第 12 章　Phase-field 法２（拡散相分解）

以下では拡散相分解における Phase-field 法の例として、Fe-Cr 合金のα(bcc)相のスピノーダル分解を例に取り、相分解過程の計算手法について具体的に解説する。また章末には付録として、Fe-Cr 合金のソースプログラム（α(bcc)相における２次元スピノーダル分解）および単純な正則溶体近似における１次元濃度プロファイルの計算プログラムを紹介する。

12-1 Fe-Cr合金の相分離の計算

同一結晶構造で、濃度場における相分離であるので、この場合の Phase-field 法は、Cahn-Hilliard の非線形拡散方程式に基づく相分解シミュレーションに等しい。またここではできるだけ議論を簡単にするために、2 次元シミュレーションについて説明し、また強磁性および反強磁性の過剰自由エネルギー項は無視する。したがって、Fe-Cr 合金の平衡状態図に直接対応した化学的自由エネルギーにはなっていないが、この部分の修正は、化学的自由エネルギーの関数形が変わるだけであるので、容易に作り直すことができるはずである。（この系では、強磁性の過剰自由エネルギーが２相分離領域を広げているので、状態図に対応した定量的な相分解計算を目的とする場合には、磁性項も考慮しなくてはならない。）また弾性場についても等方弾性論にて計算を行う。

まず本計算に必要な物質パラメータは以下のようにまとめられる。

12-2 各種パラメータの設定

12-2-1 化学的自由エネルギー

α(bcc)相の化学的自由エネルギーは、c を Cr 濃度として式(1)にて与えられる。

$$G_c(c,T) = \Omega_{12} c(1-c) + RT\{c\log(c) + (1-c)\log(1-c)\} \tag{1}$$

R はガス定数で、T は絶対温度である。Ω_{12} は原子間相互作用パラメータで、Fe-Cr 合金の平衡状態図に関する熱力学データベースから、

$$\Omega_{12} = 21020.8 - 9.31889T, \quad [\mathrm{J/mol}] \tag{2}$$

と得られる[1)]。これより化学的自由エネルギーに起因するポテンシャルは、

$$\mu_c \equiv \frac{\partial G_c}{\partial c} = \Omega_{12}(1-2c) + RT\{\log(c) - \log(1-c)\} \tag{3}$$

となる。

12-2-2 濃度勾配エネルギー

濃度勾配エネルギー式は、２次元計算では、

$$E_{surf} = \kappa_c \left(\frac{\partial c}{\partial x}\right)^2 + \kappa_c \left(\frac{\partial c}{\partial y}\right)^2 \tag{4}$$

にて与えられる。ただし濃度勾配エネルギー係数 κ_c は方向に依存しない定数と仮定する。問題は、濃度勾配エネルギー係数 κ_c をどのように設定するするかである。化学的自由エネルギー、界面エネルギー密度 γ_s（合金の整合相分解の場合、$\gamma_s \cong 0.1(\mathrm{J/m^2})$ である）および界面の幅 d（合金の整合相分解の場合、$d \cong 0.5$〜3nmである）より、κ_c をある程度見積もることは可能であるが、ここでは、κ_c をフィッティングパラメータと見なす。つまり実験で得られたスピノーダル分解の変調構造の波長の値を再現できるように κ_c を決定する。条件（温度、組成）を変えても、通常、κ_c 値は大きく変化しないので、ある１つの実験条件で κ_c 値を定めれば良い。なお本シミュレーションにおけ

るフィッティングパラメータはこの κ_c のみである。実際の計算では、$\kappa_c = 6.0\times10^{-15}(\mathrm{J\cdot m^2/mol})$ と仮定した。また勾配エネルギーに起因するポテンシャルは、変分原理に基づき、

$$\mu_{surf} = -2\kappa_c\left(\frac{\partial^2 c}{\partial x^2}+\frac{\partial^2 c}{\partial y^2}\right) \tag{5}$$

と計算される。

12-2-3 弾性歪エネルギー

弾性歪エネルギーについては、基本的に等方弾性体を仮定して計算を行う。まずFeとCrの弾性定数は、

$$C_{11}^{Fe} = 2.3310\times10^{11},\quad C_{12}^{Fe} = 1.3544\times10^{11},\quad C_{44}^{Fe} = 1.1783\times10^{11}(\mathrm{N/m^2})$$
$$C_{11}^{Cr} = 3.50\times10^{11}\quad,\quad C_{12}^{Cr} = 0.678\times10^{11},\quad C_{44}^{Cr} = 1.008\times10^{11}(\mathrm{N/m^2})$$

と与えられる[2)]。次にこれを合金組成 c_0 で重み付き平均する。

$$C_{11} = C_{11}^{Fe}(1-c_0)+C_{11}^{Cr}c_0,\qquad C_{12} = C_{12}^{Fe}(1-c_0)+C_{12}^{Cr}c_0$$

等方体では、弾性率の関数 $Y_{<hkl>}$ は $Y_{<100>}$ に等しいので、

$$Y_{<100>} = C_{11}+C_{12}-2\frac{C_{12}^2}{C_{11}} \tag{6}$$

を用いる。格子ミスマッチは、FeとCrの格子定数 $a_{Fe} = 0.28664nm$ および $a_{Cr} = 0.2884nm$ から[2)]、$\eta = (a_{Cr}-a_{Fe})/a_{Fe} = 0.00614$ と評価される。弾性歪エネルギー式は、

$$E_{str}(c) = \eta^2 Y_{<hkl>}(c-c_0)^2 \tag{7}$$

であるので、弾性歪エネルギーに起因するポテンシャルは、

$$\mu_{str} = 2\eta^2 Y_{<hkl>}(c-c_0) \tag{8}$$

と計算される。

12-2-4 拡散係数

FeとCrの自己拡散係数は、

$$D_{Fe}^{*} = 1.0\times10^{-4}\exp\left(-\frac{294000}{RT}\right)\quad,\quad D_{Cr}^{*} = 2.0\times10^{-5}\exp\left(-\frac{308000}{RT}\right)$$

にて与えられる[2)]。これより原子の拡散の易動度は、M_X を X 元素の易動度として、

$$\begin{aligned} M(c_{Fe},c_{Cr}) &= (M_{Cr}c_{Fe}+M_{Fe}c_{Cr})c_{Fe}c_{Cr} \\ &= \frac{D_{Cr}^{*}c_{Fe}+D_{Fe}^{*}c_{Cr}}{RT}c_{Fe}c_{Cr} = \frac{D_{Fe}^{*}}{RT}\left(\frac{D_{Cr}^{*}}{D_{Fe}^{*}}c_{Fe}+c_{Cr}\right)c_{Fe}c_{Cr} \end{aligned} \tag{9}$$

と表される。ここで$D_{Fe}^{*} = M_{Fe}RT,\quad D_{Cr}^{*} = M_{Cr}RT$を用いた（正確には$M_{Fe}$と$M_{Cr}$は組成の関数であるので注意）。

12-3 非線形拡散方程式の無次元化

拡散方程式の無次元化について説明する。まずエネルギーは全て、*RT*にて無次元化する。また計算領域の１辺を*L*、差分計算の分割数を*N*とすると、差分による空間分割セルの一辺の長さb_1は、$b_1 = L/N$である。このb_1を用いて距離を無次元する。また時間はb_1^2 / D_{Fe}^{*}にて無次元化する。

以上より２次元における非線形拡散方程式は、

$$\begin{aligned}\frac{\partial c}{\partial t} &= M(c)\left(\frac{\partial^2\chi}{\partial x^2}+\frac{\partial^2\chi}{\partial y^2}\right)+\left(\frac{\partial M}{\partial c}\right)\left\{\left(\frac{\partial c}{\partial x}\right)\left(\frac{\partial\chi}{\partial x}\right)+\left(\frac{\partial c}{\partial y}\right)\left(\frac{\partial\chi}{\partial y}\right)\right\}\\ &= \frac{D_{Fe}^{*}}{RT}\left(\frac{D_{Cr}^{*}}{D_{Fe}^{*}}(1-c)+c\right)c(1-c)\left(\frac{\partial^2\chi}{\partial x^2}+\frac{\partial^2\chi}{\partial y^2}\right)\\ &\quad+\frac{D_{Fe}^{*}}{RT}\left\{\left(\frac{D_{Cr}^{*}}{D_{Fe}^{*}}(1-c)+c\right)(1-2c)+\left(1-\frac{D_{Cr}^{*}}{D_{Fe}^{*}}\right)c(1-c)\right\}\left\{\left(\frac{\partial c}{\partial x}\right)\left(\frac{\partial\chi}{\partial x}\right)+\left(\frac{\partial c}{\partial y}\right)\left(\frac{\partial\chi}{\partial y}\right)\right\}\end{aligned} \tag{10}$$

となり、したがって、無次元化した方程式は、

$$\begin{aligned}\frac{\partial c}{\partial\left(\dfrac{t}{b_1^2/D_{Fe}^{*}}\right)} &= \left(\frac{D_{Cr}^{*}}{D_{Fe}^{*}}(1-c)+c\right)c(1-c)\frac{b_1^2}{RT}\left(\frac{\partial^2\chi}{\partial x^2}+\frac{\partial^2\chi}{\partial y^2}\right)\\ &\quad+\left\{\left(\frac{D_{Cr}^{*}}{D_{Fe}^{*}}(1-c)+c\right)(1-2c)+\left(1-\frac{D_{Cr}^{*}}{D_{Fe}^{*}}\right)c(1-c)\right\}\frac{b_1^2}{RT}\left\{\left(\frac{\partial c}{\partial x}\right)\left(\frac{\partial\chi}{\partial x}\right)+\left(\frac{\partial c}{\partial y}\right)\left(\frac{\partial\chi}{\partial y}\right)\right\}\end{aligned} \tag{11}$$

となる。なお濃度勾配エネルギー定数κは$b_1^2 RT$にて無次元化する。また計算は周期的境界条件にて行う。

12-4 プログラムの説明

シミュレーションプログラムを以下の付録に示す。説明は全てプログラム内に記入してある。（Javaでは、// 以降その行の終わりまでがコメント文となるので、ソースプログラム中に全ての説明を比較的容易に記入することができる。またその方がプログラム内容の理解には適しているであろう。）なお以下のプログラムは、全体を見やすくするために、若干冗長なプログラムとなっているが、アルゴリズムを少し手直しするだけで、さらなる高速化が可能である。

図12-1はシミュレーション結果の一例である。合金はFe-40at%Cr合金で、温度673Kにおける等温時効の相分解過程を本プログラムにて計算した結果で、黒化度にてCr濃度を表している。図中の時間(*s'*)は無次元化された時間である。相分解初期にスピノーダル分解によって均一な"まだら構造"が形成され、時効の進行に伴い析出粒子がオストワルド成長していく様子が計算されている（計算所用時間は通常のデスクトップパソコン(クロック数1GHz)にて数分程度）。

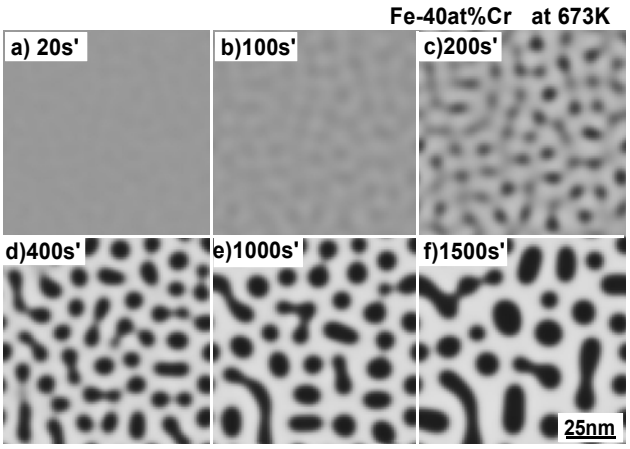

図 12-1 Fe-40at%Cr 合金の 673K 等温時効における２次元相分解シミュレーション結果

参考文献
(1) Y-Y.Chuang, J-C.Lin, and Y.A.Chang : CALPHAD, **11**(1987), pp.57-72.
(2) 金属データブック(改定３版), 日本金属学会編, (1993), 丸善

付録：相分解シミュレーションプログラム
・Java アプリケーションプログラム(FeCr2D_save.java)

```
//************************************************************
import java.awt.*;
import java.awt.event.*;
import java.io.*;

public class FeCr2D_save extends Frame{

   static int ND=80;                  //組織１辺の分割数
   static int nd=ND, ndm=ND-1;
   static int width, height, insetx, insety, xwidth, yheight;
   static double RR=8.3145;       //ガス定数
   static double [][] ch=new double[ND][ND]; //組織内の濃度データ配列

   //グローバル変数化
   static double c0, temp, time1;        //合金組成、温度、計算時間
   Image buff;
   static Graphics bg, g;

   //*************** コンストラクタ ***************************
   public FeCr2D_save(){
      xwidth=400; yheight=400; insetx=4; insety=30;
      width=xwidth+insetx*2;    height=yheight+insetx+insety;
      setSize(width, height);
      setBackground(Color.lightGray);
      setVisible(true);
      buff=this.createImage(width, height);
      bg=buff.getGraphics();
      addWindowListener(new WindowAdapter(){
         public void windowClosing(WindowEvent e){ System.exit(0); }
      });
   }

//************** main *****************************
   public static void main(String[] args){

      FeCr2D_save prog=new FeCr2D_save();

      double [][] ck = new double[ND][ND];          //拡散ポテンシャル
      double [][] ch2 = new double[ND][ND];         //組織内の濃度データ予備配列
      double mu_chem, mu_str, mu_surf;              //各ポテンシャル
      int    i, j, k, l;                            //整数
      double time1max;                              //最大時間
      double c1, c2;                                //濃度
      double cddtt;                                 //濃度の増分
      double kapa_c;                                //濃度勾配エネルギー定数
      double al;                                    //計算領域
```

```
    double b1;                          //差分ブロックサイズ
    double delt;                        //時間刻み
    double a0;                          //Feの格子定数
    double vm0;                         //モル体積
    double c11a, c12a, c44a;            //Feの弾性率
    double c11b, c12b, c44b;            //Crの弾性率
    double c11, c12, c44;               //合金の平均弾性率
    double y100;                        //弾性関数
    double eta;                         //格子ミスマッチ
    double Da, Db, Dab;                 //自己拡散係数とその比
    double Mc0;                         //易動度関数
    double L0;                          //原子間相互作用パラメータ

    double c2ip, c2im, c2jp, c2jm;      //差分ブロックにおいてcを中心に、その上下左右の濃度
    double ck1dev, ck2dev;              //拡散ポテンシャルの１階微分と２階微分
    double ck0, ckip, ckim, ckjp, ckjm; //差分ブロックにてck0を中心にその上下左右の拡散ポテンシャル
    int    ip, im, jp, jm;      //整数（i+1, i-1, j+1, j-1）
    double sumc, dc0;           //  濃度場の総和、平均組成からのずれ

    String s_temp, s_c0, s_delt;

//-------------------------------------------------------------------------
    BufferedReader input=new BufferedReader(new InputStreamReader(System.in));

    s_c0="0.4";
    try{ System.out.print("c0(0.4) =    ");   s_c0=input.readLine(); }
    catch(IOException e){ System.out.println("Exception : "+e); }
    c0=new Double(s_c0).doubleValue();

    s_temp="873.0";
    try{ System.out.print("temp(673.0)/K =    ");   s_temp=input.readLine(); }
    catch(IOException e){ System.out.println("Exception : "+e); }
    temp=new Double(s_temp).doubleValue();

    s_delt="0.1";
    try{ System.out.print("del_time1(0.1) =    ");   s_delt=input.readLine(); }
    catch(IOException e){ System.out.println("Exception : "+e); }
    delt=new Double(s_delt).doubleValue();

//**** 各種定数の設定 ************************************************************
    al=150.0;                   //計算領域の一辺の長さ(nm)
    al=al*1.0e-9;               //(m)に変換
    b1=al/nd;                   //差分１ブロックのサイズ
    time1=0.0;                  //スタート時間(ループの回数)
    time1max=10001.;            //計算打切時間(ループの回数)

    a0=2.8664E-10;                          //Feの格子定数
    vm0=6.02E23*a0*a0*a0/2.0;               //モル体積
    L0=(21020.8-9.31889*temp)/RR/temp;      //原子間相互作用パラメータ（すでに無次元化済み）
    kapa_c=6.0e-15/b1/b1/RR/temp;           //濃度勾配エネルギー係数（ で無次元化）
    eta=0.00614;                            //格子ミスマッチ
```

```
    c11a=2.331e11*vm0/RR/temp;        //Feの弾性率、 で無次元化
    c12a=1.3544e11*vm0/RR/temp;
    c44a=1.1783e11*vm0/RR/temp;
    c11b=3.5e11*vm0/RR/temp;          //Crの弾性率
    c12b=0.678e11*vm0/RR/temp;
    c44b=1.008e11*vm0/RR/temp;

    c11=(1.0-c0)*c11a+c0*c11b;        //合金の弾性率
    c12=(1.0-c0)*c12a+c0*c12b;
    c44=(1.0-c0)*c44a+c0*c44b;

    y100=c11+c12-2.*(c12*c12/c11);    //弾性関数Y<100>

    Da=1.0e-4*Math.exp(-294000.0/RR/temp);    //Feの自己拡散係数
    Db=2.0e-5*Math.exp(-308000.0/RR/temp);    //Crの自己拡散係数
    Dab=Db/Da;                                //Feの自己拡散係数で規格化
    Mc0=(c0+Dab*(1.-c0))*c0*(1.0-c0);         //易動度関数

//*************************************************************************
    prog.ini_comp_field();  //乱数によって固溶体状態の初期濃度場を設定
    //prog.datin(); //ファイルから初期濃度場を読み込み、および合金組成の再計算

//***  相分解の時間発展過程の計算スタート  ***************************************
  while(time1<=time1max){

    if((((int)(time1) % 200)==0)){
      System.out.println(time1); prog.repaint(); //所定カウント数おきに濃度場描画
    }

    if((((int)(time1) % 500)==0)){ prog.datsave(); }  //所定カウント数おきに濃度場保存
    //   現在この行の前に//があり文全体をコメント文にしている。//を外せばデータの保存
    //   が行われるが、数値データは非常に大きくなる場合があるので、//を外す場合には、
    //   ハードディスクの空き容量やデータ保存量等を良く確認した後、//を外されたい。

    //if(time1==1000.0){ prog.datsave(); } //所定カウント数の時に濃度場保存

//******[拡散ポテンシャルの計算]*********************************************
    for(i=0;i<=ndm;i++){
      for(j=0;j<=ndm;j++){
        ip=i+1; im=i-1; jp=j+1; jm=j-1;
        if(i==ndm){ip=0;}   if(i==0){im=ndm;}
        if(j==ndm){jp=0;}   if(j==0){jm=ndm;}

        c2=ch[i][j];        c1=1.0-c2;
        c2ip=ch[ip][j]; c2im=ch[im][j]; c2jp=ch[i][jp]; c2jm=ch[i][jm];

        mu_chem=L0*(c1-c2)+Math.log(c2)-Math.log(c1);       //化学ポテンシャル差
        mu_surf=-2.0*kapa_c*(c2ip+c2im+c2jp+c2jm-4.0*c2); //濃度勾配のポテンシャル
        mu_str=2.0*eta*eta*y100*(c2-c0);                    //弾性ポテンシャル

        ck[i][j]=mu_chem+mu_str+mu_surf;                    //拡散ポテンシャル
      }
    }
```

```
//******[濃度場の時間変化(非線形拡散方程式の差分解陽解法)]*****************************
      for(i=0;i<=ndm;i++){
         for(j=0;j<=ndm;j++){
            ip=i+1; im=i-1;   jp=j+1;   jm=j-1;
            if(i==ndm) {ip=0;} if(i==0) {im=ndm;}
            if(j==ndm) {jp=0;}   if(j==0) {jm=ndm;}
            ck0=ck[i][j]; ckip=ck[ip][j]; ckim=ck[im][j];ckjp=ck[i][jp]; ckjm=ck[i][jm];

            ck2dev=(ckip+ckim+ckjp+ckjm)-4.0*ck0;      //拡散ポテンシャルの２階微分
            cddtt=Mc0*ck2dev;                          //非線形拡散方程式
            //ch2[i][j]=ch[i][j]+cddtt*delt;           //濃度場の時間発展
            ch2[i][j]=ch[i][j]+cddtt*delt+1.0e-03*(2.0*Math.random()-1.0);
                                                       //濃度場の時間発展（濃度揺らぎ導入）
         }
      }

//*** [濃度場の収支の補正] ***************************************************
//***  数値計算であるので濃度場の収支の補正を行う（実際には毎ステップ行う必要はない）。] ****
      sumc=0.;
      for(i=0;i<=ndm;i++){
         for(j=0;j<=ndm;j++){
            sumc=sumc+ch2[i][j];
         }
      }
      dc0=sumc/(double)nd/(double)nd-c0;

      for(i=0;i<=ndm;i++){
         for(j=0;j<=ndm;j++){
            ch[i][j]=ch2[i][j]-dc0;
            if(ch[i][j]>=1.0){ch[i][j]=1.0-1.0e-6;}
            if(ch[i][j]<=0.0){ch[i][j]=1.0e-6;}
         }
      }

//******[時間増加]*************************************************
      time1=time1+1.0;
         //try{ thread1.sleep(100); }         // 0.01 seconds
         //catch( InterruptedException e){ }
   }//while
}//main

// ****  初期濃度場の設定 ***************************************************
   public void ini_comp_field(){
      int i, j;
      double rnd0, fac1;

      fac1=0.01; //最大1%の初期濃度揺らぎ
      for(i=0;i<=ndm;i++){
         for(j=0;j<=ndm;j++){
            rnd0=2.0*Math.random()-1.0;    ch[i][j]=c0+rnd0*fac1;
         }
      }

   }

// ***** update関数のオーバーライド（再描画時における画面のチラツキ防止）*******
```

```
    public void update(Graphics g){paint(g);}

// **** 濃度場の描画 ***************************************************
    public void paint(Graphics g){
        bg.clearRect(0, 0, width, height);
        bg.setColor(Color.lightGray);

        int i, j, ii, jj;
        int icol;
        int ixmin=0, iymin=0, igx, igy, irad0;
        int ixmax, iymax;
        double c, x, xmax, xmin, y, ymax, ymin, rad0;

        xmin=0.0; xmax=1.0;    ymin=0.0; ymax=1.0;
        ixmax=xwidth;    iymax=yheight;

        //System.out.println(time1);
        rad0=1.0/(double)nd/2.0;
        irad0=1+(int)( ((double)ixmax-(double)ixmin)/(xmax-xmin)*rad0 );

        for(i=0;i<=nd;i++){
           for(j=0;j<=nd;j++){
              x=1.0/(double)nd*(double)i+rad0;
              igx=(int)( ((double)ixmax-(double)ixmin)*(x-xmin)/(xmax-xmin)+(double)ixmin );
              y=1.0/(double)nd*(double)j+rad0;
              igy=(int)( ((double)iymax-(double)iymin)*(y-ymin)/(ymax-ymin)+(double)iymin );
              ii=i; jj=j;
              if(i==nd){ii=0;}    if(j==nd){jj=0;}
              //icol=(int)(255.0*ch[ii][jj]);
              icol=(int)(255.0*(1.0-ch[ii][jj]));
              if(icol>=255){icol=255;}    if(icol<=0){icol=0;}
              bg.setColor(new Color(icol,icol,icol));
              bg.fillRect(insetx+igx-irad0,insety+igy-irad0, irad0*2, irad0*2);
              //bg.setColor(Color.blue);
              //bg.setColor(new Color(0,0,icol));
           }
        }
        g.drawImage(buff, 0, 0, this);
    }

//************[濃度場データの保存]************************************
    private void datsave(){
      int   i,j;

      try{
         PrintWriter outfile= new PrintWriter(
            new BufferedWriter(new FileWriter("test.dat", true)) );

         outfile.println(time1);
         //outfile.println(time1+'¥n');
         for(i=0;i<=ndm;i++){
            for(j=0;j<=ndm;j++){
               outfile.println(ch[i][j]);  //濃度場の書き込み
            }
         }
         outfile.close();
      }
```

```
      catch(Exception e){System.out.println(e);System.exit(1);}
    }

//************[濃度場データの読込み]**********************************
    private void datin(){
      int   i, j;
      double sumc;
      String s_data, s_time1;

      try{
        BufferedReader infile=new BufferedReader(new FileReader("ini000.dat"));
        try{
          s_time1=infile.readLine();   time1=new Double(s_time1).doubleValue();
          for(i=0;i<=ndm;i++){
            for(j=0;j<=ndm;j++){
              s_data=infile.readLine();   ch[i][j]=new Double(s_data).doubleValue();
            }
          }
        }
        catch(Exception e){System.out.println(e); System.exit(1);}
        finally{infile.close();}
      }
      catch(Exception e){System.out.println(e); System.exit(1);}

      sumc=0.;
      for(i=0;i<=ndm;i++){
        for(j=0;j<=ndm;j++){
          sumc=sumc+ch[i][j];
        }
      }
      c0=sumc/(double)nd/(double)nd;

    }

//*************************************************************
}//class FeCr2D_save

//****以上、プログラム終わり  ************************************
```

・Java アプリケーションプログラム(RegSol1D.java)

```
//*************************************************************
import java.awt.*;
import java.awt.event.*;
import java.io.*;

public class RegSol1D extends Frame{

   static int ND=1024;              //組織１辺の分割数
   static int nd=ND, ndm=ND-1;
   static int width, height, insetx, insety, xwidth, yheight;
   static double RR=8.3145;      //ガス定数
   static double [] ch=new double[ND]; //組織内の濃度データ配列

   //グローバル変数化
```

```
   static double c0, temp, time1;       //合金組成、温度、計算時間
   Image buff;
   static Graphics bg, g;

   //*************** コンストラクタ ***************************
   public RegSol1D(){
      xwidth=800; yheight=200; insetx=4; insety=30;
      width=xwidth+insetx*2;    height=yheight+insetx+insety;
      setSize(width, height);
      setBackground(Color.lightGray);
      setVisible(true);
      buff=this.createImage(width, height);
      bg=buff.getGraphics();
      addWindowListener(new WindowAdapter(){
         public void windowClosing(WindowEvent e){ System.exit(0); }
      });
   }

//*************** main ****************************
   public static void main(String[] args){

      RegSol1D prog=new RegSol1D();

      double [] ck = new double[ND];          //拡散ポテンシャル
      double [] ch2 = new double[ND];         //組織内の濃度データ予備配列
      double mu_chem, mu_surf, mu_str;
      int    i, j, k, l;                      //整数
      double c1, c2;                          //濃度(溶媒:1, 溶質:2)
      double al, time1max, delt;
      double a01, atomNo;                     //格子定数と、単位胞内の原子数
      double vm0;                             //モル体積
      double L0;                              //原子間相互作用パラメータ
      double c11, c12, c44;                   //合金の平均弾性率
      double eta, y100, Astr;                 //格子ミスマッチ, 弾性関数
      double kapa_c;
      double Mc0;                             //易動度関数とその微分
      double cddtt;                           //濃度の増分
      double b1;                              //差分ブロックサイズ

      int ip, im;                             //整数（i+1, i-1）
      double c2ip, c2im;          //差分ブロックにおいて c2 を中心に、その左右の濃度
      double ck0, ckip, ckim;     //差分ブロックにて ck0 を中心にその左右の拡散ポテンシャル
      double sumc, dc;            //濃度場の総和、平均組成からのずれ

      String s_temp, s_c0, s_delt;

//-------------------------------------------------------------------------
      BufferedReader input=new BufferedReader(new InputStreamReader(System.in));

      s_c0="0.4";
      try{ System.out.print("c0(0.4) =    ");    s_c0=input.readLine(); }
      catch(IOException e){ System.out.println("Exception : "+e); }
      c0=new Double(s_c0).doubleValue();

      s_temp="1000.0";
```

```
    //try{ System.out.print("temp(1000.0)/K =    ");   s_temp=input.readLine(); }
    //catch(IOException e){ System.out.println("Exception : "+e); }
    temp=new Double(s_temp).doubleValue();

    s_delt="0.05";
    try{ System.out.print("del_time1(0.05) =    ");   s_delt=input.readLine(); }
    catch(IOException e){ System.out.println("Exception : "+e); }
    delt=new Double(s_delt).doubleValue();

//**** 各種定数の設定 ************************************************************

    al=500.0;               //(nm)
    al=al*1.0e-9;           //(m)に変換
    b1=al/(double)nd;       //差分１ブロックのサイズ

    time1=0.0;                  //スタート時間(ループの回数)
    time1max=1.0e+08+1.0;   //計算打切時間(ループの回数)

    a01=0.28664;    //格子定数(bcc Fe)
    atomNo=2.0;     //単位胞内の原子数
    a01=a01*1.0e-09;                        //Fe の格子定数(m に換算)
    vm0=6.02e+23*a01*a01*a01/atomNo;    //モル体積

    L0=2.5e+04;     //原子間相互作用パラメータ（J/mol）
    L0=L0/RR/temp;                          //原子間相互作用パラメータ, RT で無次元化

    eta=0.02; //格子ミスマッチ
    c11=2.3310e+11;   c12=1.3544e+11;   c44=1.1783e+11;   //bccFe の弾性定数(Pa)
    c11=c11*vm0/RR/temp;   c12=c12*vm0/RR/temp;   c44=c44*vm0/RR/temp; //RT で無次元化
    y100=c11+c12-2.0*(c12*c12/c11);     //弾性関数 Y<100>
    Astr=eta*eta*y100;

    kapa_c=5.0e-15/b1/b1/RR/temp;       //濃度勾配エネルギー係数、(b1^2*RR*temp)で無次元化

    Mc0=c0*(1.0-c0);

//*****************************************************************************
    prog.ini_comp_field();  //乱数によって固溶体状態の初期濃度場を設定
    //prog.datin(); //ファイルから初期濃度場を読み込み、および合金組成の再計算

//*** 相分解の時間発展過程の計算スタート ***************************************
  while(time1<=time1max){

    if((((int)(time1) % 100)==0)){
      System.out.println(time1); prog.repaint(); //所定カウント数おきに濃度場描画
    }

    //if((((int)(time1) % 200)==0)){ prog.datsave(); } //所定カウント数おきに濃度場保存
    //   現在この行の前に//があり文全体をコメント文にしている。//を外せばデータの保存
    //   が行われるが、数値データは非常に大きくなる場合があるので、//を外す場合には、
    //   ハードディスクの空き容量やデータ保存量等を良く確認した後、//を外されたい。

    //if(time1==1000.0){ prog.datsave(); }  //所定カウント数の時に濃度場保存
```

```
//******[拡散ポテンシャルの計算]*****************************************************
   for(i=0;i<=ndm;i++){
      ip=i+1; im=i-1;   if(i==ndm){ip=0;}   if(i==0){im=ndm;}

      c2=ch[i];        c1=1.0-c2;   c2ip=ch[ip];  c2im=ch[im];

      mu_chem=L0*(c1-c2)+Math.log(c2)-Math.log(c1);    //化学ポテンシャル差
      mu_surf=-2.0*kapa_c*(c2ip+c2im-2.0*c2);          //濃度勾配のポテンシャル
      mu_str=2.0*Astr*(c2-c0);                         //弾性ポテンシャル

      ck[i]=mu_chem+mu_str+mu_surf;                    //拡散ポテンシャル

   }

//******[濃度場の時間変化]************************************
   for(i=0;i<=ndm;i++){
      ip=i+1; im=i-1;   if(i==ndm){ip=0;} if(i==0){im=ndm;}
      ck0=ck[i];   ckip=ck[ip];   ckim=ck[im];
      cddtt=Mc0*(ckip+ckim-2.0*ck0);    //非線形拡散方程式
      //ch2[i]=ch[i]+cddtt*delt;              //濃度場の時間発展
      ch2[i]=ch[i]+cddtt*delt+1.0e-03*(2.0*Math.random()-1.0);
                                            //濃度場の時間発展（濃度揺らぎ導入）
   }

//*** [濃度場の収支の補正] ***********************************************************
//***  数値計算であるので濃度場の収支の補正を行う（実際には毎ステップ行う必要はない）。] ****
   sumc=0.;
   for(i=0;i<=ndm;i++){ sumc=sumc+ch2[i]; }
   dc=sumc/(double)nd-c0;

   for(i=0;i<=ndm;i++){
      ch[i]=ch2[i]-dc;
      if(ch[i]>=1.0){ch[i]=1.0-1.0e-06;}
      if(ch[i]<=0.0){ch[i]=1.0e-06;}
   }

//******[時間増加]*******************************************
      time1=time1+1.0;
         //try{ thread1.sleep(100); }         // 0.01 seconds
         //catch( InterruptedException e){ }
   }//while
}//main

// ****  初期濃度場の設定  ***************************************************
   public void ini_comp_field(){
      int i;
      double rnd0, fac1;

      fac1=0.01; //最大 1%の初期濃度揺らぎ
      for(i=0;i<=ndm;i++){
         rnd0=2.0*Math.random()-1.0;    ch[i]=c0+rnd0*fac1;
      }

   }
```

```
// ***** update 関数のオーバーライド（再描画時における画面のチラツキ防止）*******
    public void update(Graphics g){paint(g);}

// ****  濃度プロファイルの描画  ***************************************************
    public void paint(Graphics g){
        bg.clearRect(0, 0, width, height);
        bg.setColor(Color.lightGray);

        int ixmax=xwidth, iymax=yheight;
        int i, j, ii, i1, ii1, i2, ii2, nnn;
        double col;
        int ixmin=0, iymin=0, igx1, igy1, igx2, igy2, irad0;
        double cmax, cmin, dx, dy;
        int idx, idy;
        double c, x, xmax, xmin, y, ymax, ymin, rad0;
        double gx1, gy1, gx2, gy2;

        xmin=0.; xmax=1.;    ymin=0.; ymax=1.;
        cmax=1.0; cmin=0.0;
        dx=0.1; dy=0.1;
        idx=(int)(0.1*ixmax);
        idy=(int)(0.1*iymax);

        bg.setColor(Color.white);
        bg.fillRect(insetx, insety, xwidth, yheight);
        bg.setColor(Color.black);
        bg.drawRect(insetx, insety, xwidth, yheight);

        bg.setColor(Color.lightGray);
        for(i=0;i<=ixmax;i+=idx){ bg.drawLine(insetx+i, insety+iymin, insetx+i, insety+iymax); }
        for(i=0;i<=iymax;i+=idy){ bg.drawLine(insetx+ixmin, insety+i, insetx+ixmax, insety+i); }

        rad0=1.0/(double)nd/2.0;
        irad0=1+(int)( ((double)ixmax-(double)ixmin)/(xmax-xmin)*rad0 );

        bg.setColor(Color.red);
        for(i=0;i<=ndm;i++){
          i1=i; i2=i+1;
          gx1=1./(double)nd*(double)i1+rad0;
          gx2=1./(double)nd*(double)i2+rad0;
          ii1=i1; ii2=i2;    if(i==ndm){ii2=0;}

          gy1=(ch[ii1]-cmin)/(cmax-cmin);
          gy2=(ch[ii2]-cmin)/(cmax-cmin);

          igx1=(int)( ((double)ixmax-(double)ixmin)*(gx1-xmin)/(xmax-xmin)+(double)ixmin );
          igy1=(int)( (double)iymin+(double)iymax
                        -(((double)iymax-(double)iymin)/(ymax-ymin)*(gy1-ymin)+(double)iymin) );
          igx2=(int)( ((double)ixmax-(double)ixmin)*(gx2-xmin)/(xmax-xmin)+(double)ixmin );
          igy2=(int)( (double)iymin+(double)iymax
                        -(((double)iymax-(double)iymin)/(ymax-ymin)*(gy2-ymin)+(double)iymin) );

          bg.drawLine(insetx+igx1, insety+igy1, insetx+igx2, insety+igy2);
        }
        g.drawImage(buff, 0, 0, this);
    }
```

```
//************[濃度場データの保存]**********************************
    private void datsave(){
     int   i;

     try{
        PrintWriter outfile= new PrintWriter(
           new BufferedWriter(new FileWriter("test.dat", true)) );

        outfile.println(time1);
        //outfile.println(time1+'¥n');
        for(i=0;i<=ndm;i++){
           outfile.println(ch[i]);  //濃度場の書き込み
        }
        outfile.close();
     }
     catch(Exception e){System.out.println(e);System.exit(1);}
    }

//************[濃度場データの読込み]**********************************
    private void datin(){
     int   i;
     double sumc;
     String s_data, s_time1;

     try{
        BufferedReader infile=new BufferedReader(new FileReader("ini000.dat"));
        try{
           s_time1=infile.readLine();    time1=new Double(s_time1).doubleValue();
           for(i=0;i<=ndm;i++){
              s_data=infile.readLine();    ch[i]=new Double(s_data).doubleValue();
           }
        }
        catch(Exception e){System.out.println(e); System.exit(1);}
        finally{infile.close();}
     }
     catch(Exception e){System.out.println(e); System.exit(1);}

     sumc=0.;    for(i=0;i<=ndm;i++){ sumc=sumc+ch[i]; }    c0=sumc/(double)nd;
    }

//*************************************************************
}//class RegSol1D

//****以上、プログラム終わり  ************************************
```

********** 参考 **********

・線形スピノーダル分解理論（１次元）

スピノーダル分解理論において、全自由エネルギーG_{sys}は、

$$G_{sys} = \frac{1}{L}\int_x \left[G_c(c) + \eta^2 Y_{<hkl>}(c-c_0)^2 + \kappa\left(\frac{\partial c}{\partial x}\right)^2 \right] dx \qquad (1)$$

$$E_{str} = \frac{1}{L}\int_x \eta^2 Y_{<hkl>}(c-c_0)^2 dx \tag{2}$$

$$E_{surf} = \frac{1}{L}\int_x \kappa\left(\frac{\partial c}{\partial x}\right)^2 dx \tag{3}$$

と表現される(１次元)。式(1)右辺の積分内を F とすると、$c, x, (\partial c/\partial x)$ の３つを独立変数として変分原理を適用することによって、オイラー方程式に基づき、拡散ポテンシャルは、

$$\chi = \frac{\delta G_{system}}{\delta c} = \frac{\partial F}{\partial c} - \frac{d}{dc}\left\{\frac{\partial F}{\partial(\partial c/\partial x)}\right\} = \frac{\partial G_c(c)}{\partial c} + 2\eta^2 Y_{<hkl>}(c-c_0) - 2\kappa\left(\frac{\partial^2 c}{\partial x^2}\right) \tag{4}$$

と計算される。これを以下の発展方程式に代入すると（濃度ゆらぎ項は簡単のため無視)、拡散方程式は、

$$\begin{aligned}\frac{\partial c}{\partial t} &= \frac{\partial}{\partial x}\left\{M(c_A,c_B)\left(\frac{\partial \chi}{\partial x}\right)\right\} \\ &= \frac{\partial}{\partial x}\left\{M(c_A,c_B)\left(\frac{\partial^2 G_c(c)}{\partial c^2} + 2\eta^2 Y_{<hkl>}\right)\left(\frac{\partial c}{\partial x}\right)\right\} - 2\frac{\partial}{\partial x}\left\{M(c_A,c_B)\kappa\left(\frac{\partial^3 c}{\partial x^3}\right)\right\} \\ &= \frac{\partial}{\partial x}\left\{\tilde{D}\left(\frac{\partial c}{\partial x}\right)\right\} - 2\frac{\partial}{\partial x}\left\{\tilde{K}\left(\frac{\partial^3 c}{\partial x^3}\right)\right\}\end{aligned} \tag{5}$$

と導かれる。ここで、

$$\tilde{D} \equiv M(c_A,c_B)\left(\frac{\partial^2 G_c(c)}{\partial c^2} + 2\eta^2 Y_{<hkl>}\right), \tilde{K} \equiv M(c_A,c_B)\kappa \tag{6}$$

と置いた。$\tilde{D}$ は整合相分解における相互拡散係数に他ならない。以上からCahn-Hilliardの非線形拡散方程式は、

$$\frac{\partial c}{\partial t} = \frac{\partial}{\partial x}\left\{\tilde{D}\left(\frac{\partial c}{\partial x}\right)\right\} - 2\frac{\partial}{\partial x}\left\{\tilde{K}\left(\frac{\partial^3 c}{\partial x^3}\right)\right\} \tag{7}$$

と与えられる。線形スピノーダル分解を扱った教科書では、$M(c_A,c_B)$ を定数と仮定して $\tilde{K}$ を微分の外に出している。

以下、線形スピノーダル分解理論に基づき、スピノーダル分解の優先波長と、温度の関係式を導いてみよう。まず、式(7)(6)を次のように近似する。

$$\frac{\partial c}{\partial t} = \frac{\partial}{\partial x}\left\{\tilde{D}\left(\frac{\partial c}{\partial x}\right)\right\} - 2\frac{\partial}{\partial x}\left\{\tilde{K}\left(\frac{\partial^3 c}{\partial x^3}\right)\right\} \cong \tilde{D}\left(\frac{\partial^2 c}{\partial x^2}\right) - 2\tilde{K}\left(\frac{\partial^4 c}{\partial x^4}\right) \tag{8}$$

$$\begin{aligned}&\tilde{D} \equiv M(c_A,c_B)\left(\frac{\partial^2 G_c(c)}{\partial c^2} + 2\eta^2 Y_{<hkl>}\right) \cong M\left(-2\Omega + \frac{RT}{c_0(1-c_0)} + 2\eta^2 Y_{<hkl>}\right) \\ &\tilde{K} \equiv M(c_A,c_B)\kappa \cong M\kappa\end{aligned} \tag{9}$$

1 次元濃度場を cos 波にて、

$$c(x,t) = c_0 + Q(k,t)\cos(kx) \tag{10}$$
$$\frac{\partial c}{\partial t} = \frac{\partial Q}{\partial t}\cos(kx),\ \left(\frac{\partial^2 c}{\partial x^2}\right) = -k^2 Q(k,t)\cos(kx),\ \left(\frac{\partial^4 c}{\partial x^4}\right) = k^4 Q(k,t)\cos(kx)$$

と置く。これらを式(8)に代入すると、

$$\frac{\partial c}{\partial t} = \tilde{D}\left(\frac{\partial^2 c}{\partial x^2}\right) - 2\tilde{K}\left(\frac{\partial^4 c}{\partial x^4}\right)$$
$$\frac{\partial Q}{\partial t}\exp(ikx) = \tilde{D}\{-k^2 Q(k,t)\exp(ikx)\} - 2\tilde{K}\{k^4 Q(k,t)\exp(ikx)\}$$
$$\frac{\partial Q}{\partial t} = (-\tilde{D}k^2 - 2\tilde{K}k^4)Q(k,t) \tag{11}$$
$$Q(k,t) = Q(k,0)\exp\{(-\tilde{D}k^2 - 2\tilde{K}k^4)t\} = Q(k,0)\exp\{R(k)t\}$$
$$\therefore\ c(x,t) = c_0 + Q(k,0)\exp\{R(k)t\}\exp(ikx)$$

を得る。ここで、

$$R(k) \equiv -\tilde{D}k^2 - 2\tilde{K}k^4 \tag{12}$$

は（濃度）振幅拡大係数と呼ばれる。これを波数 k で微分し、0 と置くことによって、スピノーダル分解の優先波数 k_c を求めることができる。

$$\frac{\partial R}{\partial k} = -2\tilde{D}k_c - 8\tilde{K}k_c^3 = 0$$
$$-\tilde{D} - 4\tilde{K}k_c^2 = 0$$
$$k_c^2 = -\frac{\tilde{D}}{4\tilde{K}} = -\frac{M\left(-2\Omega + \dfrac{RT}{c_0(1-c_0)} + 2\eta^2 Y_{<hkl>}\right)}{4M\kappa} \tag{13}$$
$$= \frac{2(\Omega - \eta^2 Y_{<hkl>})c_0(1-c_0) - RT}{4\kappa c_0(1-c_0)} = -\frac{R}{4\kappa c_0(1-c_0)}T + \frac{\Omega - \eta^2 Y_{<hkl>}}{2\kappa}$$

特に、$c_0 = 0.5$ の場合、

$$k_c^2 = \frac{(\Omega - \eta^2 Y_{<hkl>}) - 2RT}{2\kappa} = -\frac{R}{\kappa}T + \frac{(\Omega - \eta^2 Y_{<hkl>})}{2\kappa} \tag{14}$$

となる。式(13)(14)より、優先波数の二乗は時効温度に比例することがわかる。したがって、実験的に温度を変えて、優先波数を測定し、温度に対してプロットすれば、その傾きから濃度勾配エネルギー定数 κ が、その切片から相互作用パラメータ Ω や $\eta^2 Y_{<hkl>}$ の情報を得ることが出来る。さらに線形スピノーダル分解理論では、優先波数が 0（すなわち分解波長が無限大）を与える温度がスピノーダル温度である（式(13)の始めの式にて右辺の分子が 0）ので、上述のグラフの、温度軸（横軸）との切片がスピノーダル温度に対応していることになる。

・ヘッセ行列とスピノーダル線（３元系におけるスピノーダル線の計算式）

３元系の自由エネルギーを $f(c_A,c_B)$ とする。溶質の収支条件から $c_C=1-c_A-c_B$ である。$f(c_A,c_B)$ を合金組成 (c_{0A},c_{0B}) のまわりで、２次のオーダーまでテーラー展開する。

$$f_{\mathrm{II}}(c_A,c_B)=f(c_{0A},c_{0B})+\frac{\partial f_0}{\partial c_A}(c_A-c_{0A})+\frac{\partial f_0}{\partial c_B}(c_B-c_{0B})$$
$$+\frac{1}{2}\left\{\frac{\partial^2 f_0}{\partial c_A^2}(c_A-c_{0A})^2+2\frac{\partial^2 f_0}{\partial c_A\partial c_B}(c_A-c_{0A})(c_B-c_{0B})+\frac{\partial^2 f_0}{\partial c_B^2}(c_B-c_{0B})^2\right\}$$

ヘッセ行列は、

$$\nabla^2 f=\begin{pmatrix}\dfrac{\partial^2 f}{\partial c_A^2} & \dfrac{\partial^2 f}{\partial c_A\partial c_B}\\ \dfrac{\partial^2 f}{\partial c_B\partial c_A} & \dfrac{\partial^2 f}{\partial c_B^2}\end{pmatrix}$$

にて定義される。ナブラは組成場による微分である。組成場をベクトルとして、$\mathbf{c}=(c_A,c_B)$ と表現すると、自由エネルギーは、

$$f_{\mathrm{II}}(\mathbf{c})=f(\mathbf{c}_0)+\left(\nabla f_0,\mathbf{c}-\mathbf{c}_0\right)+\frac{1}{2}\left(\mathbf{c}-\mathbf{c}_0,\nabla^2 f_0\ (\mathbf{c}-\mathbf{c}_0)\right)$$

と表現できる。これは、

$$\nabla^2 f_0\ (\mathbf{c}-\mathbf{c}_0)=\begin{pmatrix}\dfrac{\partial^2 f}{\partial c_A^2} & \dfrac{\partial^2 f}{\partial c_A\partial c_B}\\ \dfrac{\partial^2 f}{\partial c_B\partial c_A} & \dfrac{\partial^2 f}{\partial c_B^2}\end{pmatrix}\begin{pmatrix}c_A-c_{0A}\\ c_B-c_{0B}\end{pmatrix}=\begin{pmatrix}\dfrac{\partial^2 f}{\partial c_A^2}(c_A-c_{0A})+\dfrac{\partial^2 f}{\partial c_A\partial c_B}(c_B-c_{0B})\\ \dfrac{\partial^2 f}{\partial c_B\partial c_A}(c_A-c_{0A})+\dfrac{\partial^2 f}{\partial c_B^2}(c_B-c_{0B})\end{pmatrix}$$

$$\left(\mathbf{c}-\mathbf{c}_0,\nabla^2 f_0\ (\mathbf{c}-\mathbf{c}_0)\right)=\begin{pmatrix}c_A-c_{0A} & c_B-c_{0B}\end{pmatrix}\begin{pmatrix}\dfrac{\partial^2 f}{\partial c_A^2}(c_A-c_{0A})+\dfrac{\partial^2 f}{\partial c_A\partial c_B}(c_B-c_{0B})\\ \dfrac{\partial^2 f}{\partial c_B\partial c_A}(c_A-c_{0A})+\dfrac{\partial^2 f}{\partial c_B^2}(c_B-c_{0B})\end{pmatrix}$$
$$=\frac{\partial^2 f_0}{\partial c_A^2}(c_A-c_{0A})^2+2\frac{\partial^2 f_0}{\partial c_A\partial c_B}(c_A-c_{0A})(c_B-c_{0B})+\frac{\partial^2 f_0}{\partial c_B^2}(c_B-c_{0B})^2$$

より確認できる。

点 (c_{0A},c_{0B}) で、$\nabla f_0=0$ が成り立てば、

$$f_{\mathrm{II}}(\mathbf{c})=f(\mathbf{c}_0)+\left(\nabla f_0,\mathbf{c}-\mathbf{c}_0\right)+\frac{1}{2}\left(\mathbf{c}-\mathbf{c}_0,\nabla^2 f_0\ (\mathbf{c}-\mathbf{c}_0)\right)=f(\mathbf{c}_0)+\frac{1}{2}\left(\mathbf{c}-\mathbf{c}_0,\nabla^2 f_0\ (\mathbf{c}-\mathbf{c}_0)\right)$$

$$\therefore\ \ f_{\mathrm{II}}(\mathbf{c})-f(\mathbf{c}_0)=\frac{1}{2}\left(\mathbf{c}-\mathbf{c}_0,\nabla^2 f_0\ (\mathbf{c}-\mathbf{c}_0)\right)$$

であるので、関数 $f(\mathbf{c})$ は、$\nabla f=0$ となる点で、ヘッセ行列が定値（正値、負値）であれば、極値

（極小値、極大値）を取る（正値：ヘッセ行列の２つの固有値が共に正、負値：ヘッセ行列の２つの固有値が共に負）。したがって、変曲点はヘッセ行列の行列式

$$\begin{vmatrix} \dfrac{\partial^2 f}{\partial c_A^2} & \dfrac{\partial^2 f}{\partial c_A \partial c_B} \\ \dfrac{\partial^2 f}{\partial c_B \partial c_A} & \dfrac{\partial^2 f}{\partial c_B^2} \end{vmatrix} = \left(\frac{\partial^2 f}{\partial c_A^2}\right)\left(\frac{\partial^2 f}{\partial c_B^2}\right) - \left(\frac{\partial^2 f}{\partial c_A \partial c_B}\right)^2 = 0$$

にて定義される。