] H.D.Brody and M.C.Flemings, *Transactions of the Metallugical Society of AIME*, **236** (1966) 615

[2] T.W.Clyne and W.Kurz, *Met.Trans.*, **12A** (1981) 965

[3] I.Ohnaka, *Trans. ISIJ*, **26** (1986) 1045

[4] S.Kobayashi, *J.Cryst.Growth*, **88** (1988) 87

[5] T.Umeda, T.Himemiya and W.Kurz, in *"Computer Aided Inovation of New Materials II"*, p.1719, ed. by M.Doyama, J.Kihara, M.Tanaka and R.Yamamoto, published by North-Holland Co., 1993.

[6] 姫宮利融, 梅田高照 「日本金属学会春期大会講演概要 (1994)」 No.(363), (p.157)

[7] 姫宮利融, 梅田高照 「日本鉄鋼協会北海道支部・日本金属学会北海道支部両支部合同夏期講演大会概要集 (1995)」 p.9 (A04)

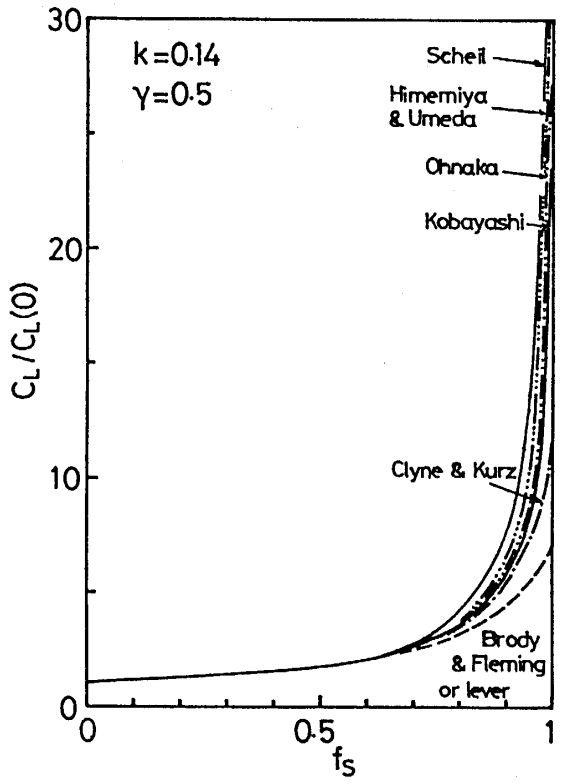

Fig.2 Solute composition in the liquid as solidification proceeds ($k = 0.14$ and $\gamma = 0.5$) .

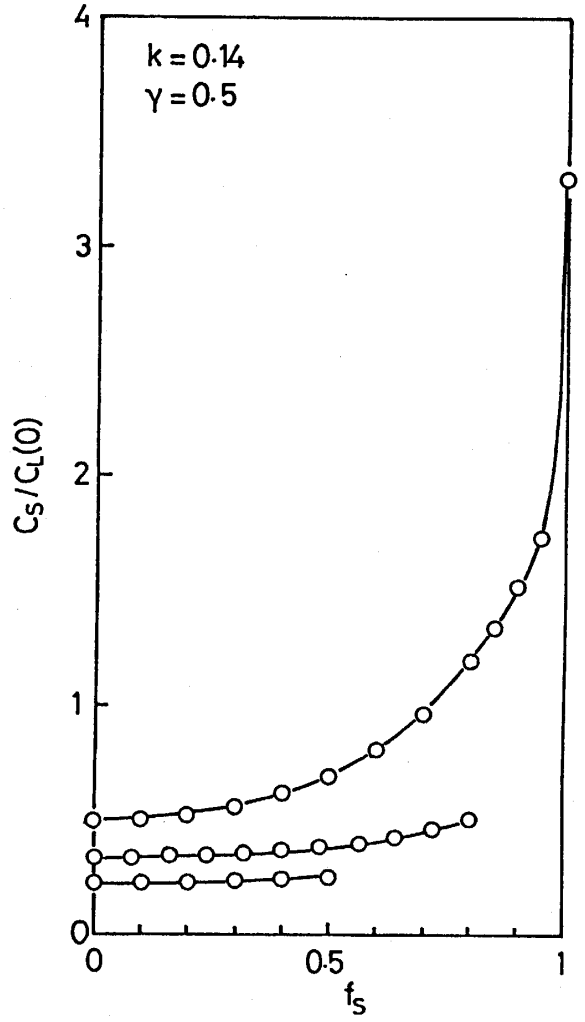

Fig.3 Solute distribution profile in the solid as solidification proceeds ($k = 0.14$ and $\gamma = 0.5$) .

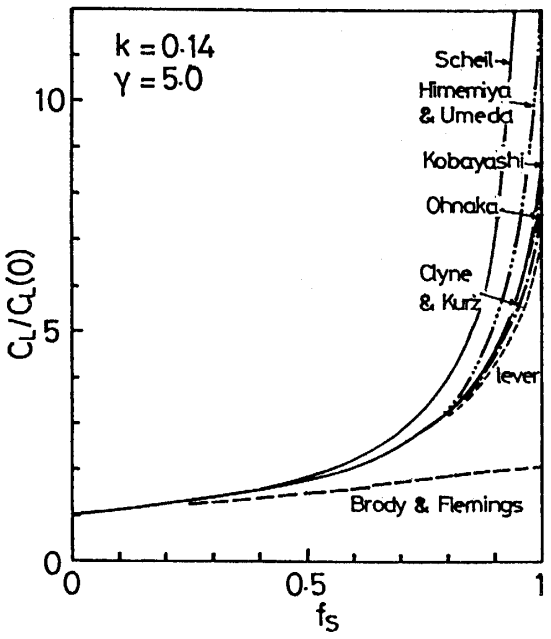

Fig.4 Solute composition in the liquid as solidification proceeds ($k = 0.14$ and $\gamma = 5.0$) .

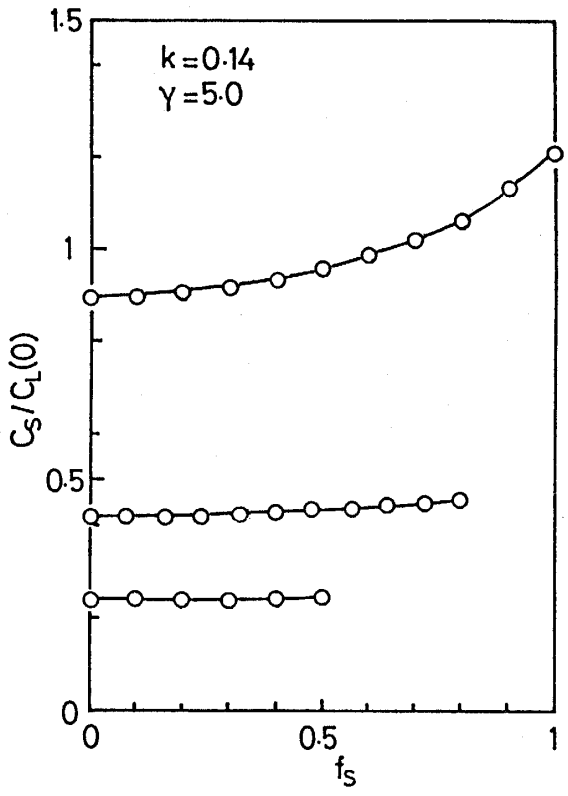

Fig.5 Solute distribution profile in the solid as solidification proceeds ($k = 0.14$ and $\gamma = 5.0$) .