Javaによる非線形拡散方程式（解離機構．kick-out機構）の数値解

Javaによる非線形拡散方程式（解離機構・kick-out機構）の数値解

師　岡　正　美　(工学部電気工学科)

Java Programming for Numerical Solution of Nonlinear Diffusion Equation (Dissociative and Kick-out Mechanisms)

Masami MOROOKA (Department of Electrical Engineering)

Abstract

A Java Programming to solve nonlinear diffusion equation by Crank-Nicolson's implicit and Guss-Seidel's iteration methods is presented. Dependences of the time increment, the criterion in the iteration, and the initial value on the numerical solution have been investigated using the program. The time increment to constrict the calculation decreases with the decrease of the criterion and the initial value. The criterion and the initial values are enough to be used a value less than 10^{-8} and 10^{-6}, respectively, in our case.

In the practice of Au diffusion in Si, which is affected by dissociative and kick-out mechanisms, useful in-diffusion and out-diffusion profiles are obtained within one minute of computing time even in use of an old type Mac computer.

Keywords; *Java programming, nonlinear diffusion equation, Au diffusion in Si*.

1．はじめに

筆者は十数年に渡って，Si中のAu拡散の研究を行っている。Si中でAu原子は格子間位置と置換位置を占め，拡散の遅い置換Au原子濃度が，拡散の速い格子間Au原子濃度より極めて大きい。即ち，Si中Au拡散に於いては，格子間不純物原子と置換不純物原子の位置交換が不純物拡散に支配的影響を与え，この位置交換に空孔（解離機構）や格子間Si原子（kick-out機構）の固有点欠陥が関与して，複雑な非線形拡散を行う。

この非線形拡散方程式の解は，従来高速コンピュータを用いFortranで数値解として求めていたが，今回，最近注目されているJavaを使用したところ簡単にパソコンで良い結果が得られたので報告する。なお，パソコンはPower Mac 9600/300, Javaプログラミングには Code Warriorを，図作成ソフトとして Kaleida Graphを用いた。

平成14年2月4日受付

2．Si中Au拡散方程式

状況によって濃度の大小はあるが，通常，Si中では空孔と格子間Si原子の両方が存在する[1]と考えたが良い。空孔が存在すると，表面から拡散してきた格子間Au原子は空孔に入って置換金となる。この時の反応は次式で表される。

表面Au層　　表面
　↓拡散D_i　↓拡散D_v
格子間Au(C_i)＋空孔(C_v) → 置換Au(C_s)．（2．1）
　↑拡散D_v 反応K_v
　拡張欠陥

また，格子間Au原子はSi原子を格子間位置に蹴飛ばし置換Auとなり，この時の反応は次式で表される。

表面Au層　　表面
　↓拡散D_i　↑拡散D_I
格子間Au(C_i)＋S_i(C_0)→格子間S_i(C_I)＋置換Au(C_s)．
　反応K_I　↓拡散D_I
　拡張欠陥

（2．2）

さらに格子間Si原子と空孔が結合して格子位置Si原子になったり，その逆過程でそれぞれが解離発生して次式が得られる。

$$S_i(C_0) \underset{K_{Fl}}{\overset{K_{Fr}}{\rightleftarrows}} \text{空孔}(C_v) + \text{格子間}S_i(C_I). \qquad (2．3)$$

ここで，C，D，Kはそれぞれ，濃度，拡散係数，反応定数を表す。各反応過程が局所熱平衡を満足し，さらに格子間金が熱平衡濃度C_{i0}としたとき，即ち金拡散が空孔・格子間Si原子の拡散で律速されるとき，置換金に対する拡散方程式として次式[2]が得られている。

$$\frac{\partial C}{\partial \tau}-a\bullet\frac{\partial}{\partial \tau}\left(\frac{1}{C}\right)=b\bullet\frac{\partial^2 C}{\partial \xi^2}+c\bullet(1-C)-d\bullet\frac{\partial^2}{\partial \xi^2}\left(\frac{1}{C}\right)-e\bullet\left(1-\frac{1}{C}\right). \qquad (2．4)$$

ここで，C_sは熱平衡濃度C_{s0}で規格化し($C=C_s/C_{s0}$)，表面からの距離xは試料厚Lで規格化し($\xi=x/L$)，時間tは実効拡散時間で規格化した$\left(\tau=t\,(D_v+D_I)/L^2\right)$値を用い，さらに空孔・格子間Si原子の拡張欠陥への拡散を空間的に平均化している。さらに，a, b, c, dは次式で与えられる定数である。

$$a=\frac{C_{I0}}{C_{s0}+C_{v0}} \qquad (2．5)$$

$$b=\frac{D_v C_{v0}}{(D_v+D_I)(C_{s0}+C_{v0})} \qquad (2．6)$$

$$c=\frac{K_{vd} C_{v0} L^2}{(D_v+D_I)(C_{s0}+C_{v0})} \qquad (2．7)$$

$$d=\frac{D_I C_{I0}}{(D_v+D_I)(C_{s0}+C_{v0})} \qquad (2．8)$$

$$e=\frac{K_{Id} C_{I0} L^2}{(D_v+D_I)(C_{s0}+C_{v0})} \qquad (2．9)$$

ここで，C_{v0}，C_{I0}はそれぞれC_v，C_Iの熱平衡濃度，K_{vd}，K_{Id}はそれぞれ空孔，格子間Si原子の拡張欠陥での生成消滅の反応定数である。

3．偏微分方程式の数値解法

微分方程式の数値解法については多くの文献[3]に詳しく述べられている。拡散方程式は時間tと距離(x, y, z)を変数とした偏微分方程式で通常表される。一般的に，偏微分方程式は差分方程式に書き換え，[4]ある誤差の範囲での近似数値解を求める。ここでは，（2．4）式に見られる非線形一次元拡散を取り扱う。

数値計算を行うために，導関数を差分近似する。連続関数$f(u)$において，変数uが微少量Δuだけ変化した時の関数の値はTaylorの定理より次式で与えられる。

$$f(u+\Delta u)=f(u)+(\Delta u)f'(u)+\frac{1}{2}(\Delta u)^2 f''(u)+\frac{1}{6}(\Delta u)^3 f'''(u)+\bullet\bullet\bullet, \qquad (3．1)$$

$$f(u-\Delta u)=f(u)-(\Delta u)f'(u)+\frac{1}{2}(\Delta u)^2 f''(u)-\frac{1}{6}(\Delta u)^3 f'''(u)+\bullet\bullet\bullet, \qquad (3．2)$$

ここで，$f^{(n)}(u)$はn次の導関数である。2式の和または差をとり，それぞれ$(\Delta u)^4$または$(\Delta u)^3$以降の項を無視すると，2次および1次の導関数は下記の差分式で近似できる。

$$f''(u)=\frac{1}{(\Delta u)^2}\{f(u+\Delta u)-2f(u)+f(u-\Delta u)\}, \qquad (3．3)$$

$$f'(u)=\frac{1}{2(\Delta u)}\{f(u+\Delta u)-f(u-\Delta u)\}. \qquad (3．4)$$

（3．4）式を中心差分近似と呼び，1次の導関数は次式で近似する場合もある。

$$f'(u)=\frac{1}{\Delta u}\{f(u+\Delta u)-f(u)\},$$
（前進差分近似）（3．5）

$$f'(u)=\frac{1}{\Delta u}\{f(u)-f(u-\Delta u)\}.$$
（後退差分近似）（3．6）

2変数関数$f(x, t)$について微分方程式を解く場合には，それぞれの変数の刻みをh，kとして$x=ih$，$(i=0, 1, \cdots, n-1)$，$t=jk$，$(j=0, 1, \cdots, m-1)$で表わす。$f(x, t)=f(ih, jk)=f_{ij}$，$f(x+h, t+k)=f((i+1)h, (j+1)k)=f_{i+1, j+1}$，$f(x-h, t)=f((i-1)h, jk)=f_{i-1, j}$の様に略記する。

陽解法は，各偏微分を下記のように差分化する。

$$\frac{\partial^2 f}{\partial x^2}=\frac{1}{h^2}(f_{i+1, j}-2f_{i, j}+f_{i-1, j}), \qquad (3．7)$$

$$\frac{\partial f}{\partial x}=\frac{1}{2h}(f_{i+1,j}-f_{i-1,j}),\quad\text{（中央差分近似）（3．8）}$$

$$\frac{\partial f}{\partial t}=\frac{1}{k}(f_{i,j+1}-f_{i,j}).\quad\text{（前進差分近似）（3．9）}$$

即ち，まずある時刻tにおける分布を求め，次に$t+k$の分布を求めるので，時間の導関数は前進差分近似を用いる。陽解法では，解が安定するには$k/h^2<1/2$の条件があり，時間刻みが制限されるために，拡散の遅い現象では実験結果を解析するには計算時間が長くなる。陽解法ではxの導関数として$t=jk$での値を使用したが，陰解法では$t=(j+1)k$の寄与を考慮し，

$$\frac{\partial^2 f}{\partial x^2}=\frac{1}{h^2}\{(1-\lambda)(f_{i+1,j}-2f_{i,j}+f_{i-1,j})+\lambda(f_{i+1,j+1}-2f_{i,j+1}+f_{i-1,j+1})\},\quad(3.10)$$

$$\frac{\partial f}{\partial x}=\frac{1}{2h}\{(1-\lambda)(f_{i+1,j}-f_{i-1,j})+\lambda(f_{i+1,j+1}-f_{i-1,j+1})\},\quad(3.11)$$

を用いる。ここで，λは$t=j(k+1)$における寄与の割合で，$\lambda=0.5$のときCrank-Nicolsonの陰解法である。陰解法ではk/h^2の値によらず解は安定である。ここでは，計算時間が節約できる陰解法を用いる。

4．非線形拡散方程式（(2.4)式）の数値解

（3．9），（3.10）式より

$$\frac{\partial C}{\partial\tau}=\frac{1}{k}(C_{i,j+1}-C_{i,j}),\quad(4.1)$$

$$\frac{\partial}{\partial\tau}\left(\frac{1}{C}\right)=\frac{1}{k}\left(\frac{1}{C_{i,j+1}}-\frac{1}{C_{i,j}}\right),\quad(4.2)$$

$$\frac{\partial^2 C}{\partial\xi^2}=\frac{1}{h^2}\{(1-\lambda)(C_{i+1,j}-2C_{i,j}+C_{i-1,j})+\lambda(C_{i+1,j+1}-2C_{i,j+1}+C_{i-1,j+1})\},\quad(4.3)$$

$$\frac{\partial^2}{\partial\xi^2}\left(\frac{1}{C}\right)=\frac{1}{h^2}\left\{(1-\lambda)\left(\frac{1}{C_{i+1,j}}-\frac{2}{C_{i,j}}+\frac{1}{C_{i-1,j}}\right)+\lambda\left(\frac{1}{C_{i+1,j+1}}-\frac{2}{C_{i,j+1}}+\frac{1}{C_{i-1,j+1}}\right)\right\}.\quad(4.4)$$

（4．1）－（4．4）式を（2．4）式に代入し，$C_{i,j+1}$について整理すると次式が得られる。

$$\left(1+2b\lambda\cdot\frac{k}{h^2}\right)C_{i,j+1}-\left(a+2d\lambda\cdot\frac{k}{h^2}\right)\frac{1}{C_{i,j+1}}=b\lambda\cdot\frac{k}{h^2}\cdot C_{i+1,j+1}-d\lambda\cdot\frac{k}{h^2}\cdot\frac{1}{C_{i+1,j+1}}+b(1-\lambda)\cdot\frac{k}{h^2}\cdot C_{i+1,j}-d(1-\lambda)\cdot\frac{k}{h^2}\cdot\frac{1}{C_{i+1,j}}+\left(1-2b(1-\lambda)\cdot\frac{k}{h^2}-ck\right)C_{i,j}-\left(a-2d(1-\lambda)\cdot\frac{k}{h^2}-ek\right)\frac{1}{C_{i,j}}+b(1-\lambda)\cdot\frac{k}{h^2}\cdot C_{i-1,j}-d(1-\lambda)\cdot\frac{k}{h^2}\cdot\frac{1}{C_{i-1,j}}+(c-e)k.\quad(4.5)$$

（4．5）式を次の様に書き換える。

$$a_{12}C_{i,j+1}-a_{22}\frac{1}{C_{i,j+1}}=a_{13}C_{i+1,j+1}-a_{23}\frac{1}{C_{i+1,j+1}}+a_{11}C_{i-1,j+1}-a_{21}\frac{1}{C_{i-1,j+1}}+b_{13}C_{i+1,j}-b_{23}\frac{1}{C_{i+1,j}}+b_{12}C_{i,j}-b_{22}\frac{1}{C_{i,j}}+b_{11}C_{i-1,j}-b_{21}\frac{1}{C_{i-1,j}}+(c-e)k.\quad(4.6)$$

ここで，$a_{11}=a_{13}=b\lambda k/h^2$，$a_{12}=1+2b\lambda k/h^2$，$a_{21}=a_{23}=d\lambda k/h^2$，$a_{22}=a+2d\lambda k/h^2$，$b_{11}=b_{13}=b(1-\lambda)k/h^2$，$b_{12}=1-2b(1-\lambda)k/h^2-ck$，$b_{21}=b_{23}=d(1-\lambda)k/h^2$，$b_{22}=a-2d(1-\lambda)k/h^2-ek$とした。初期値$C_{i,0}$が与えられ，$j=1$，2，3，・・・と順次計算していくので，

$$g_j=b_{13}C_{i+1,j}-b_{23}\frac{1}{C_{i+1,j}}+b_{12}C_{i,j}-b_{22}\frac{1}{C_{i,j}}+b_{11}C_{i-1,j}-b_{21}\frac{1}{C_{i-1,j}}+(c-e)k\quad(4.7)$$

とおくと，g_jは既知の値である。このとき（4．6）式は

$$a_{12}C_{i,j+1}-a_{22}\frac{1}{C_{i.j+1}}=a_{13}C_{i+1,j+1}-a_{23}\frac{1}{C_{i+1,j+1}}+a_{11}C_{i-1,j+1}-a_{21}\frac{1}{C_{i-1,j+1}}+g_j\quad(4.8)$$

となる。（4．8）式は未知数3個（$C_{i+1,j+1}$，$C_{i,j+1}$，$C_{i-1,j+1}$）でそのままでは解けないので，Gauss-Seidel反復法で近似解を求める。（4．8）式を$C_{i,j+1}$について解くと

$$C_{i,j+1}=\frac{1}{2a_{12}}\left(a_{13}C_{i+1,j+1}-a_{23}\frac{1}{C_{i+1,j+1}}+a_{11}C_{i-1,j+1}-a_{21}\frac{1}{C_{i-1,j+1}}+g_j\right)+$$

$$\frac{1}{2a_{12}}\left\{\left(a_{13}C_{i+1,j+1}-a_{23}\frac{1}{C_{i+1,j+1}}+a_{11}C_{i-1,j+1}-a_{21}\frac{1}{C_{i-1,j+1}}+g_j\right)^2+4a_{12}a_{22}\right\}^{1/2} \quad (4.9)$$

が得られ，$C_{i,j+1}$が未知数$C_{i+1,j+1}$と$C_{i-1,j+1}$の関数で与えられる。Gauss-Seidel反復法では$C_{i,j+1}$の$(n+1)$次近似${C_{i,j+1}}^{(n+1)}$を$C_{i+1,j+1}$のn次近似${C_{i+1,j+1}}^{(n)}$と$C_{i-1,j+1}$の$(n+1)$次近似${C_{i-1,j+1}}^{(n+1)}$で表す。即ち，

$$\begin{aligned}{C_{i,j+1}}^{(n+1)}=&\frac{1}{2a_{12}}\left(a_{13}{C_{i+1,j+1}}^{(n)}-a_{23}\frac{1}{{C_{i+1,j+1}}^{(n)}}\right.\\&\left.+a_{11}{C_{i-1,j+1}}^{(n+1)}-a_{21}\frac{1}{{C_{i-1,j+1}}^{(n+1)}}+g_j\right)+\\&\frac{1}{2a_{12}}\left\{\left(a_{13}{C_{i+1,j+1}}^{(n)}-a_{23}\frac{1}{{C_{i+1,j+1}}^{(n)}}+\right.\right.\\&\left.\left.a_{11}{C_{i-1,j+1}}^{(n+1)}-a_{21}\frac{1}{{C_{i-1,j+1}}^{(n+1)}}+g_j\right)^2+\right.\\&\left.4a_{12}a_{22}\right\}^{1/2}\end{aligned} \quad (4.10)$$

と近似する。${C_{i,j+1}}^{(0)}=C_{i,j}$（既知の値）として1次，2次と順次計算し，${C_{i,j+1}}^{(n+1)}$と${C_{i,j+1}}^{(n)}$がある誤差の範囲内で一致した時の${C_{i,j+1}}^{(n)}$の値を$C_{i,j+1}$の真の値とする。

5．Javaプログラミング

拡散における各定数，$j+1$での寄与の割合λ, ξとτのきざみhとk，（4.10）式収束判定条件，及び初期分布を与えて（4.10）式を順次解けば良い。拡散における定数に対しては，ここでは拡散メカニズム判定のため，（2.1）式のvacancy型，（2.2）式のself-interstitial型の寄与の割合をパラメータにする。

固体中でvacancy型とself-interstitial型拡散が同時に起こる場合のホスト原子自己拡散係数はD_{self}は

$$D_{self}=f_V D_V C_{V0}/C_0+f_I D_I C_{I0}/C_0 \quad (5.1)$$

で表される。ここで，f_V, f_Iは結晶構造に依存し，シリコン結晶に対しては$f_V=0.5, f_I=0.7273$[5]が与えられている。自己拡散におけるself-interstitial型拡散の割合d_{Isd}とvacancy型拡散の割合d_{Vsd}を次式で与え，この値を各拡散メカニズムの寄与の割合とし，d_{Isd}を濃度分布に対する計算パラメータとする。

$$d_{Isd}=f_I D_I C_{I0}/D_{self}C_0, \quad (5.2)$$

$$d_{Vsd}=f_V D_V C_{V0}/D_{self}C_0=1-d_{Isd}. \quad (5.3)$$

このとき，

$$C_{I0}=d_{Isd}D_{self}C_0/f_I D_I, \quad (5.4)$$

$$C_{V0}=d_{Vsd}D_{self}C_0/f_V D_V \quad (5.5)$$

で与えられ，C_{I0}，D_{self}，D_IまたはC_{V0}，D_{self}，D_Vを同時に定数として与えると，パラメータとして与えたd_{Isd}と矛盾する。どちらのメカニズムが効くにしても，D_{self}が大きいと拡散が速くなるので，この値を拡散時間依存性実験データとのfittingパラメータにとるのが適切である。

Javaによる数値計算については多くの解説書[6]があるので参考にされたい。ここで作成したプログラムを次に示す。プログラムでは，行列データの数値出力が分布に対応するように，iとjを入れ替えj行（τ軸）i列（ξ軸）としている。プログラム中の//はコメント行，/*～*/はコメント文である。また，本プログラムのファイル名はDiffusionAuSi2.javaである。

実行に際しましてまずx軸分割数（57行m）を決める。分割数が大きく刻みが小さい程計算精度は良いが計算時間が長い。ここでは，グラフが連続的に見える100分割（m=99）とした。次に，時間軸刻み（60行r）を決める。陰解法では制限が無いとされているが，この値の計算結果への影響については次章で述べる。拡散実時間と時間刻みから時分割計算回数（62行n）が与えられる。時間刻みを小さくすると計算精度はよくなるが，回数は刻みに反比例して大きくなり計算時間が長くなる。ここでは，まず数十秒で計算出来るn=10000とし，最終的には1分程度で計算出来るn=100000とした。次に初期濃度（50行c0i）を決める。不純物導入では実際の値は0に近いが，kick-out機構が影響すると，濃度の逆数で数値計算するので，この値が小さいとGauss-Seidel反復がなかなか収束しない。計算における初期濃度の影響も次章で述べる。ここでは，0の代りに収束の状況によってc0i=10^{-4}～10^{-8}とした。次に，Gauss-Seidel反復における収束判定値（133行eps）を決める。この値も小さい方が計算精度は良いが計算に時間がかかり，小さすぎると収束しない。計算における収束判定の影響も次章で述べる。ここでは，実際に計算結果が得られるeps=10^{-8}～10^{-12}とした。

計算の結果はJavaでグラフィックスを行うこともできるが，表形式に出力し，使い慣れた表計算ソフトにコピーして作図するのが簡便である。通常の表計算ソフトではタブキーで行替，改行キーで列替のデータがコピーされるので，データをタブキーで区切る（例えば174行"¥t"）と良い。また，表計算ソフトでは，

```
/* M.Morooka 2001.9.1 numerical solution of Au diffusion in Si,implicit and Gauss-Seidel method*/
import java.io.*; //入出力パッケージ使用
public class DiffusionAuSi2 {
    public static void main(String args[]) throws IOException {

        //温度temp、dIsd、試料厚xLの読み込み
        double temp,dIsd,xL;
        InputStreamReader in = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(in);
        System.out.print(" Input temperatyre (C):  ");
        String n1 = br.readLine();
        temp = Double.valueOf(n1).doubleValue();
        System.out.print(" Input component of interstitial mechanism dIsd:  ");
        String n2 = br.readLine();
        dIsd = Double.valueOf(n2).doubleValue();
        System.out.print(" Input smple thickness (cm):  ");
        String n3 = br.readLine();
        xL = Double.valueOf(n3).doubleValue();

        //定数を与える
        double c0,k,kT,cs0,dself,dVsd,dV,fV,cV0,fI,aD,dI,cI0;
        c0=4.9E22; // density of Si atom (1/cc)
        k=8.617386E-5;
        kT=k*(temp+273.15);
        cs0=8.15E22*Math.exp(-1.76/kT);//solid solubility of sub. Au
        dself=100*1.46E3*Math.exp(-5.02/kT); // self diffusion constant of Si
        dVsd=1.0-dIsd;
        dV=9.0E-5*Math.exp(-0.18/kT); // diffusion constant of vacancy
        fV=0.5;
        cV0=dVsd*dself*c0/(fV*dV);
        fI=0.7273;
        aD=1.0; // ratio of diffusion constants aD=dV/dI
        dI=dV/aD;
        cI0=dIsd*dself*c0/(fI*dI);
        System.out.println("  ");
        System.out.println(" substitutional Au concentration cs0 = "+cs0+"  (/cc)");
        System.out.println(" vacancy concentration cV0 = "+cV0+"  (/cc)");
        System.out.println(" vacacy diffusion dV = "+dV+"  (cm^2/s)");
        System.out.println(" self-interstitial  concentration cI0 = "+cI0+"  (/cc)");
        System.out.println(" self-interstitial diffusion dI = "+dI+"  (cm^2/s)");
        System.out.println(" self diffusion constant of Si dself = "+dself+"  (cm^2/s)");

        double kVd,kId;
        kVd=0.0; // vacancy capture rate by dislocation, kVd=2π*nd*dV/ln(rd/rd0)
        kId=0.0; // self-interstitial capture rate by dislocation, kId=2π*nd*dI/ln(rd/rd0)
        System.out.println(" vacancy capture rate by extended-defects kVd = "+kVd+"  (/s)");
        System.out.println(" self-interstitial capture rate by extended-defects kId = "+kId+"  (/s)");
        //初期値、境界値
        double c0i,c00,c01;
        c0i=1.0E-6; // initial condition at t=0
        c00=1.0; // boundary condition at x=0
        c01=1.0; // boundary condition at x=1

        //距離と時間の分割
        int m,j,jmax,n,i,imax,ipn,ipm; // i：時間t軸、j：距離x軸
        double h,r,kk,tmax;
        m=99; // xの分割数(m+1) x=0 -> j=0, x=1 -> j=jmax
        jmax=m+1;
        h=1.0/(m+1); // increment of x,     h=Δx
        r=0.5*1.0; // r=k/h^2 (陰解法では制限無し.陽解法では< 0.5 )
        kk=r*h*h; // increment of time t,    kk=Δt
        n=100000; // 時分割計算数, t=0 -> i=0, t=tmax -> i=imax
        imax=n;//ipnの整数倍であることが望ましい
        tmax=kk*imax;

        ipn=10; // output No. among 0~imax
        ipm=imax/ipn;

        double ramda;
        ramda=0.5; // ramda=0.5, Crank-Nicorson

        //計算における数値
        double aa,ab,ac,ad,ae,a11,a21,a12,a22,a13,a23,b11,b21,b12,b22,b13,b23;
        aa=cI0/(cs0+cV0);
        ab=dV*cV0/((dV+dI)*(cs0+cV0));
        ac=kVd*cV0*xL*xL/((dV+dI)*(cs0+cV0));
        ad=dI*cI0/((dV+dI)*(cs0+cV0));
        ae=kId*cI0*xL*xL/((dV+dI)*(cs0+cV0));

        a11=ab*ramda*r;
        a21=ad*ramda*r;
        a12=1.0+2*ab*ramda*r;
        a22=aa+2*ad*ramda*r;
        a13=a11;
        a23=a21;
        b11=ab*(1.0-ramda)*r;
        b21=ad*(1.0-ramda)*r;
        b12=1.0-2*ab*(1.0-ramda)*r-ac*kk;
        b22=aa-2*ad*(1.0-ramda)*r-ae*kk;
        b13=b11;
        b23=b21;

        System.out.println("  a11= "+a11+"¥t"+"  a21= "+a21+"¥t"+"  a12= "+a12+"¥t"+"  a22= "+a22+"¥t"+"  a13= "+a13+"¥t"+"  a23= "+a23);
        System.out.println("  b11= "+b11+"¥t"+"  b21= "+b21+"¥t"+"  b12= "+b12+"¥t"+"  b22= "+b22+"¥t"+"  b13= "+b13+"¥t"+"  b23= "+b23);

        // 行列定義
        double [] f0 = new double[jmax+1]; // 初期分布
        double [] f1 = new double[jmax+1]; // Gauss Seidel 繰り返し計算におけるi分布
```

```
        double [] f2 = new double[jmax+1]; // Gauss Seidel 繰り返し計算における(i+1)分布
        double [][] fOut = new double[ipn+1][jmax+1]; // 出力行列fOut[i][j],i行j列
        double [] time = new double[ipn+1]; // 出力時拡散時間

        // boundary condition
        f0[0]=c00;
        f1[0]=c00;
        f2[0]=c00;
        f0[jmax]=c01;
        f1[jmax]=c01;
        f2[jmax]=c01;
        for(i=0;i<ipn+1;i++) {
            fOut[i][0]=c00;
            fOut[i][jmax]=c01;
        }
        // initial condition
        for(j=1;j<jmax;j++) {
            f0[j]=c0i;
            f1[j]=c0i;
            fOut[0][j]=c0i;
        }
        time[0]=0.0;

        // Crank Nicolson implicit method
        /* a12*f2[j] - a22/f2[j] =
             a13*f2[j+1] - a23/f2[j+1] + a11*f2[j-1] - a21/f2[j-1] +
             b13*f1[j+1] - b23/f1[j+1] + b12*f1[j] -b22/f1[j] + b11*f1[j-1] - b21/1[j-1] + kk*(ac-ae)
        */

        // Gauss Seidel反復法
        double eps,sum,fOld,kkacae;
        int ii,ipd,ip;
        ipd=0;
        ip=0;
        eps=1.0E-8; // 収束判定値
        kkacae=kk*(ac-ae);
        for(i=0;i<imax;i++){
            for(j=0;j<jmax+1;j++){
                f2[j]=f1[j];
            }
            ii=0;
            do{
               sum=0.0;
               for(j=1;j<jmax;j++){
                fOld=f2[j];
                f2[j]=(a13*f2[j+1] - a23/f2[j+1] + a11*f2[j-1] - a21/f2[j-1] +
                       b13*f1[j+1] - b23/f1[j+1] + b12*f1[j] -b22/f1[j] + b11*f1[j-1] - b21/f1[j-1] + kkacae)/(2*a12) +
                       Math.sqrt((a13*f2[j+1] - a23/f2[j+1] + a11*f2[j-1] - a21/f2[j-1] +
                       b13*f1[j+1] - b23/f1[j+1] + b12*f1[j] -b22/f1[j] + b11*f1[j-1] - b21/f1[j-1] + kkacae)*
                       (a13*f2[j+1] - a23/f2[j+1] + a11*f2[j-1] - a21/f2[j-1] +
                       b13*f1[j+1] - b23/f1[j+1] + b12*f1[j] -b22/f1[j] + b11*f1[j-1] - b21/f1[j-1] + kkacae)+4*a12*a22)/(2*a12);
                   sum=sum+Math.abs(fOld-f2[j]);
               }
               //System.out.println(" sum = "+sum);//誤差の和
               ii=ii+1;
            }while(sum>eps);
            //System.out.println(" ii = "+ii);//繰り返し回数
            for(j=0;j<jmax+1;j++){
                f1[j]=f2[j];
            }
            //計算結果のipm毎の出力
            ipd=ipd+1;
            if(ipd>=ipm){
               ipd=0;
               ip=ip+1;
               for(j=0;j<jmax+1;j++){
                fOut[ip][j]=f2[j];
               }
               time[ip]=ip*ipm*kk*xL*xL/(dV+dI);
            }
        }

        // 行列のプリント
        for(i=0;i<ipn+1;i++){
            for(j=0;j<jmax+1;j++){
                System.out.print(fOut[i][j]+"¥t"); // "¥t" : タブキーの送り
            }
            System.out.print("¥n"); // 改行
        }
        // 拡散時間プリント
        for(i=0;i<ipn+1;i++){
            System.out.print("t = "+time[i]+" (s)  (i="+i*ipm+")"+"¥t"); // "¥t" : タブキーの送り
        }
        System.out.print("¥n");
        System.out.print("¥n");

        // 行列の転置
        for (j=0; j<jmax+1; j++) {
            for (i = 0; i < ipn+1; i++ ) {
                System.out.print(fOut[i][j]+"¥t");
            }
            System.out.print("¥n");
        }
    }
}
```

通常列毎にデータ種が異なるのに対して，Javaプログラムでは行毎にデータ種が異なる方が出力に便利であるので，最後に出力データの行列を転置（186−189行)する。ここでは，表計算ソフトとしてKaleidaGraphを用いた。

6．計算結果の時間刻み，収束判定値，初期濃度依存性

6．1　時間刻み依存性

図1にr=0.5，図2にr=0.5×100，図3にr=0.5×1000の場合の計算結果を示す。計算に用いたその他の値は図中に記す。図1の計算ではn=100000となるために，約1分の計算時間を要する。r=0.5×1000では表面近くの計算の初期で収束が良く無いが，t>482sの収束した領域では図1，2の結果と大きな違いは認められない。即ち，図1，2に見られるように，計算が収束すれば，計算結果に対する時間刻みの依存性は無いと言える。図1，2，3の場合で収束するのは，r=0.5×100程度までであるが，この値は初期濃度と収束判定値が小さいと減少し，初期濃度が1／100（$c0i=10^{-6}$）になると，収束最大刻みは約1／10（r=0.5×10程度まで）となる。時間刻みが収束最大刻みより大きくなると収束せず，Gauss-Seidel繰返し計算を続ける。

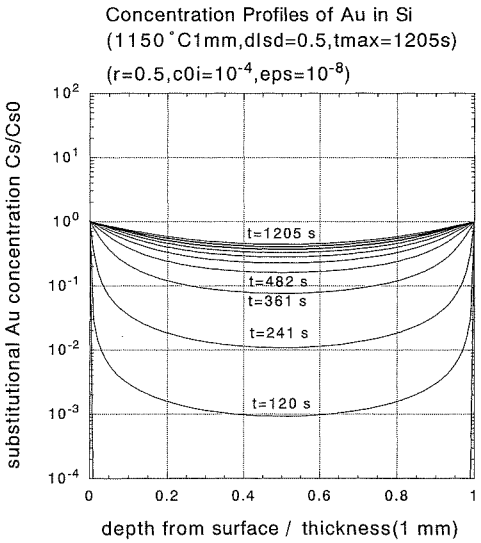

図1．計算結果の時間刻み依存性（r=0.5）

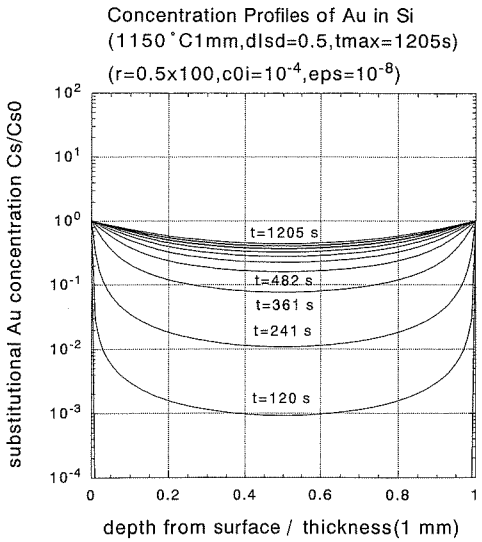

図2．計算結果の時間刻み依存性（r=0.5×100）

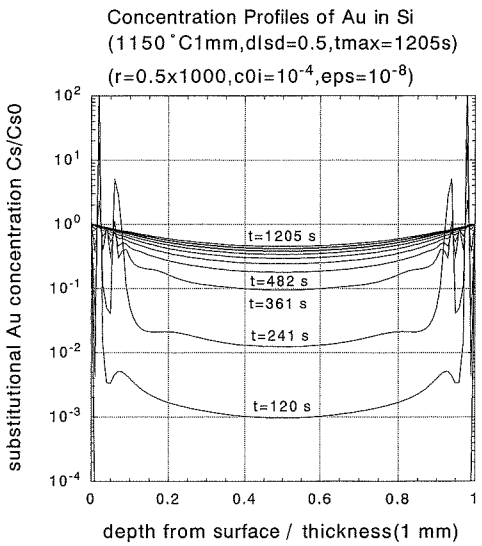

図3．計算結果の時間刻み依存性（r=0.5×1000）

6．2　収束判定値依存性

図4に収束判定値$eps=10^{-10}$，図5に$eps=10^{-8}$，図6に$eps=10^{-6}$の場合の計算結果を示す。計算に用いたその他の値は図中に記す。図6の$eps=10^{-6}$の場合には，t>723sの領域では図4，5の結果と大きな違いは認められないが，時間が短い時の分布で誤差が大きい。即ち，収束判定値については，その値を減少

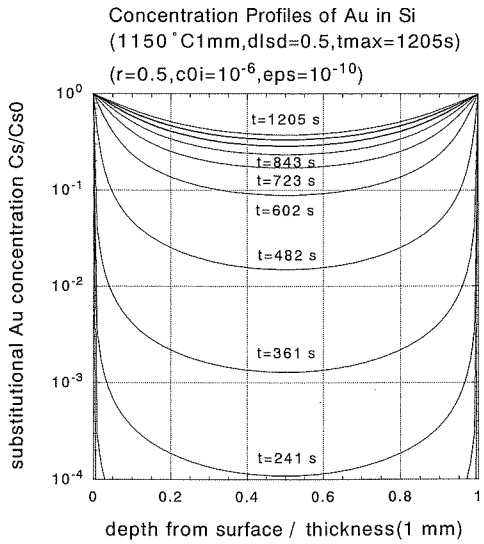

図４．計算結果の Gauss-Seidel 反復収束判定値依存性（eps＝10^{-10}）

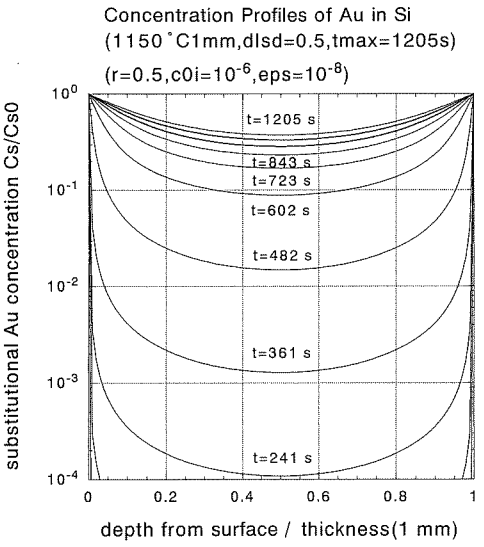

図５．計算結果の Gauss-Seidel 反復収束判定値依存性（eps＝10^{-8}）

させながら数回計算し，分布が収束した値を使用したが良い。図４，５に見られるように，今の場合には eps＝10^{-8}程度以下を使用すれば充分である。収束判定値が小さすぎると Gauss-Seidel 反復が収束せず繰返し計算を続ける。この場合プログラム152行と155行のコメント指定命令（//）を削除することによって，おおよその最小判定値が推定出来る。この最小値は初期

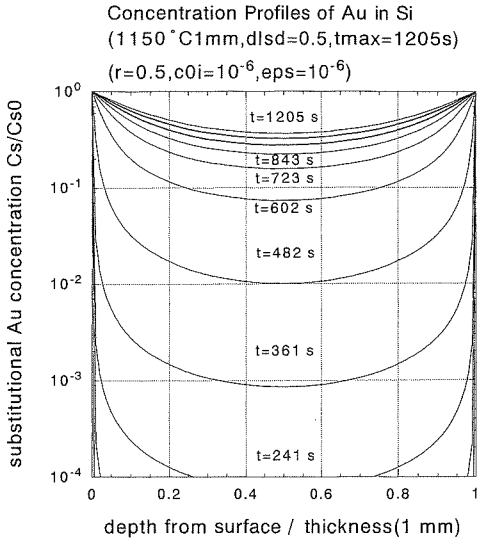

図６．計算結果の Gauss-Seidel 反復収束判定値依存性（eps＝10^{-6}）

濃度に反比例し，図４，５，６の c0i＝10^{-6}の場合には eps＝2.5×10^{-11}で図１，２，３の c0i＝10^{-4}の場合には eps＝2.5×10^{-13}であった。

6．3　初期濃度依存性

図７に初期濃度 c0i＝5.0×10^{-8}，図８に c0i＝10^{-6}，図９に c0i＝10^{-4}の場合の計算結果を示す。計算に用

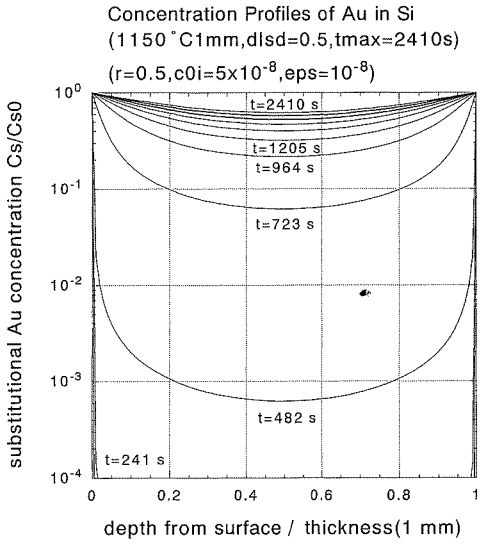

図７．計算結果の初期濃度依存性（c0i＝5.0×10^{-8}）

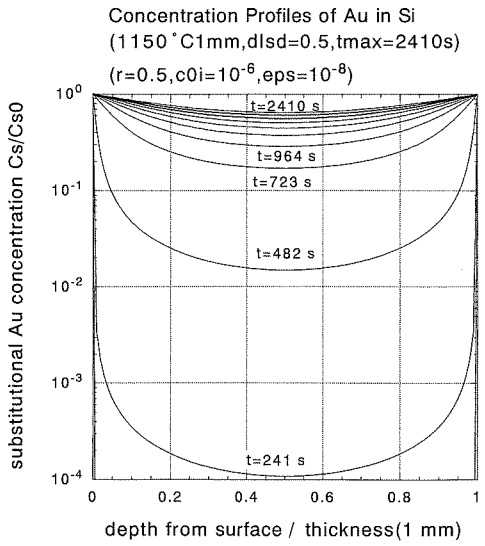

図８．計算結果の初期濃度依存性（c0i＝10^{-6}）

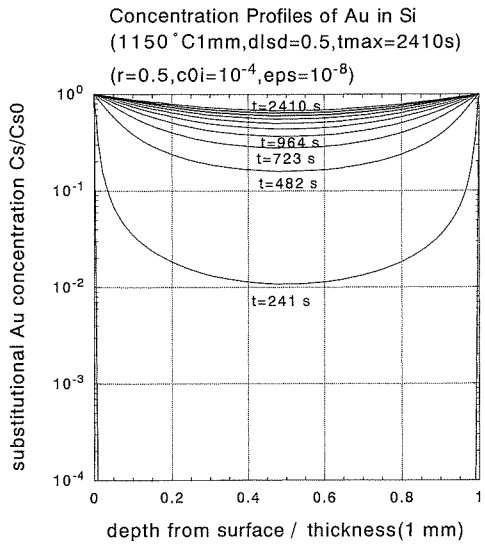

図９．計算結果の初期濃度依存性（c0i＝10^{-4}）

いたその他の値は図中に記す。もともと式（２．２）の kick-out 拡散に於いては，実効拡散係数が濃度依存性を示し，計算結果は初期濃度に依存する。この依存度は，図７，８，９に見られるように拡散の初期に大きい。不純物導入の場合には，初期濃度はできるだけ小さい方が実際の初期条件に近いが，図10に示すように，ある値より小さ過ぎると濃度が増加せず，良い計算結果が得られない。よって，c0i を減少させ

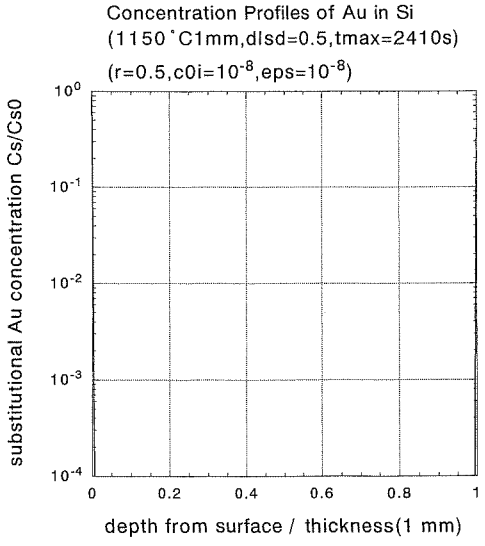

図10．初期濃度が小さ過ぎて良い計算結果が得られない場合（c0i＝10^{-8}）

て計算し，導入分布が得られるもっとも小さい値を使用したが良い。

７．計算結果

７．１　不純物導入分布

図11に dIsd＝0.0（完全解離機構），図12に dIsd＝1.0（完全 kick-out 機構），図13に dIsd＝0.1（10%kick

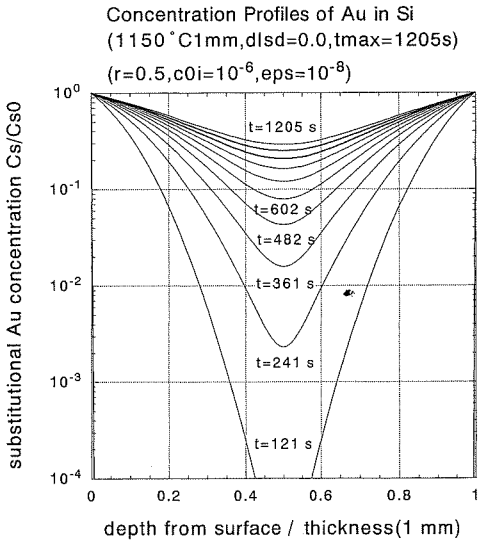

図11．不純物導入分布（完全解離機構 dIsd＝0.0）

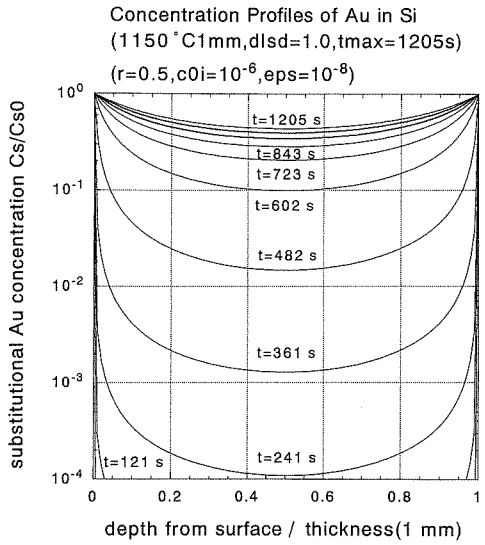

図12. 不純物導入分布(完全 kick-out 機構 dIsd=1.0)

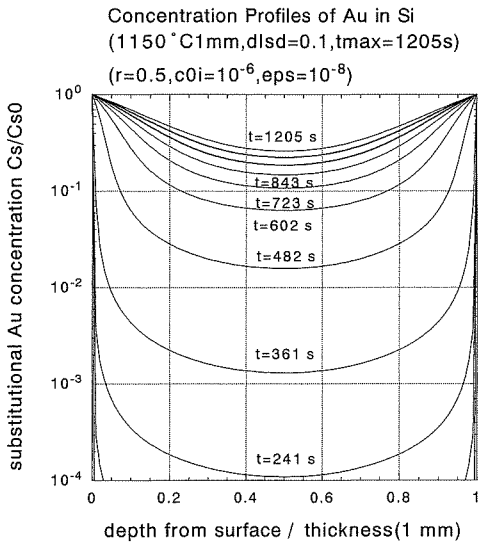

図13. 不純物導入分布(10%kick-out 機構 dIsd=0.1)

-out 機構）の場合の導入時濃度分布の計算結果を示す。計算に用いたその他の値は図中に記す。一般に知られている様に，解離機構は補誤差型分布，kick-out 機構は U 型分布が得られている。10%kick-out 機構では，試料中央部の拡散時間が短い間は kick-out 機構が効いているが，試料表面近くでは解離機構の影響が見られる。このことは，kick-out 機構の実効拡散係数が濃度の二乗に反比例[7]し，濃度が小さいと大きくなり，大きいと小さくなる事によるものと思われる。いずれにせよ濃度の増加に対して，図11と12のそれぞれ濃度が高い方の影響が大きくなる。

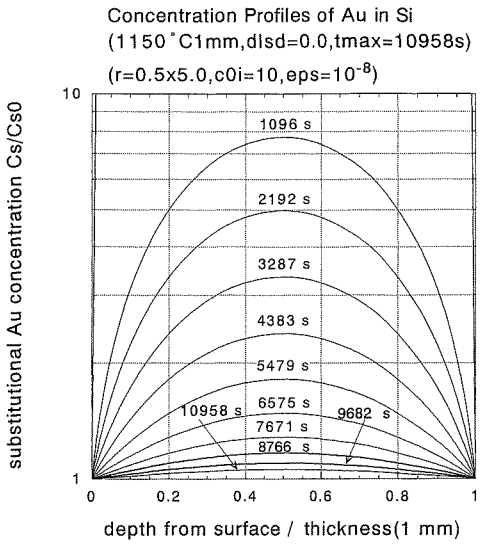

図14. 不純物外方拡散分布（完全解離機構 dIsd =0.0)

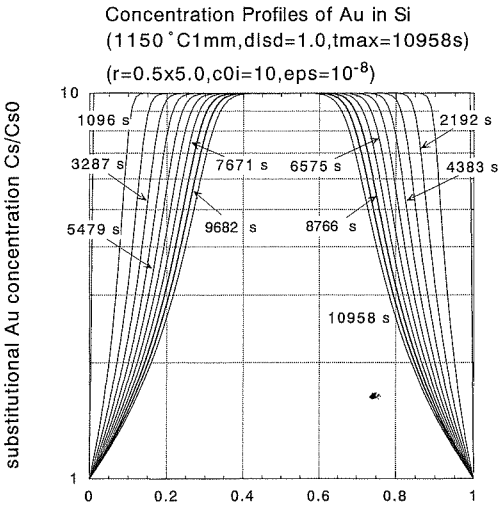

図15. 不純物外方拡散分布（完全 kick-out 機構 dIsd =1.0)

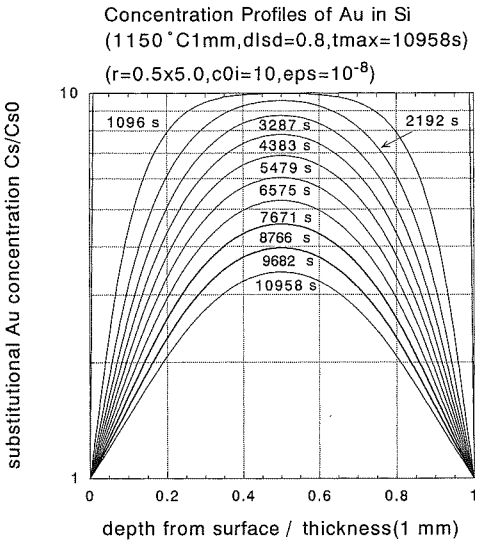

図16．不純物外方拡散分布（80%kick-out機構 dIsd =0.8）

7．2　不純物外方拡散分布

予め高濃度の不純物を含む物質を熱処理すると，過飽和不純物は熱平衡濃度へ減少し（この事を回復と呼ぶ），濃度分布は表面で濃度が少ない外方拡散分布となる。初期濃度が熱平衡濃度の10倍の時の外方拡散分布を図14（完全解離機構），15（完全kick-out機構），16（80%kick-out機構）に示す。kick-out機構の場合には，7．1節で述べたように濃度の大きい中央部での濃度変化が小さく，図15に示すように回復しにくい。よって，本節での計算では，C_{I0}の値を大きくして回復を早くするために，プログラム32行 aD=10.0 とした。その他，初期濃度以外の定数は導入分布の場合と同じである。図14の完全解離機構では，拡散係数が濃度によらない補誤差関数型濃度減少に従って，試料の内部でも濃度が減少する。一方，実効拡散係数が濃度の二乗に反比例するkick-out拡散では，図15に見られるように試料表面近くでは濃度減少が加速されるのに対して，中央部ではなかなか減少しない。二つのメカニズムが両方効く場合には，濃度減少が大きい方が影響し，図16に見られる様に，試料表面近くではkick-out機構が実効的となり，中央部では解離機構が実効的となる。

8．ま と め

解離機構とkick-out機構が同時に影響する不純物拡散における非線形拡散方程式を，javaで解くためのプログラミングを行った。Si中Au拡散について濃度分布を計算したところ，旧式パソコン使用にもかかわらず，実際の分布を解析するのに十分に役に立つ結果が得られた。なお，ここで提示した図を得るために要する計算時間は，最大の場合で1分程度であった。

謝　辞

Mac使用に対して種々御教示いただいた電気工学科梶原寿了教授に感謝いたします。

参 考 文 献

1）W. Frank, U. Gösele, H. Mehrer and A. Seeger : Diffusion in Crystalline Solids ed. by G. E. Murch and A. S. Nowick (Academic Press, 1984) p.134.
2）師岡正美，友景　肇，吉田正幸：信学技報 Vol. 87, No. 94（1987）pp. 21－27.
3）例えば，B. Carnahan, H. A. Luther, J. O. Wilkes 著，藤田宏他訳：計算機による数値計算法（日本コンピュータ協会，1982）.
4）例えば，桃島茂治：電学誌，112巻10号（1992）pp. 785－789.
5）K. Comppaan and Y. Haven: Trans. Faraday Soc.52 (1956) p.768 and 54 (1958) p.1498.
6）例えば，赤間世紀：Java 2による数値計算（技報堂出版，1999）
7）例えば，文献1）p. 119.