Lennard-Jonesポテンシャルによる粒界エネルギーの計算

関西学院大学理工学部
情報科学科 西谷研究室 6641 岩倉新太郎

平成23 年2 月21 日

概 要

本研究は Lennard-Jones ポテンシャルを用いて小傾角粒界における粒界エネルギーの数値解を求めることを目的にしている．エネルギー値の計算は [結晶構造のデータの `input` > 結晶緩和 > エネルギー値の計算] の手順で行われるため，本研究では初めに任意の結晶モデルの座標データを生成するプログラムを作り，次に結晶のエネルギー値を計算するプログラムを作成し結晶緩和をしていない状態でエネルギー値を計算できることを確認し，最後に西谷研 B4 松塚の作った brent 法と MC 法の結晶緩和プログラムを実装した結晶構造のエネルギー値を計算する．

目 次

第1章　序論

1.1　小傾角粒界

結晶構造のおける線状の結晶欠陥のことを転位と呼び，転位が境界になっている曲線を転位線と呼ぶことがある [1]．転位には転位線に対し垂直に結晶面がずれる刃状転位と，転位線に対して平行に結晶面がずれる螺旋転位があるが，本研究では刃状転位の集合体である小傾角粒界を元に粒界エネルギーの計算を行う．

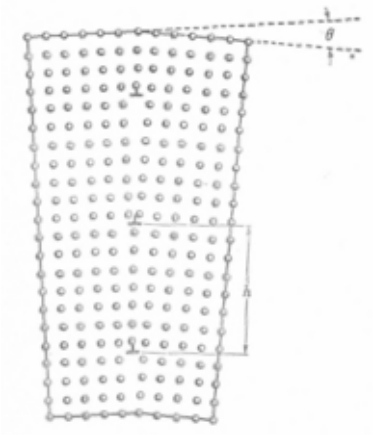

図 1.1: 小傾角粒界 [2]

結晶粒界は転位の集合体を意味する．その中でも角度θの小さいものを小傾角粒界と呼ぶ．

1.2　粒界エネルギー

一般に，結晶粒界の存在する結晶の方が粒界の存在しない完全結晶よりもエネルギーが高い状態にあり，そのエネルギー値の差を単位面積辺りに換算したものを粒界エネルギーと呼ばれている．固体を構成する原子には 2 原子分子と同様の相互作用エネルギーの距離依存性がある．エネルギー値が最も低い距離を平衡原子間距離，エネルギー値を凝集エネルギーと呼ぶ [3]．完全結晶は結晶中の原子

対がすべて平衡原子間距離で並んだ状態にあるため結晶の持つエネルギー値が最も低く安定した状態であるが，粒界結晶は粒界によって原子配置 (原子間距離) が乱れているためにエネルギー値が上昇している．

1.3 Lennerd-Jones ポテンシャル

Lennerd-Jones ポテンシャルはこのような 2 つの原子間の相互作用エネルギーを直感的に表現したポテンシャルモデルの中で最も標準的に使用されているものであり，本研究はこれを用いて粒界結晶の持つエネルギー値を数値間で求めることを目的としている．

第2章　Latticeの生成

エネルギー値の計算を行うために実際の小傾角粒界に類似した結晶構造モデルを用意しなくてはいけない．任意の回転角θやモデルの大きさを持つ結晶を使って結晶緩和，あるいはエネルギー値を計算する作業を効率良く行うために，コマンドラインから指定した大きさと回転角θを持つ原子位置ファイルを生成するプログラムを作る必要がある．

2.1　元ファイルの読み込み

2.1.1　POSCAR.txt

以下に結晶モデル座標値の元ファイル POSCAR.txt を読み込む部分を記述する．

```
file = open('POSCAR.txt')
pos=[]
while line = file.gets do
  if /Direct/ =~ line
    while line = file.gets do
      pos0=line.chomp.split(" ")
      pos1=[]
      pos0.each do |p|
        pos1 << p.to_f
      end
      pos << pos1
    end
  end
end
file.close
```

このプログラムでは元ファイルPOSCAR.txtを読み込み，配列posとして帰す，という作業をしている．データソースに用いられるPOSCAR.txtは第一原理計算ソフトVASPの原子位置入力ファイルとして元から用意されていたものである．POSCAR.txtの中身は以下である．

```
(Al)4  (Fm-3m) ~ 533 initial: 532 minimi\\
   1.00000000000000\\
  5.718886265000000   0.0000000000000000  0.0000000000000000\\
  0.000000000000000   5.7188862650000000  0.0000000000000000\\
  0.000000000000000   0.0000000000000000  5.7188862650000000\\
   1\\
Direct\\
  0.0000000000000000  0.0000000000000000  0.0000000000000000\\
  0.0000000000000000  0.5000000000000000  0.5000000000000000\\
  0.5000000000000000  0.0000000000000000  0.5000000000000000\\
  0.5000000000000000  0.5000000000000000  0.0000000000000000\\
```

このPOSCAR.txtのDirectより下の4行を読み込み，結晶モデルの最小単位としたものを配列posとして帰している．各行が原子1個分の座標を示し，数字は左から順にx座標 y座標 z座標を表す．結晶モデルが持つ個別の原子は，それぞれの座標を配列[x,y,z]として位置を記録し，それらの原子をすべて登録した二重配列posを1個の結晶モデルと考える．この時点で配列posには[0.0 0.0 0.0][0.0 0.5 0.5] [0.5 0.0 0.5] [0.5 0.5 0.5]の4個の原子が入っている．

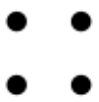

図 2.1: 配列pos平面図

奥行きz軸方向を省略した平面から見るとこのように見える．以下の各セクションではこのような平面図を使って説明する．

2.1.2 size

ARGV[n]にはプログラム実行時にコマンドラインから入力した値や文字が入る．

```
size = {
  :nx => ARGV[0].to_i,
  :ny => ARGV[1].to_i,
  :nz => ARGV[2].to_i,
  :theta => atan(1.0/ARGV[3].to_f),
}
```

nx,ny,nz,thetaをデータ群sizeとしてプログラムの冒頭で取得する．

2.2　結晶モデルの作成

結晶モデルは元ファイルである POSCAR.txt の拡張，回転，平行移動，界面作成という手順で行われる．

2.2.1　POSCAR.txt からの拡張

作られた結晶モデルは隣接したセルと連続した状態であるという仮定の下で使用される．(エネルギー値を計算する際には周期的境界条件が適用されることを前提としている．) そのため，小傾角 (回転角 θ) が小さくなるほど，大きい結晶モデルを使わなければ粒界 1 個分を 1 セルの中に入れることができない．プログラムでは作成する結晶モデルの大きさを常に回転角 θ に合わせて拡張出来なければならない．

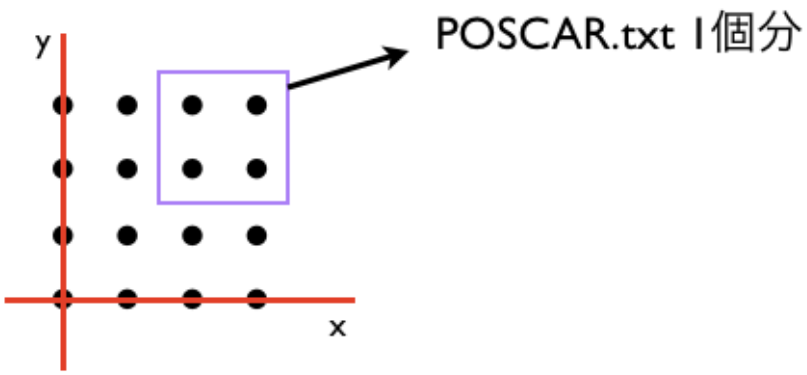

図 2.2: nx,ny,nz=2,2,2 の結晶モデル

前の節で作成した結晶モデル配列 pos を x,y,z 方向に指定した size(以下 nx,ny,nz) 個分，長くした直方体を作成する．ここでは nx,ny,nz=2,2,2 等と表現する．結晶モデルの拡張には以下のメソッドを使用する．

```
def extend(pos,size)
 atom_pos=[]
 for i in 0..size[:nx] -1 do
   for j in 0..size[:ny] -1 do
     for k in 0..size[:nz] -1 do
       pos.each do |p0|
        p1=[]
        p1[0] = p0[0]+1.0*i
        p1[1] = p0[1]+1.0*j
```

```
        p1[2] = p0[2]+1.0*k
        atom_pos << p1
      end
    end
  end
 end
 return atom_pos
end
```

このメソッドでは結晶モデル配列 pos とデータ群 size を引数にして，pos を大きさ nx ny nz に拡張して帰す.

2.2.2 回転

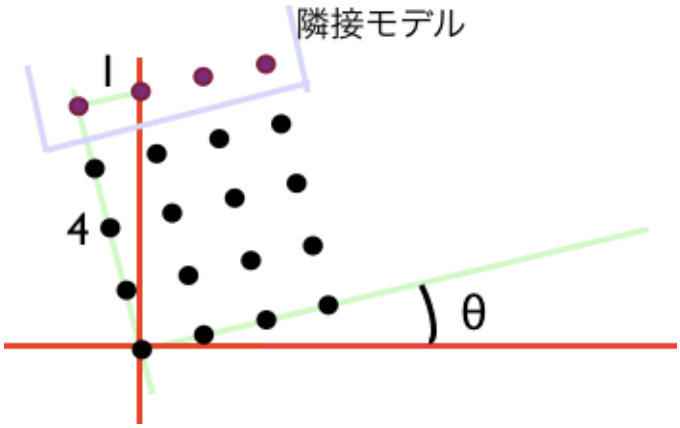

図 2.3: pos を回転角 θ =arctan(1/4) で原点を軸に回転

拡張させた結晶モデルを角度 θ で原点を中心に回転させる．このとき，結晶モデルの隣接セルの一番外側の原子 (紫色の●で示した部分) が x=0 の直線上に来なければいけない．回転角 θ は arctan(1/n) を使って定義する.

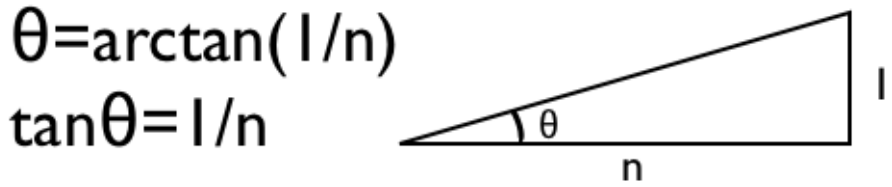

図 2.4: θ =arctan(1/n) のとき tan θ =1/n

θ =arctan(1/n) のとき tan θ =1/n である．コマンドラインからはこの n を指定して θ を決定する．n の値は ny の数値を 2 倍したものと対応している．θ =arctan(1/n) の回転角を持つ結晶モデルは ny の値が n/2 以上の大きさでなければ作成することが出来ない．図では ny=2 の結晶モデルを θ =arctan(1/4) で回転させている．

```
def rotation(pos,size)
  pos2=[]
  pos.each do |pos1|
    tmp = []
    tmp << pos1[0]*cos(size[:theta]) + pos1[1]*(-sin(size[:theta]))
    tmp << pos1[0]*(sin(size[:theta])) + pos1[1]*cos(size[:theta])
    tmp << pos1[2]
    pos2 << tmp
  end
  return pos2
end
```

線形代数の回転の公式を結晶モデル内の原子に適用して帰すメソッドを使う．結晶モデル pos とデータ群である size を引数に使っている．

2.2.3 平行移動

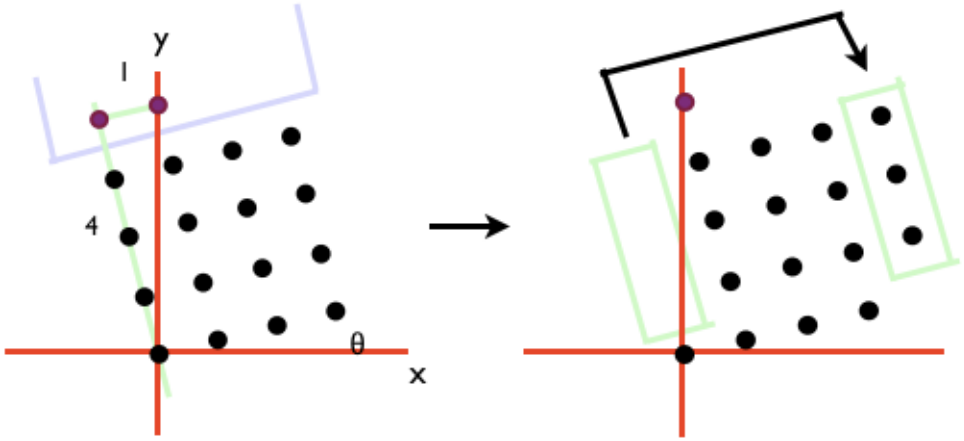

図 2.5: x 軸値が 0 よりも小さい原子を右側に平行移動

結晶モデルを回転させることで，x 座標が 0 よりも小さくなった原子を結晶モデルの右側につける必要がある．以下がメソッドである．

```
def translation(pos,size)
  dx = size[:nx].to_i * cos(size[:theta])
  dy = size[:nx].to_i * sin(size[:theta])
  10 *size[:ny].to_i.times do
    for i in 0..pos.size-1 do
      if pos[i][0] < -0.00001 then
        #    if pos[i][0] < 0 then
        pos[i][0] = pos[i][0]+dx
        pos[i][1] = pos[i][1]+dy
      end
    end
  end
  return pos
end
```

x 座標が 0 以下の全ての原子を x 軸方向に dx y 軸方向に dy 移動させるという意味である．移動距離 dx は `nx*cos` θ，dy は `nx*sin` θ の値に設定する．

2.2.4 界面作成

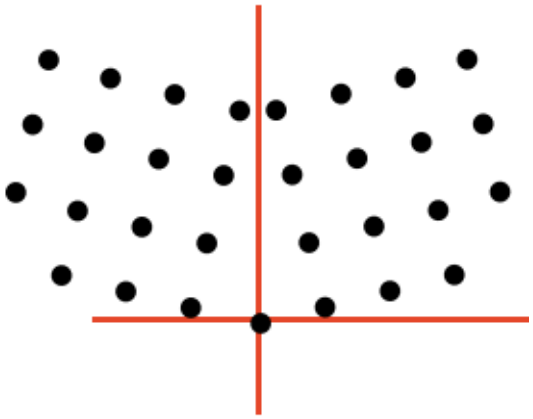

図 2.6: x=0 平面で対象になるように同じものをつける

x=0 の平面を中心にミラーを作成してくっつけ，x=0 平面を界面として扱う．(図 2.6)

```
def mkMirror(pos)
  dummy = Marshal.load(Marshal.dump(pos))
  for i in 0..dummy.size-1 do
    dummy[i][0] = dummy[i][0]-2*dummy[i][0]
  end
  dummy.each do |p|
    pos << p
  end
  pos=Remove_duplicate(pos)
  return pos
end

def Remove_duplicate(pos)
  for i in 0..pos.size-1 do
    if pos[i][0] < 0.0001 && pos[i][0] > -0.0001
      pos[i][0] = 0.0
    end
  end
  pos = pos.uniq
  return pos
end
```

元の結晶モデルのコピーから 2*x の物を引いたものが x=0 を中心とした反対方向向きのモデルであり，元の結晶モデルと足し合わせせ，座標が重複した部分 (x=0 界面上にあるものが 2 つになる) 消去したものが結晶モデルの 1 ユニットセルになる．配列内の同じ要素を消す時には配列の接尾に.uniq をつける．

2.3 プログラムの実行結果

メソッド extend,rotation,toranslation,mkMirror の順番に実行することで作成した結晶モデルを txt ファイルに書き出す．

```
pos=extend(pos,size) //拡張
pos=rotation(pos,size) //回転
pos=translation(pos,size) //平行移動
pos=mkMirror(pos) //界面作成

file = File.open(ARGV[4],'w') //ファイルに書き出し (ファイル名指定)
file.printf("%f \n",size[:nx])
file.printf("%f \n",size[:ny])
file.printf("%f \n",size[:nz])
file.printf("%f \n",ARGV[3].to_f)
file.printf("%s \n","Direct")
for i in 0..pos.size-1 do
  file.printf("%f %f %f\n",pos[i][0],pos[i][1],pos[i][2])
end
file.close
```

コマンドラインから実行ファイル名 nx ny nz n 書込ファイル名 の順番に入力する．nx,ny,nz=1,1,1 arctan=(1/2) のプログラムの実行例を以下に示す．このように作成した結晶モデルを (1,1,1,2) の結晶モデル等と呼ぶこともある．

```
[roota-2:~/ruby/atom] iwakurashintarou% ruby Edit.rb 1 1 1 2 1112.txt
[roota-2:~/ruby/atom] iwakurashintarou% cat 1112.txt
nx= 1.000000
ny= 1.000000
nz= 1.000000
cf= 2.000000
Direct
0.000000 0.000000 0.000000
0.670820 0.894427 0.500000
0.447214 0.223607 0.500000
```

```
0.223607 0.670820 0.000000
-0.670820 0.894427 0.500000
-0.447214 0.223607 0.500000
-0.223607 0.670820 0.000000
```

実行例と作成した結晶モデルの中身である．Direct より下の部分がエネルギー値計算のプログラムで使用される．

第3章　エネルギー値の計算

次にエネルギー値の計算をするプログラムを作成した．Lennerd-Jones ポテンシャルを用いた 2 原子間の相互作用エネルギーを計算し，それを原子内の全ての原子対に対して適用することで結晶モデルの持つエネルギー値を求めることを目的としている．

3.1　L-J-Potential

$$U(r) = 4\ \varepsilon\ [(\sigma/r)^{12} - (\sigma/r)^{6}] \tag{3.1}$$

Lennard Jones Potential は 2 原子間の相互作用エネルギーを求めるのに使用する式である．

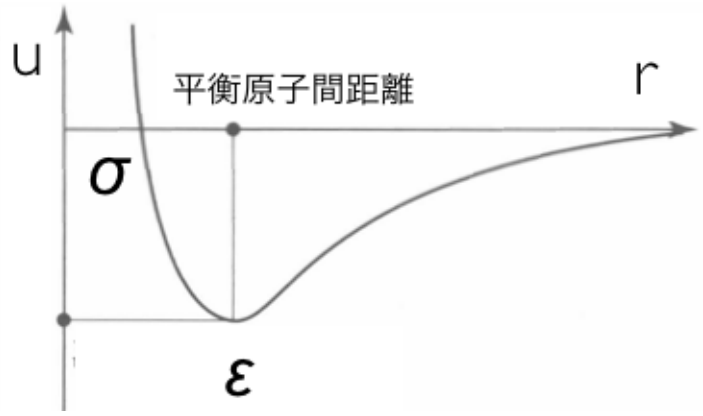

図 3.1: 相互作用エネルギーの距離依存性

相互作用エネルギー U(r) の距離依存性はこのようになる．U(r) は 2 原子間の距離 r におけるエネルギー，グラフの ε は凝集エネルギー (このとき r 方向が平衡原子間距離) を表している．粒界結晶の持つエネルギーは結晶内の全ての原子対についての Lennerd-Jones ポテンシャルの和によって与えられる [4]．プログラムで

は以下のようになる．また，σ/r や ε の値がどのようにして決定されるかについてを以下に示す．

```
A0=1.587401051
E0=-1*4.0/12.0
def ij_energy(r)
  a=1.0/(A0*r)
  return E0*((a**6 )-(a**12)) //Lennerd-Jones
end
```

原子間距離 r を引数に 2 点間の相互作用エネルギーを求める．プログラムの中ではσ/r の値を a と置き，a は 1.0/AO*r と定義されている．AO の値は $d/\sqrt{2*(0.5^2)}$ で決定される．今回は原子間距離 0.5 を基礎に持つ結晶モデルを使っているため，ルートの中で 0.5 が使われている．d には $(1/x)^6-(1/x)^{12}$ を微分した値である $2^{1/6}$ が入り，これを計算して 1.587401051 となった．次に d を $(1/x)^6-(1/x)^{12}$ の式に代入すると 1/4 になるが，これを-1/12 に合わせるために-4/12 をかける必要がある．そこで E0 を-1*4.0/12.0 と定義し全体に掛けている．また，この関数は class Atom の内部で定義されている．class Atom は巻末の付録で示す．

3.2 NeighborList

NeighborList(以下 NL) は近接している物の集合を意味する．エネルギー値を計算するにあたりすべての原子間の相互作用を考慮すると計算に時間がかかりすぎるので，影響力の弱い，原子から遠い位置にある原子との相互作用を遮断する．

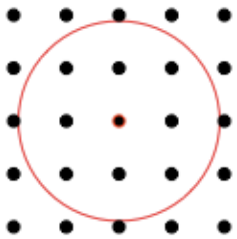

図 3.2: NeighborList

1 個の原子から既定の範囲内にある原子を NL に登録し，NL に登録した原子との間の相互作用エネルギーの総和が 1 つの原子の持つエネルギーであると考える．ここでは隣接原子より 1 つ奥にある原子の手前までを NL 範囲に指定している．

```
def makeNL(rc)
  $atom_list.each do |ai| ai.reset_nl end
  nmax=$atom_list.length-1
  for i in 0..nmax do
    ai=$atom_list[i]
    for j in i+1..nmax do
      aj=$atom_list[j]
     if distance(ai.pos,aj.pos)<rc then
        ai.nl << j
        aj.nl << i
      end
    end
  end
end
```

NL の範囲を引数にして実行すると結晶モデル$atom_list の全原子の NL が作成される．原子 1 つの持つエネルギー値を求める関数を以下に示す,

```
def energy()
  ene=0
  nl.each do |j|
    ene+=ij_energy(distance(@pos,$atom_list[j].pos))
  end
  return ene
end
```

2 点間の相互作用エネルギーを求める関数 ij_energy を NL 内の原子に対して適用し，総和である 1 つの原子の持つエネルギーを求めて帰す．2 点間距離を求める関数 distance は周期的境界条件の説明時に示す.

3.2.1 周期的境界条件 (PeriodicBoundaryCondition)

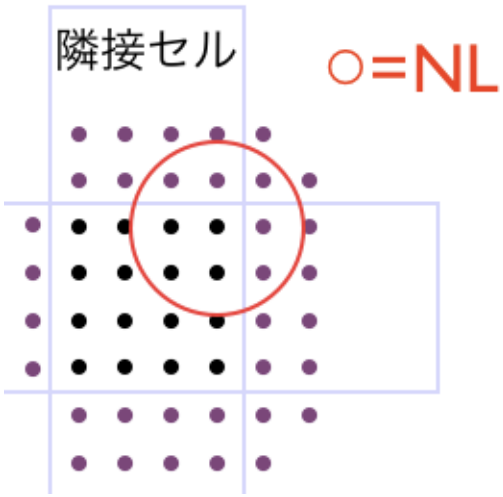

図 3.3: NL が隣接セルに及ぶ

結晶モデルの total-E は隣接する結晶モデルと連続した状態であることを仮定して行われため，結晶モデルの端にある原子の NL に隣接セルの原子が入っている状態のエネルギー値を求める必要がある．

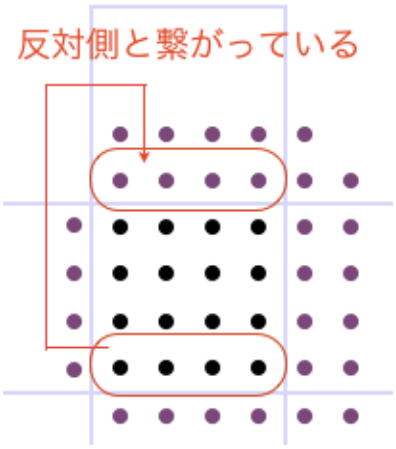

図 3.4: 反対側と繋ぐ

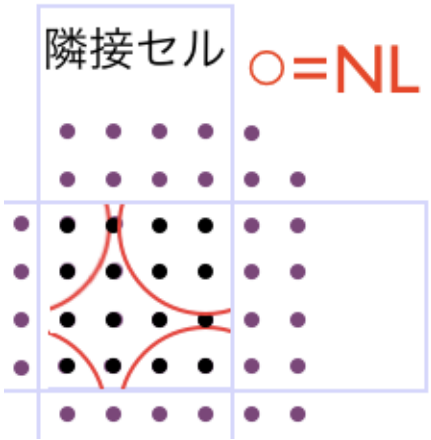

図 3.5: PBC 実装後の NL

原子間距離を求める関数に周期的境界条件 (PBC) の式を実装すると，主体となるセルの端の壁に当たると反対側から出てくるような状況を作ることが出来るため，1 つの結晶モデルを使って同じセルが周囲に無数に並列しているモデルの計算を行うことが出来るようになる．以下は原子 a,b 間の原子間距離を求める関数である．これは NL の作成やエネルギー値の計算に使用される．

```
def distance(a,b)
  tmp=0
  for i in 0..2 do
    x=a[i]-b[i]
    x=x-(x/$L[i]).round*$L[i]
    tmp+=x*x
  end
  return sqrt(tmp)
end
```

この関数に `x=x-(x/$L[i]).round*$L[i]` の 1 行を入れることで周期的境界条件が適応される．

3.2.2 固定境界条件 (DirichletCondition)

x 軸方向に対しては周期的境界条件を適応しないが，固定境界条件を使う．固定境界条件とは未知数の部分を既知の値に固定するという意味である．ここでは x 軸方向の両端の一番外の原子が持つエネルギーを計算せず，除外した原子の持つエネルギー値を 1 に固定する．完全結晶状態の持つ原子のエネルギーは 1 であり，粒界結晶の粒界面から遠い位置にある構造は完全結晶と殆ど同じものだからである．

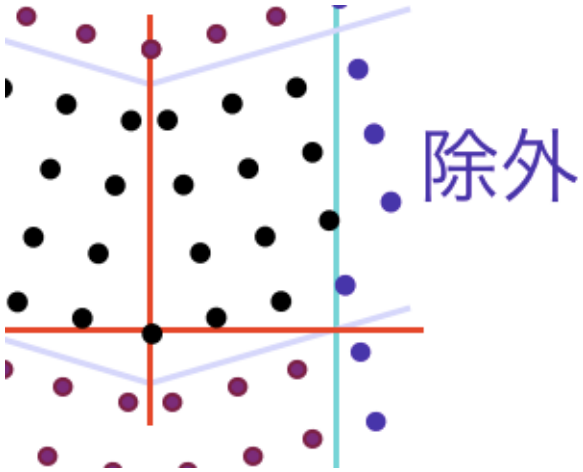

図 3.6: 固定境界条件

縦線よりも外側にある原子を除外する．縦線の内側にある原子は外側の原子とも繋がった状態でエネルギー値を計算することが出来るが，外側の原子が持つエネルギーは全て 1 になる．

```
def mkAtom_include
  atom_include=[]
  max_x = -100
  min_x = 100
  $atom_list.each do |i_a|
    if i_a.pos[0] > max_x then
      max_x = i_a.pos[0]
    elsif i_a.pos[0] < min_x then
      min_x = i_a.pos[0]
    end
  end
  $atom_list.each_with_index do |i_a,idx|
    if i_a.pos[0] < max_x-0.51*cos($theta) && i_a.pos[0] > min_x+0.51*cos($theta)
      atom_include << idx
    end
  end
  return atom_include
end
```

除外された外側の原子以外を atom_include として取得する関数を示す．一番外側の原子座標を max_x,min_x として指定し，その位置から 0.51*cos θ 以内のものを除外している．

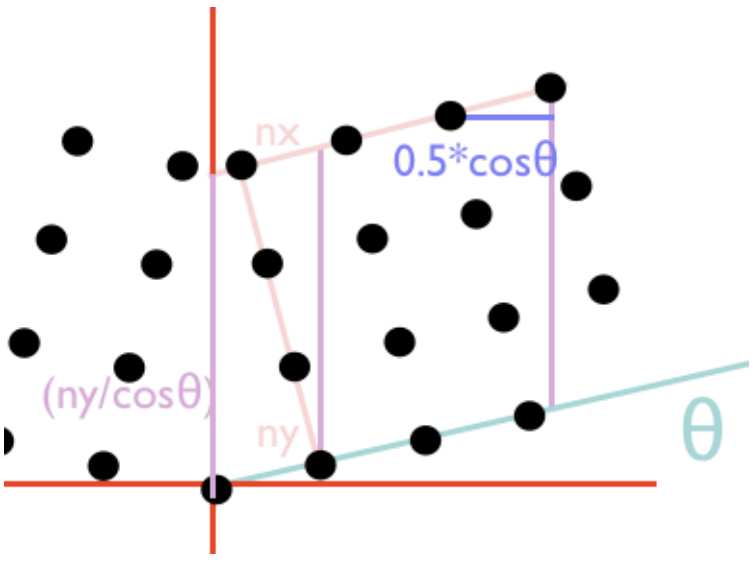

図 3.7: 原子の距離関係

粒界のない位置の原子間距離が 0.5 であるため，外側から 0.51*cos θ 以内を

指定すれば一番外側の原子を取り除くことが出来る．実際は粒界付近の原子以外は緩和で位置が変わることはないのでもう少し深くまでを取り除いても問題はない．

3.3 結晶エネルギー (TotalE)

結晶モデル内にある原子の持つエネルギーの総和なので totalE と呼んでいる．後に粒界エネルギーを求めるのに使用される．以下は totalE を導出するのに用いる関数である．

```
def total_E()
  total_E=0.0
  $atom_include.each do |i|
    total_E+=$atom_list[i].energy()
  end
  return total_E
end
```

固定境界条件で定めた$atom_include 内のすべての原子の持つエネルギーの合計値を帰す．$atom_list[i].energy() は$atom_list の i 番目の原子の持つエネルギーという意味である．

3.4 結晶緩和 (Relax)

結晶粒界が発生すると，結晶構造はよりエネルギー値の低い安定した状態に近づくために原子位置の緩和が起こる．プログラムではこれを外部緩和と内部緩和に分割して行う．

3.4.1 外部緩和 (OuterRelax)

結晶モデルの界面よりも右側をブロック単位で動かすことで原子の安定位置を模索する．

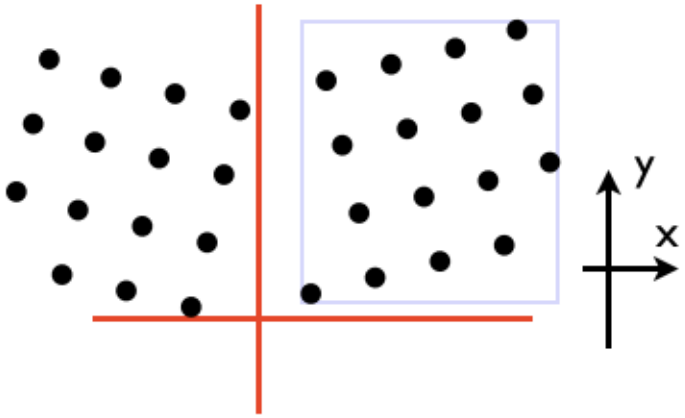

図 3.8: 外部緩和

外部緩和には Brent 法や CG 法である．実際の計算には brent 法を使用した．以下が Brent 法を使った外部緩和による最安定位置 (原点からの x,y,z 方向移動距離) の計算結果を Maple で出力したものある．

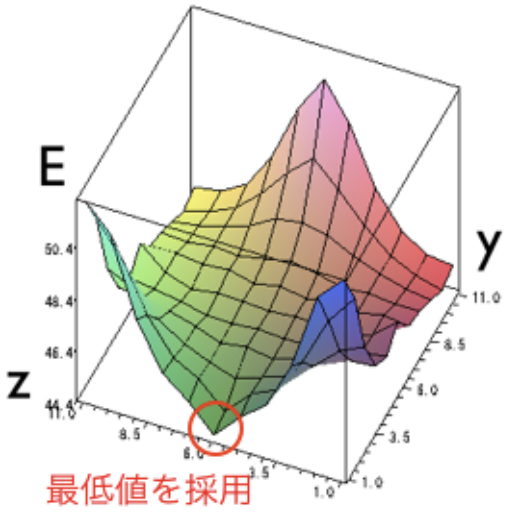

図 3.9: brent 法

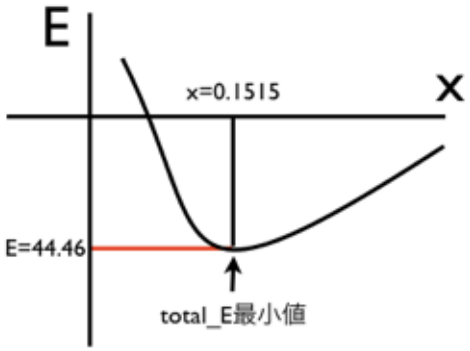

図 3.10: x 軸決定

Brent 法では y,z 軸方向への移動距離が決定されている時の x 軸の最安定位置を計算する．上の表は y,z 軸方向に 0 から 1.0 までの 11 段階を，縦軸はその座標で x 軸方向の移動距離 (0 から 0.3 までの 31 段階) が Brent 法で決定された時の totalE を表している．図 3.9 の例では x,y,z=2,2,2 arctan(1/2) の結晶モデルを使っている．totalE 最低値 44.46 は原点から x,y,z=0.15,0.0,0.5 移動した位置の時である．プログラムは以下のようにして実行する．

```
for $dy in 0..$nrange do
  for $dz in 0..$nrange do
    brent($xrange0,15.0,$xrange1,@x,1.0e-2,0.0)
  end
end
```

プログラム実行時には x 軸方向対象の brent 法プログラムを y,z 方向の必要回数分ループさせる．また，Brent 法では安定位置を求めるだけで直接原子位置を変更するわけではないので，別個に原子位置を移動する必要がある．そのために以下の関数を使う．

```
def block_move(dx,dy,dz)
  $atom_list.each do |atom|
    if atom.pos0[0]>=0.0 then
      atom.pos[0]=atom.pos0[0]+dx
      atom.pos[1]=atom.pos0[1]+dy
      atom.pos[2]=atom.pos0[2]+dz
    end
  end
end
```

3.4.2 内部緩和 (InnerRelax)

結晶モデル内の原子を 1 つずつばらばらに動かして最安定位置を求める手法を内部緩和と呼んでいる．

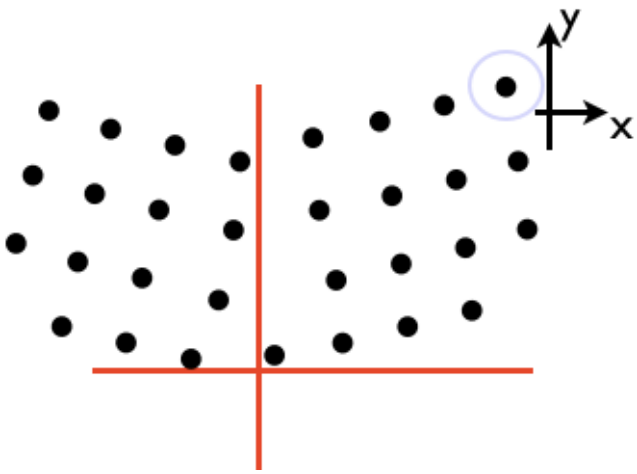

図 3.11: 内部緩和

MC 法や降下法がある．プログラムの中身は松塚が作成した．実際の計算では実行速度に優位性を持つ MC 法を採用した．内部緩和は以下のようにして呼び出される．

```
$inner_rel=$atom_include.size //内部緩和実行回数
def relaxed()
  block_move($min[0],$min[1],$min[2])
  InnerRelax($inner_rel) //内部緩和
  makeNL(0.9) //内部緩和が終わった時点での NL
  $min[3] = total_E()
  pp $min[3]
end
```

関数 blockmove を使ってはじめに外部緩和で求めた位置への移動を実行し．直後に内部緩和を行い直接原子位置を緩和させる．`$min[0-2]` には外部緩和で求めた座標位置が入っている．内部緩和の実行回数は結晶モデルの大きさに比例して多くしている．内部緩和が終わった時点での NL を作成し，totalE を導出する．totalE は`$min` という配列の要素の 1 つとして取得する．`$min[3]` を以後この結晶モデルの持つエネルギー値として扱う．

3.5 粒界エネルギーの角度依存性

エネルギー値の計算は，2 章のプログラムで作成した結晶モデルを読み込み，外部緩和，内部緩和，totalE の計算 の順番で行われる．

3.5.1 エネルギー値

粒界エネルギーは，完全結晶の持つエネルギー値と粒界結晶の持つエネルギー値の差を単位面積辺りに換算したものである．完全結晶は 1 つの原子の持つエネルギーが-1 なので，完全結晶の totalE は [-原子の個数] と同じ値になる．粒界結晶のエネルギー値はプログラムを使って求めたものを使用する．

$$\text{粒界エネルギー} = \frac{totalE - \text{完全結晶の } E}{\text{面積}} \tag{3.2}$$

プログラムでは以下のように記述している．

```
boundary_E = ($min[3]-(-$atom_include.size))/($ny/cos($theta))*$nz)
```

分子では totalE である$min[3] から固定境界条件で完全結晶の外側を削除した原子の個数である (-include.size) を引いたものである，totalE も外側の原子を削除したもので計算しているので，完全結晶でも外側を考慮していない．分母では ny/cos θ と nz を掛け合わせたものを面積としている．ny/cos θ の長さは図 3.8 を参照する．x 方向は固定境界条件があるため nx の長さを考慮する必要はない．結晶モデルは角度 θ にあわせて ny を拡張する必要があるが nx はすべて同じ長さでよく，nx の値を仮に変更しても同じ粒界エネルギー値が導出されるようになった．

nx,ny,nz=1,1,1 θ =arctan(1/n) のとき n=2 の結晶のことを (x,y,z,n)=(1,1,1,2) の結晶モデルと呼ぶことを前にも説明したが，ここでも同じ表記方法を用いて説明する．計算には (x,y,z,n)=(1,1,1,2),(2,2,2,4),(2,4,2,8),(2,8,2,16) の結晶モデルを用いた．arctan(1/2) の結晶モデルは (2,1,2,2) の大きさになると決まっているので，単に arctan(1/2) の結晶モデルと言えば (2,1,2,2) であり，arctan(1/4) であれば (2,2,2,4) の結晶モデルを意味する．NL の範囲をすべて 0.9 で統一し，内部緩和実行回数=原子数と設定し，結晶モデルの外部緩和→内部緩和の順番で実行した後に上記の式で粒界エネルギーを計算すると以下のようになった．

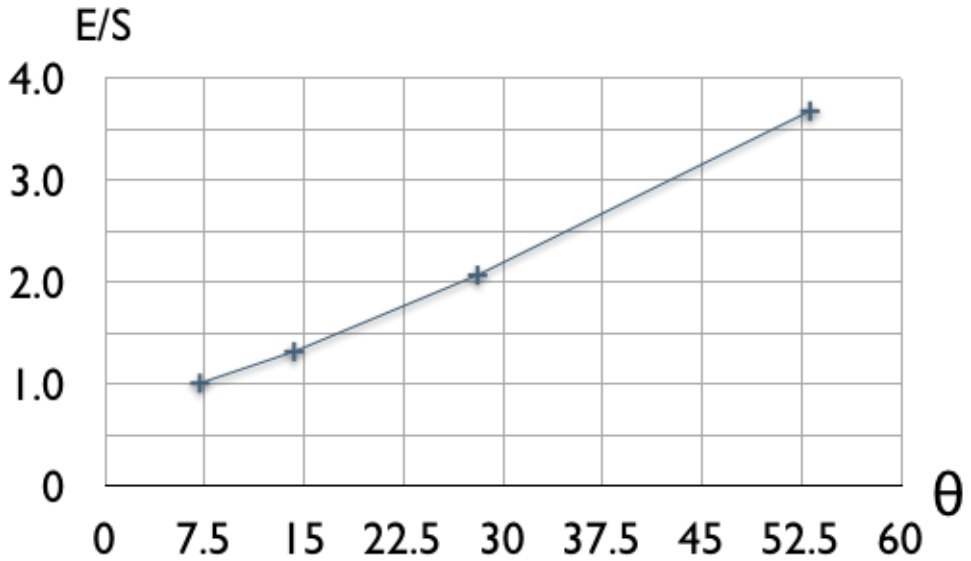

図 3.12: 粒界エネルギーの θ 依存度

θ =arctan(1/2) のときの θ が 0.4636 を示す．右に arctan(1/n) の n の値が 4,8,16 と分母が倍になっていく．本来，小傾角が半分になるとエネルギー値も半分程度まで下がらなければいけないという趣旨の指摘を頂いた．n=8 から n=16 になった時にエネルギー値があまり変化していないため，arctan(1/16) の結晶モデルのエネルギー値が正しく計算できていない可能性がある．

3.5.2 視覚化

内部緩和までの一連の作業が終わった時点での結晶構造を Maya で視覚化すると以下のようになった．それぞれ nx,ny,nz,n=(2,1,2,2),(2,2,2,4),(2,4,2,8),(2,8,2,16) の結晶モデルである

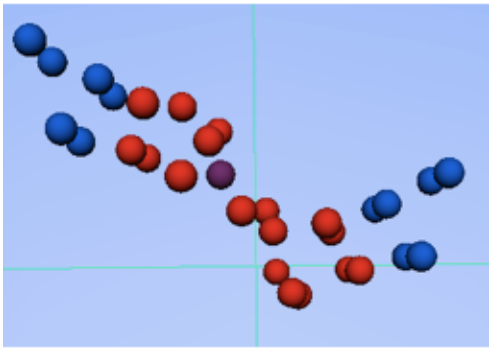

図 3.13: arctan(1/2) の結晶

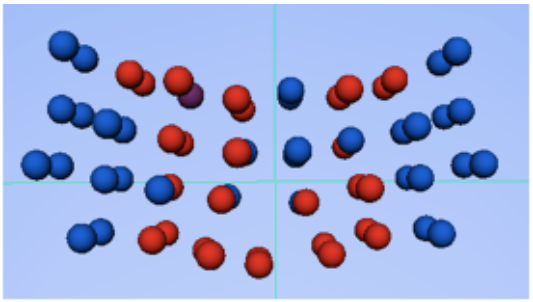

図 3.14: arctan(1/4) の結晶

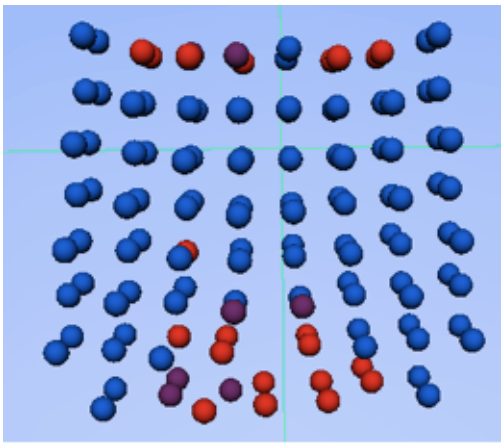

図 3.15: arctan(1/8) の結晶

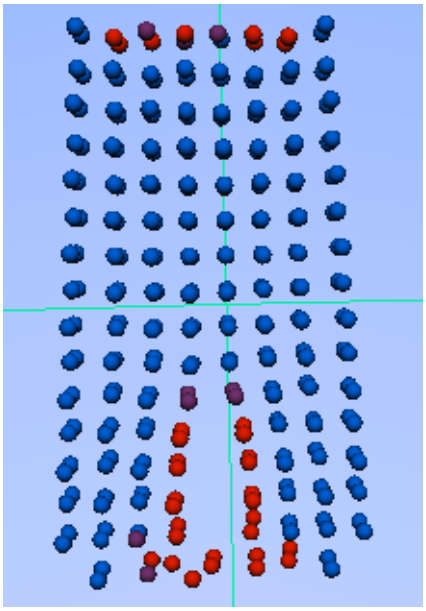

図 3.16: arctan(1/16) の結晶

Maya の視覚化プログラムは B4 釜下が作ったものを使用した．配位数 (NL の数) が 12 の原子は青色，11 は紫色，10 以下の原子は赤色になっている．粒界近辺の原子は配位数が少なくなっていることが分かる．この視覚化のプログラムは原子緩和が正しく行われているか，粒界付近の配位数がどのように変化するか，緩和の順番や回数を変えるとどの程度の変化があるか等を観測するために使用された．

arctan(1/4),arctan(1/8) の原子は緩和により界面が緩やかな斜面になっているが，arctan(1/16) の界面は下部の大きな溝が埋まらず，上部が完全結晶のような形状に緩和されてしまっているため，結晶緩和が正しく行えていない可能性がある．arctan(1/2) の結晶は結晶本体の大きさが小さく，外部緩和により片側が y 軸方向に大きくずれて動いているため，これを見ただけでは周期的境界条件で上下に繋がる様子が把握しにくいと言える．

3.5.3 計算条件の試行

結晶緩和の順番や回数，NL の指定範囲を変えるとどのように結果が変化するのかを試行した．

外部緩和→内部緩和をループさせる

```
3.times do
  for $dy in 0..$nrange do
    for $dz in 0..$nrange do
      brent($xrange0,15.0,$xrange1,@x,1.0e-2,0.0)
    end
  end
  relaxed()
end
```

内部緩和後に外部緩和をもう一度する必要があるのではないかと考えて試してみたが，Maya で視覚化したモデルの形状もエネルギー値も全く同じ結果が得られた．

内部緩和の数を 3 倍に増やす

```
$inner_rel=$atom_include.size //内部緩和実行回数
↓
$inner_rel=$atom_include.size*3 //内部緩和実行回数
```

エネルギー値が 0.1 程度小さい値になった．内部緩和によるエネルギー値は結果に少しランダム要素が関わるため，回数を増やすことで結果の信頼性が上がるが，根本的にはほぼ同じ結果が得られたと言える．

内部緩和で使用する NL の範囲を 0.9 から 1.4 まで拡張

関数 InnerRelax の内部の関数宣言 makeNL(0.9) を makeNL(1.4) に書き換えた．arctan(1/16) のモデルの結晶の粒界が少し狭くなりエネルギー値は全体的に少し下がった，

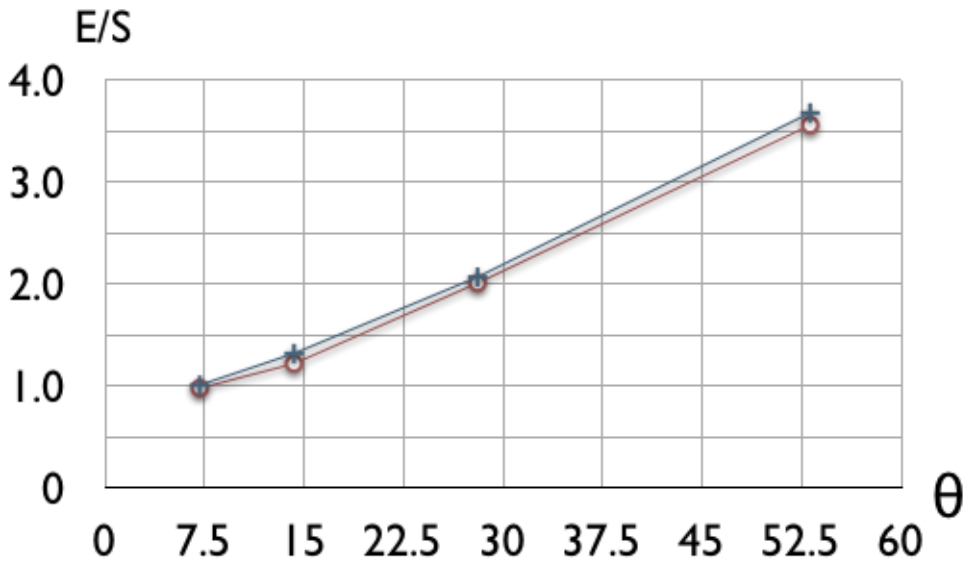

図 3.17: 内部緩和 NL 範囲拡張後=赤線

NL0.9		NL1.4	
7.156	1.01	7.156	.98
14.257	1.32	14.257	1.22
28.086	2.07	28.086	2.01
53.153	3.68	53.153	3.56

図 3.18: 数値比較

エネルギー値の減少割合が逓減する根本的な問題の解決には至らなかった.

第4章　総括

本研究によって得られた結果を以下に示す.

- スクリプト言語 ruby を使用し，任意の大きさと小傾角を持つ粒界結晶を作成した．この方法を使う利点はいくつかある．まず，同形式のデータに統一することで読み込ませるデータの数値を変更するだけで表示結果の変化を確認することができる．また，作成した結晶モデルの座標データを西谷研 B4 釜下の作った Maya の視覚化プログラムに読み込ませることができるため，作成した結晶モデルの形が想定していたものと逸脱していないかを確認することが出来る.

- 作成した結晶モデルの座標データを読み込み，Rennard-Jones ポテンシャルを用いてエネルギー値の計算を行うプログラムを作成した．西谷研 B4 松塚の作った brent 法と MC 法の結晶緩和プログラムを導入したが，エネルギー値の計算部分とは別個にされているため，プログラムの一部を外しての動作をチェックすることができ，作業手順の変更や外部から持ってきたプログラムの実装が容易になった.

- このように作業を分割して行うことにより，プログラムの欠陥を見つけることが容易であるほか，データの整理や修正がしやすくなったが，プログラムの細かな部分の欠陥を見つかるのはそれほど簡単なことではなく，例として周期的境界条件が正しく動作しているか，等を確認するには至らなかった．しかし，結晶緩和を行った結晶モデルを Maya で視覚化することで直接原因を考察することが出来た.

謝辞

本研究の遂行にあたり，終始多大な有益な御指導，及び丁寧な助言を頂いた西谷滋人教授に深い感謝の意を表します．

また，本研究を進めるにつれ，西谷研究室に所属する同輩達，並びに先輩方からの様々な知識の供給，ご協力を頂き，本研究を成就させる事ができました．この場を借りて心から深く御礼申し上げます．

参考文献

[1] 鈴木秀次 著 『転位論入門』(アグネ出版 1978)

[2] 鈴木秀次 著 『転位論入門』 (アグネ出版 1978)

[3] 西谷滋人 著 『固体物理の基礎』 (森北出版 2006)

[4] C.Kittel 著 宇野良清 津屋昇 新関駒次郎 森田章 山下次郎 共役 『キッテル固体物理学入門 第 8 版』(丸善 2005)

付録

2章で作成した結晶モデル作成プログラムを以下に示す.

```
require 'pp'
include Math
require 'scanf'
size = {
  :nx => ARGV[0].to_i,
  :ny => ARGV[1].to_i,
  :nz => ARGV[2].to_i,
  :theta => atan(1.0/ARGV[3].to_f),
}

file = open('POSCAR.txt')
pos=[]
while line = file.gets do
  if /Direct/ =~ line
    while line = file.gets do
      pos0=line.chomp.split(" ")
      pos1=[]
      pos0.each do |p|
        pos1 << p.to_f
      end
      pos << pos1
    end
  end
end
file.close

def extend(pos,size)
  atom_pos=[]
  for i in 0..size[:nx] -1 do
    for j in 0..size[:ny] -1 do
```

```
      for k in 0..size[:nz] -1 do
        pos.each do |p0|
          p1=[]
          p1[0] = p0[0]+1.0*i
          p1[1] = p0[1]+1.0*j
          p1[2] = p0[2]+1.0*k
          atom_pos << p1
        end
      end
    end
  end
  return atom_pos
end

def rotation(pos,size)
  pos2=[]
  pos.each do |pos1|
    tmp = []
    tmp << pos1[0]*cos(size[:theta]) + pos1[1]*(-sin(size[:theta]))
    tmp << pos1[0]*(sin(size[:theta])) + pos1[1]*cos(size[:theta])
    tmp << pos1[2]
    pos2 << tmp
  end
  return pos2
end

def translation(pos,size)
  dx = size[:nx].to_i * cos(size[:theta])
  dy = size[:nx].to_i * sin(size[:theta])
  10 *size[:ny].to_i.times do
    for i in 0..pos.size-1 do
      if pos[i][0] < -0.00001 then
        #    if pos[i][0] < 0 then
        pos[i][0] = pos[i][0]+dx
        pos[i][1] = pos[i][1]+dy
      end
    end
  end
  return pos
```

```
end

def mkMirror(pos)
  dummy = Marshal.load(Marshal.dump(pos))
  for i in 0..dummy.size-1 do
    dummy[i][0] = dummy[i][0]-2*dummy[i][0]
  end
  dummy.each do |p|
    pos << p
  end
  pos=Remove_duplicate(pos)
  return pos
end

def Remove_duplicate(pos)
  for i in 0..pos.size-1 do
    if pos[i][0] < 0.0001 && pos[i][0] > -0.0001
      pos[i][0] = 0.0
    end
  end
  pos = pos.uniq
  return pos
end


print "nx=",size[:nx],"\n","ny=",size[:ny],"\n","nz=",size[:nz],"\n","cf=",ARGV[3]

pos=extend(pos,size)
pos=rotation(pos,size)
pos=translation(pos,size)
pos=mkMirror(pos)

for i in 0..pos.size-1 do
  printf("%f %f %f\n",pos[i][0],pos[i][1],pos[i][2])
end
pp pos.size

file = File.open(ARGV[4],'w')
file.printf("%s %f \n","nx=",size[:nx])
```

```
file.printf("%s %f \n","ny=",size[:ny])
file.printf("%s %f \n","nz=",size[:nz])
file.printf("%s %f \n","cf=",ARGV[3].to_f)
file.printf("%s \n","Direct")
for i in 0..pos.size-1 do
  file.printf("%f %f %f\n",pos[i][0],pos[i][1],pos[i][2])
end
file.close
```

3 章で作成したエネルギー値計算プログラムに結晶緩和を実装したものを以下に示す.

```
include Math
require 'pp'
require 'matrix'


class Atom
  attr_accessor :pos, :pos0, :nl
  def initialize(pos)
    @nl=[]
    @pos=Array.new(pos)
    @pos0=Array.new(pos)
  end
  def energy()
    ene=0
    nl.each do |j|
      ene+=ij_energy(distance(@pos,$atom_list[j].pos))
    end
    return ene
  end
  def reset_nl()
    @nl=[]
  end

  A0=1.587401051
  E0=-1*4.0/12.0
  def ij_energy(r)#LJP
    a=1.0/(A0*r)
    return E0*((a**6 )-(a**12))
```

```
  end
end

def makeLattice()
  file = open(ARGV[0])
  atom_list=[]; n=[]; i=0
  while line = file.gets do
    if /Direct/ =~ line
      while line = file.gets do
        pos0=line.chomp.split(" ")
        pos1=[]
        pos0.each do |p|
          pos1 << p.to_f
        end
        atom_list << Atom.new(pos1)
      end
    end
  end
  file.close
  make_L()
  return atom_list
end

def make_L()
  $n=inputsize()
  $nx = $n[0]
  $ny = $n[1]
  $nz = $n[2]
  $cf = $n[3]

  $theta=atan(1.0/$cf)
  lx = 100
  ly = $ny/cos($theta)
  lz = $nz
  pp $theta
  $L=[lx,ly,lz]
end
```

```
def inputsize()
  file = open(ARGV[0])
  $n=[]
  while line = file.gets do
    if /Direct/ =~ line
    else
      n0 = line.chomp#.split(" ")
      $n << n0.to_i
    end
  end
  file.close
  return $n
end


def distance(a,b)
  tmp=0
  for i in 0..2 do
    x=a[i]-b[i]
    x=x-(x/$L[i]).round*$L[i]
    tmp+=x*x
  end
  return  sqrt(tmp)
end

def mkAtom_include
  atom_include=[]
  max_x = -100
  min_x = 100
  $atom_list.each do |i_a|
    if i_a.pos[0] > max_x then
      max_x = i_a.pos[0]
    elsif i_a.pos[0] < min_x then
      min_x = i_a.pos[0]
    end
  end

  $atom_list.each_with_index do |i_a,idx|
    if i_a.pos[0] < max_x-0.51*cos($theta) && i_a.pos[0] > min_x+0.51*cos($theta)
```

```
      atom_include << idx
    end
  end
  return atom_include
end

def makeNL(rc)
  $atom_list.each do |ai| ai.reset_nl end
  nmax=$atom_list.length-1
  for i in 0..nmax do
    ai=$atom_list[i]
    for j in i+1..nmax do
      aj=$atom_list[j]
     if distance(ai.pos,aj.pos)<rc then
        ai.nl << j
        aj.nl << i
      end
    end
  end
end

def maxpos(pos)
  max_x = -100
  pos.each do |i_a|
    if i_a.pos[0] > max_x then
      max_x = i_a.pos[0]
    end
  end
  return max_x
  print "max_x=",max_x,"\n"
end

def minpos(pos)
  min_x = 100
  pos.each do |i_a|
    if i_a.pos[0] < min_x then
      min_x = i_a.pos[0]
    end
  end
```

```
  return min_x
  print "min_x=",min_x,"\n"
end

def block_move(dx,dy,dz)
  $atom_list.each do |atom|
    if atom.pos0[0]>=0.0 then
      atom.pos[0]=atom.pos0[0]+dx
      atom.pos[1]=atom.pos0[1]+dy
      atom.pos[2]=atom.pos0[2]+dz
    end
  end
end

def total_E()
  total_E=0.0
  $atom_include.each do |i|
    total_E+=$atom_list[i].energy()
  end
  return total_E
end

def diff_E(atom,j,dx)
  e0 = atom.energy()
  atom.nl.each do |i|
    if $atom_include.member?(i) then
      e0+= $atom_list[i].energy()
    end
  end
  atom.pos[j]+=dx
  e1 = atom.energy()
  atom.nl.each do |i|
    if $atom_include.member?(i) then
      e1+= $atom_list[i].energy()
    end
  end
  diff_E=e0-e1
  atom.pos[j]-=dx
  return diff_E
```

```
end

T=0.001
$trial=0
$accept=0
def InnerRelax(trial)
  trial.times do
    makeNL(1.9)
    $atom_include.each do |i|
      atom=$atom_list[i]
      for j in 0..2 do
        if $MC then
          $trial+=1;$accept+=1
          dx=(1.0-2.0*rand())/100.0
          atom.pos[j]+=dx
          de=diff_E(atom,j,-dx)
          if exp(-de/T)<rand() then #mc
            $accept-=1
            atom.pos[j]-=dx
          end
        else
          $trial+=1;$accept+=1
          dx=(1.0-2.0*rand())/100.0
          atom.pos[j]+=dx
          de=diff_E(atom,j,-dx)
          if de>0 then
            $accept-=1
            atom.pos[j]-=dx
          end
        end
      end
    end
  end
end

$nrange=10
$dy=0.0
$dz=0.0
def f(x)
```

```
  dx=x/100.0
  dy=$dy/$nrange.to_f
  dz=$dz/$nrange.to_f
#  makeNL()
  block_move(dx,dy,dz)
  makeNL(0.9)
  e = total_E()
  return e
end

def mysign(a,b)
  if  b>=0 then c=(a).abs
  else c=-1*(a).abs end
end

def hatena(a,b,c,d)
  if a>=b then return c
  else return d end
end

$min=[0.0,0.0,0.0,10000000]
$max=[100.0,100.0,100.0,-10000000]
$e1={}
def brent(ax,bx,cx,x,tol,xmin)
  cgold=1.0-2.0/(1.0+sqrt(5.0))
  zeps=1.0e-8
  e=0
  d=0

  if ax<cx then a=ax else a=cx end
  if ax>cx then b=ax else b=cx end

  x=bx;w=bx;v=bx
  fw=f(x);fv=f(x);fx=f(x)
  for j in 1..50 do
    xm=0.5*(a+b)
    tol1=tol*(x).abs+zeps
    tol2=2.0*tol1
    if (x-xm).abs<=tol2-0.5*(b-a) then
```

```
      xmin=x
      print(x,"  \t",fx,"\n");
      $e1[[$dy,$dz]] = fx
#         print("[",$dy,",",$dz,"] ",$e1[[$dy,$dz]],"\n")
      if $min[3]>fx then
        $min=[x/100.0,$dy/$nrange.to_f,$dz/$nrange.to_f,fx]
      elsif $max[3]<fx then
        $max=[x/100.0,$dy/$nrange.to_f,$dz/$nrange.to_f,fx]
      end
      break
    end

    if (e).abs>tol1 then
      r=(x-w)*(fx-fv)
      q=(x-v)*(fx-fw)
      p=(x-v)*q-(x-w)*r
      q=2.0*(q-r)

      if q>0.0 then p=-p end
      q=(q).abs
      etemp=e
      e=d
      if (p).abs >= (0.5*q*etemp).abs || p<= q*(a-x) || p >= q*(b-x) then
        e=hatena(x,xm,a-x,b-x)
        d=cgold*e
      else
        d=p/q
        u=x+d
        if (u-a) < tol2 || (b-u)<tol2 then d=mysign(tol1,xm-x) end
      end

    else
      e=hatena(x,xm,a-x,b-x)
      d=cgold*e
    end

    u=hatena((d).abs,tol1,x+d,x+mysign(tol1,d))
    fu=f(u)
    if fu<=fx then
```

```
      if u>=x then a=x
      else b=x end
      v=w;w=x;x=u;fv=fw;fw=fx;fx=fu
    else
      if u<x then a=u else b=u end
      if fu<fw || w==x then v=w;w=u;fv=fw;fw=fu
      elsif fu <= fv || v==x || v==w then v=u;fv=fu end
    end
  end
  xmin=x
end

def relaxed()
  block_move($min[0],$min[1],$min[2])
  print "relaxed()内:",$atom_list[0].pos[0]," ",$atom_list[0].pos[1]," ",$atom_lis
  InnerRelax($inner_rel)
  makeNL(0.9)
  $min[3] = total_E()
pp $min[3]
#  $relaxed_pos = Marshal.load(Marshal.dump($atom_list))
end

$xrange0=0
$xrange1=30
$atom_list=makeLattice()
$atom_include=mkAtom_include()
$t0=Time.now

for $dy in 0..$nrange do
  for $dz in 0..$nrange do
    brent($xrange0,15.0,$xrange1,@x,1.0e-2,0.0)
  end
end


$MC=true#MCを使用するときtrue 降下法のときfalse
$inner_rel=$atom_include.size
relaxed()
```

```
print "計算にかかった時間 ",Time.now-$t0,"秒\n"
atom_idx=[]

for i in 0..$atom_list.size-1 do
  atom_idx << i
end

if $MC
  print("mc の計算結果\n")
else
  print("降下法の計算結果\n")
end

pp -$atom_list.size
pp -$atom_include.size
pp $min[3]
boundary_E = ($min[3]-(-$atom_include.size))/($ny*(1/cos($theta))*$nz)
print("1POSCAR 辺りの粒界エネルギー:",boundary_E,"\n")

file = File.open(ARGV[1],'w')#データ群出力
file.print("nx,ny,nz,cf=",$nx," ",$ny," ",$nz," ",$cf,"\n")
file.print("theta:",$theta,"\n")
file.print("1/cos(theta):",1/cos($theta),"\n")
file.print("atom_include.size:",$atom_include.size,"\n")
file.print("min_E:",$min[3],"\n")
file.print("D_E:",$min[3]-(-$atom_include.size),"\n")
file.print("boundary_e:",boundary_E,"\n")
file.print("atom_include.size:",$atom_include.size)
file.close


file = File.open(ARGV[2],'w')#座標出力
atom_idx.each do |i|
  min_x1=minpos($atom_list)
  max_x1=maxpos($atom_list)
  if $atom_list[i].pos[0] < max_x1-0.51*cos($theta) && $atom_list[i].pos[0] > min_
    file.printf("%f %f %f %d %f \n",$atom_list[i].pos[0],$atom_list[i].pos[1],$ato
  else
    file.printf("%f %f %f %d %f \n",$atom_list[i].pos[0],$atom_list[i].pos[1],$ato
```

```
 end
end
file.close
```