卒業論文

pseudoVASP の開発

関西学院大学理工学部
情報科学科 西谷研究室 9547 中井　遥

2013 年 3 月

指導教員　西谷滋人　教授

概 要

西谷研究室では，構造緩和やモンテカルロシュミレーションなどの原子系を操作するコントローラを作成している．そのコントローラは第一原理計算ソフト VASP を何度も実行することで計算を行っているが，VASP の計算はその高い精度ゆえに膨大な時間を必要とする．そこで本研究では，コントローラ作成に要する時間を短縮するために，VASP よりも短い時間で VASP と同様の出力を行う pseudoVASP を作成した．計算時間短縮のため，第一原理計算の代わりに，原子間ポテンシャルのひとつである Lennard-Jones ポテンシャルを用いてエネルギーと force を計算した．

また作成したプログラムの検証のため，2011 年度卒業生真嶋の作成した内部緩和のプログラムを使用し，計算を行った．その結果，VASP よりも大幅に短い計算時間で，VASP よる計算結果とほぼ同様の結果を得ることができた．

目 次

第1章　序論

第一原理計算ソフト VASP は材料の物性を精度よく求めることが可能であるが，一回の計算に膨大な時間を要する．西谷研究室では，内部緩和，モンテカルロシミュレーションなどの原子系を操作するコントローラを作っているが，VASP を何度も呼び出すため，本物を使っていては開発に多大な時間がかかる．そこで本研究では，より短い計算時間であたかも VASP を実行しているかのように動く pseudo(似非)VASP を作ることを目的とする．また，昨年度真嶋によって実装された内部緩和プログラムを使って，その動作の検証と性能を計測する．

1.1　コントローラ

第一原理計算を行うソフトウェアである VASP は，平面波基底，擬ポテンシャル法を用いることで，高精度な計算を進めるプログラムである [1]．西谷研究室では，この VASP の計算を駆動するコントローラを作成している．構造緩和や分子動力学，モンテカルロシュミレーションなどこれらのコントローラでは，VASP を使った計算で求めたエネルギーやフォースの値から様々な処理を行い，新たな原子座標を作成する．そして作成した原子座標を VASP に入力として与え，再び計算を行う．以上を，各コントローラごとに定められた計算終了条件を満たすまで何度も繰り返すことで，解を求めている．

1.2　VASP と pseudoVASP

1.2.1　入出力ファイル

コントローラは VASP を使って計算を行うことを想定して作成されるので，本研究で作成する pseudoVASP の入力と出力も VASP で使用するものと同じ形式をとる必要がある．VASP の入力には，POSCAR という計算モデルに関するファイルを使用する．POSCAR では，モデル構築において，格子ベクトルなどユニットセルの形状に関する情報や原子の位置を決定している．またその出力として OUTCAR ファイルを作成している．OUTCAR は計算終了後に作成されるファイルであり，その計算結果が出力される．計算モデルの系全体のエネルギーやフォース，原子

座標，計算時間なども記載されている．図 1.1 に入出力ファイルとコントローラ，VASP,pseudoVASP との関係を示した．

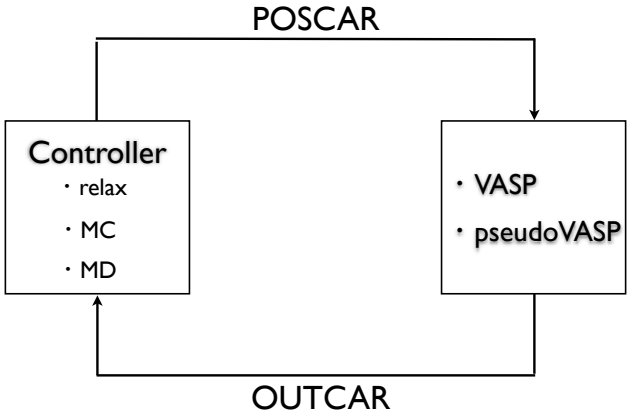

図 1.1: 関係性の模式図

1.2.2 pseudoVASP の必要性

ではなぜ pseudoVASP が必要になるのであろうか．コントローラを用いて計算を行う場合，その計算時間のほとんどは第一原理計算に費やされる．第一原理計算は，その高い精度ゆえに多大な時間を必要とする．例えば Al 原子 32 個で構成された原子モデルを考えてみる．VASP を用いてそのモデルのエネルギーを求めた時にかかる時間は 803 秒であった．コントローラを使って VASP で計算を行う場合，最適解を求めるまで何度もこの計算を行うことになる．仮に解を求めるまで 20 回の計算を行ったとすれば，VASP による計算時間だけで 16060 秒，つまり 4 時間以上もの時間がかかることになる．作成したコントローラの動作を確認するという目的に対して，これはあまりにも膨大な時間を必要としていると考えられる．

本研究で作成する pseudoVASP では計算時間を短縮するために，経験的な原子間ポテンシャルを利用する．計算時間を短縮する方法のひとつが，一回のエネルギー計算にかかる時間を短くするということである．表 1.1 に，第一原理計算と原子間ポテンシャルの性能のトレードオフを示した．第一原理計算は信頼性は高いものの計算には時間がかかり，一方原子間ポテンシャルには信頼性は乏しいものの計算速度が高速であるいう特徴がある．そこで pseudoVASP では，原子間ポテ

ンシャルのひとつである Lennard-Jones ポテンシャルを用いて系のエネルギーを求めていく.

表 1.1: 第一原理計算と原子間ポテンシャルの性能比較.

	信頼性	速度
第一原理計算	○	×
原子間ポテンシャル	×	○

本研究の目的は，VASP を用いて第一原理計算を行うよりも短い計算時間で，原子のもつエネルギーやフォースを求める pseudoVASP を開発することである．pseudoVASP では， Lennard-Jones ポテンシャルを用いて系のエネルギーを求め，求めたエネルギーを微分することにより各原子のもつフォースを計算していく．コントローラの開発過程において VASP の代わりに pseudoVASP を使用することで，開発にかかる時間を大幅に短縮することができると考えられる．なお，pseudoVASP の入出力ファイルの形式は VASP で使用するものと同じものを使用し， VASP と pseudoVASP を容易に切り替えることが可能な状態にしておく．作成した pseudoVASP が正しく動作を行うか確認するため，今回は 2011 年度卒業生真嶋の作成した内部緩和のプログラムを用いて計算を行う [2]． 真嶋は SiO を用いて計算を行ったが，今回は動作の確認が目的であるため，より単純な構造をもつ Al 原子のモデルを用いて計算を行っていく．以降，その内部緩和プログラムを inner_relax とよぶ.

第2章　手法

2.1　計算の流れ

今回作成した pseudoVASP は，以下の流れで計算を行っている．

(i) POSCAR から原子座標を読み取る．

(ii) 読み取った原子座標から近接原子を選択し，配列ネイバーリストに格納．

(iii) Lennard-Jones ポテンシャルでエネルギーを計算

(iv) 求めたエネルギーを距離で微分して force を計算

(v) 結果を OUTCAR 形式で出力

以上の流れを図 2.1 に示した．各項目におけるプログラムの詳細は付録にて解説を行っているが，ここではエネルギー計算，force の計算の部分について解説を行う．

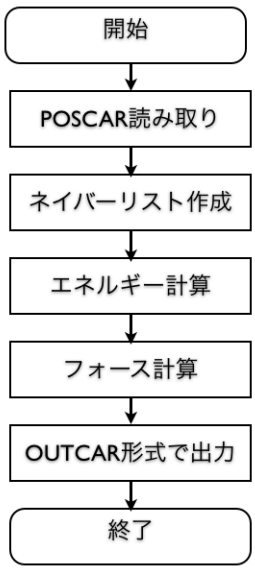

図 2.1: 計算のフローチャート．

なお，今回の計算には Al のユニットセル [3] を三軸方向にそれぞれ 2 倍ずつ拡張したスーパーセルを用いた．図 2.2 はそのモデルを表している．本研究ではこのモデルの原子を一個移動させ，真嶋のプログラムによる内部緩和を行っていく．

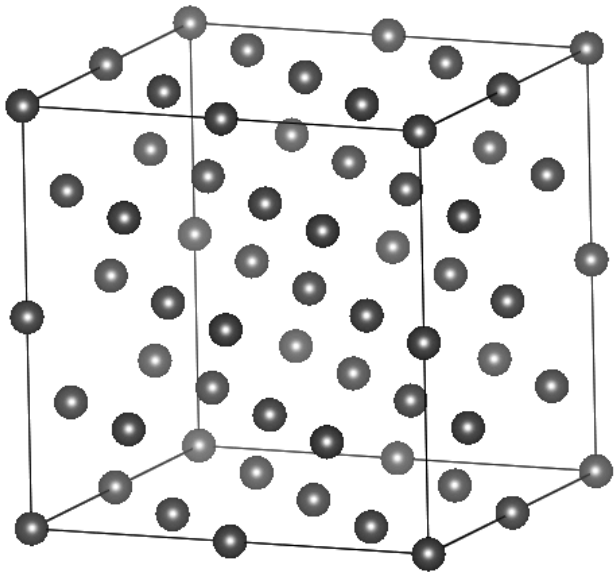

図 2.2: $2 \times 2 \times 2$ に拡張したスーパーセル．

2.2 Lennard-Jones ポテンシャルによるエネルギー計算

ばねモデルは，固体の原子レベルの力学的振る舞いを記述する最も適切なモデルである．今回用いる Lennard-Jones ポテンシャルは，ばねモデルを数学的に記述する際によく使用されているポテンシャルであり，二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 [4]．式 (2.1) にその式を示しておく．

$$\left.\begin{aligned} E_i &= \textstyle\sum_i \psi(R_{i,j}) \\ \psi(R_{i,j}) &= A(\tfrac{1}{R_{i,j}})^{12} + B(\tfrac{1}{R_{i,j}})^{6} \end{aligned}\right\} \tag{2.1}$$

ここで ψ は相互作用エネルギーを表しており，$R_{i,j}$ は原子 i と原子 j の原子間距離を示している．また，A, B はそれぞれ定数を表している．相互作用エネルギー $\psi(R_{i,j})$ の距離依存性は図 2.3 のようになる．$\psi(R_{i,j})$ は二原子間の距離 $R_{i,j}$ におけるエネルギー，グラフの ε は凝集エネルギー (このときの $R_{i,j}$ が平衡原子間距離) を表している．結晶の持つエネルギーはすべての原子対についての Lennard-Jones ポテンシャルの和によって与えられる．

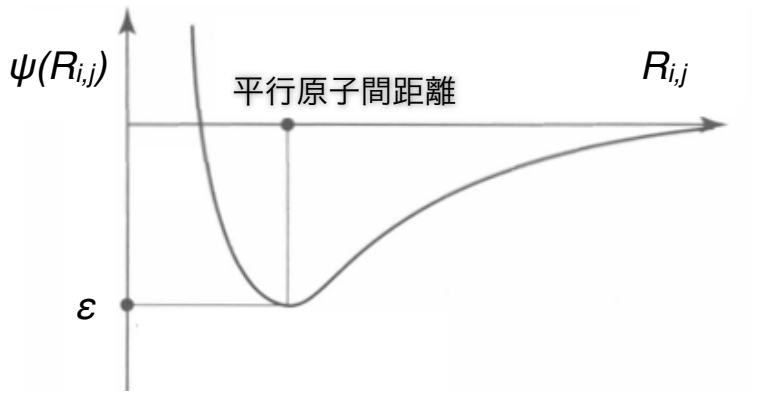

図 2.3: 相互作用エネルギーの距離依存性.

本プログラムでは，以下の関数でその計算を行った.

```
def lj_energy()
  a0=1.587401051
  e0=-1*4.0/12.0
  ene=0.0
  nl.each do |j|
    r=distance(@pos,$atom_list[j].pos)
    a=1.0/(a0*r)
    ene+=e0*((a**6)-(a**12))
  end
  return ene
end
```

ここで nl は近接原子を格納した配列である．nl にあらかじめ格納しておいた原子座標を取り出し，関数 distance により 2 原子間の距離を求め Lennard-Jones ポテンシャルを適用している.

2.3 force の計算

原子系において原子が安定な位置にあるかどうかを調べるためには，各原子の持つ force を計算する必要がある．原子のもつ force が 0 に近ければ近いほど，その原子は安定な位置にあるといえる．force はエネルギーを距離で微分して求めることができるので，ここでは Lennard-Jones ポテンシャルを用いて求めたエネルギーを原子間の距離で微分する．以下のプログラムでその計算を行う．

```
  def lj_force()
    a0=1.587401051
    e0=-1*4.0/12.0
    f=[0.0,0.0,0.0]
    nl.each do |j|
      t=f_distance(@pos,$atom_list[j].pos)
      x=t[0]
      y=t[1]
      z=t[2]
      r=x**2+y**2+z**2
      dedr=-e0*(12/(a0**12*(r)**7)-6/(a0**6*(r)**4))
      f[0] += x*dedr
      f[1] += y*dedr
      f[2] += z*dedr
    end
    printf("%10.6f %10.6f %10.6f %10.6f %10.6f %10.6f\n",pos0[0],pos0[1],pos0[2],f[0],f[1],f[2])
    return f
  end
end
```

t には，関数 f_distance を用いて二原子間の距離を x, y, z 成分ごとに求めて格納している．ここで，エネルギーの距離微分を行っているのは， $dedr = -e_0 \times (12/(a_0^{12} \times (r)^7) - 6/(a_0^6 \times (r)^4))$ の部分であり，その導出を以下に示す．

はじめに，式 (2.1) に示した Lennard-Jones ポテンシャルを用いて二原子間のエネルギーを求める．式 (2.2) は Lennard-Jones ポテンシャルを本プログラムにおける変数で表したものである．

$$\phi = e_0 \left\{ \left(\frac{1}{a_0 * r} \right)^{12} - \left(\frac{1}{a_0 * r} \right)^{6} \right\} \tag{2.2}$$

ここで r は二原子間の距離を示す変数であるので，

$$r = \sqrt{x^2 + y^2 + z^2} \tag{2.3}$$

と表すことができる．これを式 (2.2) に代入すると

$$\phi = e_0 \left\{ \frac{1}{a_0^{12}(x^2 + y^2 + z^2)^6} - \frac{1}{a_0^6(x^2 + y^2 + z^2)^3} \right\} \tag{2.4}$$

となる．これが各原子間におけるポテンシャルを表している．次に，このポテン

シャルから force を求めるため，x, y, z の各成分ごとの距離で微分していくと，

$$\left\{\begin{array}{l} \frac{d\phi}{dx} = -e_0 \left\{ \frac{12x}{a_0^{12}(x^2+y^2+z^2)^7} - \frac{6x}{a_0^6(x^2+y^2+z^2)^4} \right\} \\ \frac{d\phi}{dy} = -e_0 \left\{ \frac{12y}{a_0^{12}(x^2+y^2+z^2)^7} - \frac{6y}{a_0^6(x^2+y^2+z^2)^4} \right\} \\ \frac{d\phi}{dz} = -e_0 \left\{ \frac{12z}{a_0^{12}(x^2+y^2+z^2)^7} - \frac{6z}{a_0^6(x^2+y^2+z^2)^4} \right\} \end{array}\right. \tag{2.5}$$

となる．これが，各成分ごとの二原子間に生じる force を表している．この計算を近接原子すべてで行い，成分ごとに和を求めることで各原子にかかる force を算出している．この関数では簡略化のため，式 (2.5) において $dedr = -e0\left(\frac{12}{a0^{12}(x^2+y^2+z^2)^7} - \frac{6}{a0^6(x^2+y^2+z^2)^4}\right)$ と置き換え，

$$\left\{\begin{array}{l} \frac{d\phi}{dx} = dedr \times x \\ \frac{d\phi}{dy} = dedr \times y \\ \frac{d\phi}{dz} = dedr \times z \end{array}\right. \tag{2.6}$$

として計算を行っている．以上の式を用いて各原子の受ける force を求め，配列 f に x, y, z 成分ごとに格納している．また，この関数の中で原子座標と force を OUTCAR に書き込んでいる．

2.4 内部緩和

本研究では，作成した pseudoVASP の動作，性能を検証するために真嶋の作成したプログラムによる内部緩和を行う．内部緩和の流れを以下に示す．

(i) OUTCAR から原子座標とエネルギー，force を取り出す．

(ii) CG 法で共役な方向を決める．

(iii) mnbrak で極小を囲い込む

(iv) Brent 法で 1 次元における極小を求める

(v) 以上を，最小にたどり着くまで繰り返す．

図 2.4(a) は上記の内部緩和の流れを示している．このプログラムでは，エネルギー計算を行う際に原子座標や force を取得し，新たな原子座標を作成している．作成した新たな原子座標を POSCAR に書き込み，それを用いて pseudoVASP での計算を行っている．図 2.4(b) に pseudoVASP との連携の流れを示した．なお，

ここで示される Δx は Brent 法によって新たに求めた原子座標と，前回の計算時に用いた原子座標との各成分ごとの差を表している．また ΔE は上記の手順 (i) で取り出したエネルギーと，手順 (ii) から (iv) の計算を行い求めたエネルギーとの差を表している．Δx ，ΔE は今回は計算時間を短くするため図 2.4 のように条件を設定しているが，計算の用途によってはその値を変更することが必要となる．今回用いる inner_relax において,Δx は関数 brent 中の delta_x ，ΔE は関数 frprmn 中の delta_e という変数で定めている．各プログラムの詳細は付録にて説明する．

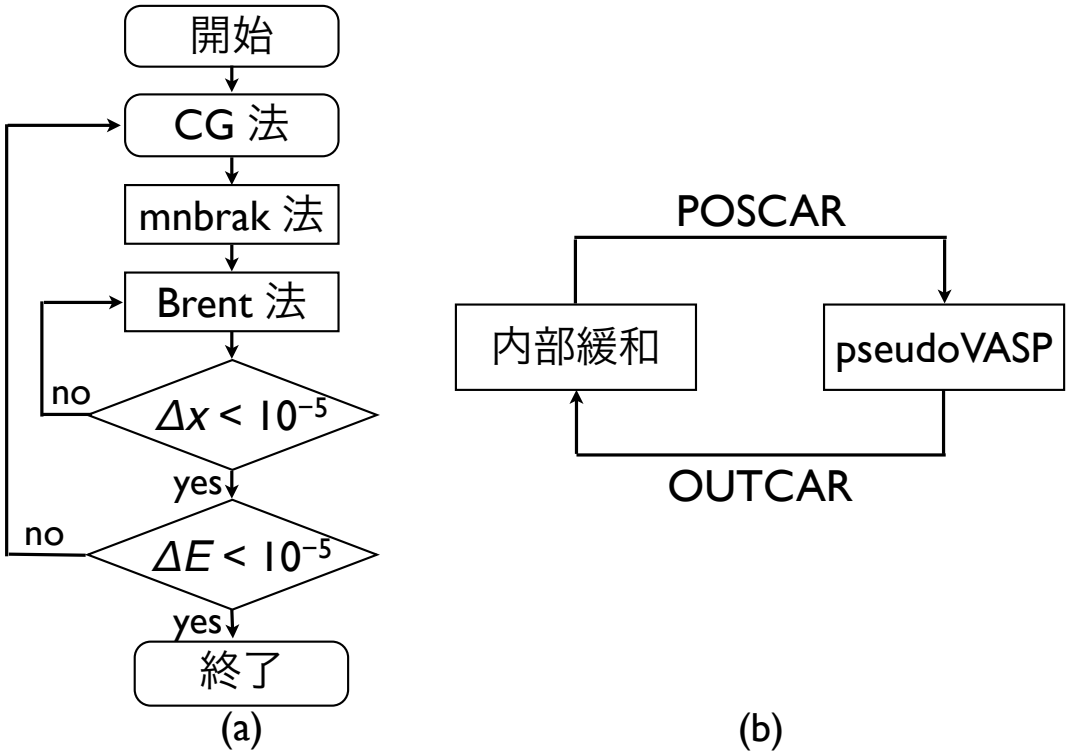

図 2.4: (a) 内部緩和の流れ，(b)pseudoVASP との連携の流れ．

2.5　最小値の計算

ここで，実際に内部緩和を行っている計算手法について説明する [2].

2.5.1　mnbrak

1 次元での最小値を求める手法である Brent 法 (詳細は 2.5.2 節) では最小値を挟み込むような 3 点を初期値として与えなければならないため，そのような点を選ぶ関数 mnbrak を用いる．ある極小のまわりに点を選んだ状態のことを極小を「囲い込む」という．図 2.5(a) は極小を囲い込んでいない 3 点であり，図 2.5(b) は囲い込みができている 3 点を示したものである．

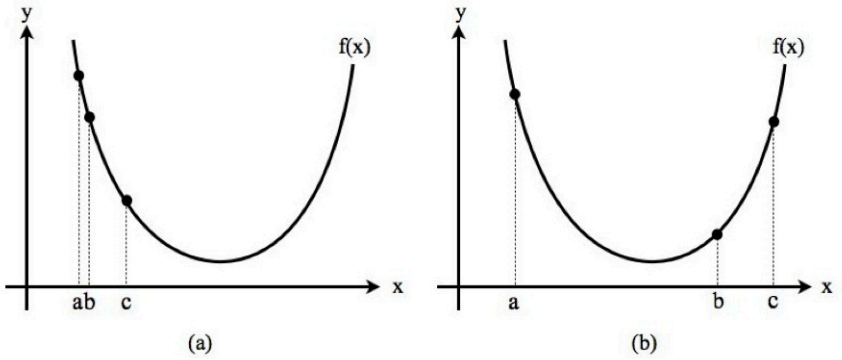

図 2.5: (a) 囲い込みできていない 3 点 (b) 囲い込みできている 3 点.

ある区間 [a,b] に根 (関数値が 0 の点) があるかどうかを考える場合，a,b の関数値が異符号ならば，確実に根が存在すると言える．しかしその区間に極小値があるかどうかは 2 点だけでは判定できない．そこで　a＜b＜c となる 3 点を与えたとする．もし図 2.5(b) のように $f(b)$ が $f(a)$ と $f(c)$ の両方よりも小さければ，区間 (a,c) 内に必ず極小を持つということがわかる [5]. 関数 mnbrak ではこのような 3 点を求める．この内部緩和プログラムでは，緩和する原子の座標 (pos1) と，その原子の最近接原子間距離の 1/10 倍の点の座標 (pos2) の 2 点を与え，これらの 2 点の座標を入力としてエネルギーの計算をおこなう．その求めたエネルギーと原子座標を初期値として囲い込む 3 点を求める．3 点目の座標は pos2+gold*(pos2-pos1) として計算する．図 2.6(a),(b) が関数 mnbrak で囲い込むまでの 3 点を示したものである．図 2.6(a) では極小を囲い込めているように見えるが，真ん中の青い点が右側の緑の点よりも大きいため，極小を囲い込んでいると断定できない．そのためこの段階では終了せず，図 2.6(b) のような 3 点になるまで動く．ここで初期点 (pos1) を極小の右側にとったものを図 2.7 に示した．ここで初期値として pos1 に x=6, pos2 に x=7 をとっている．この場合は pos1 よりも pos2 が下にあるた

め，pos1 と pos2 を入れ替え，左側に進んでいく．ただし gold=(1+$sqrt(5)$)/2 とし，黄金比を表している．黄金比を用いるのは，最小値を求める際に黄金分割法を用いるからである．

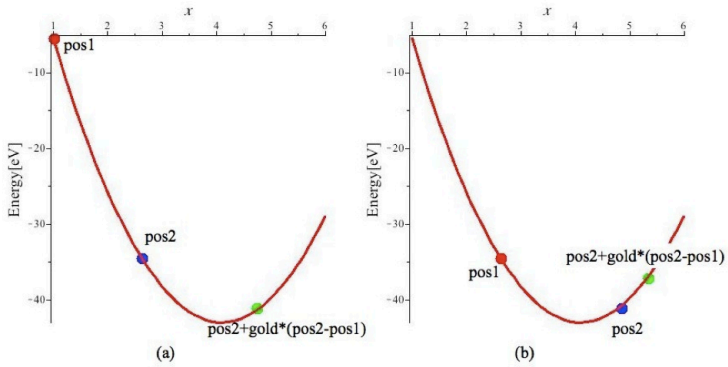

図 2.6: (a) 囲い込み中の 3 点 (b) 囲い込み終了時の 3 点.

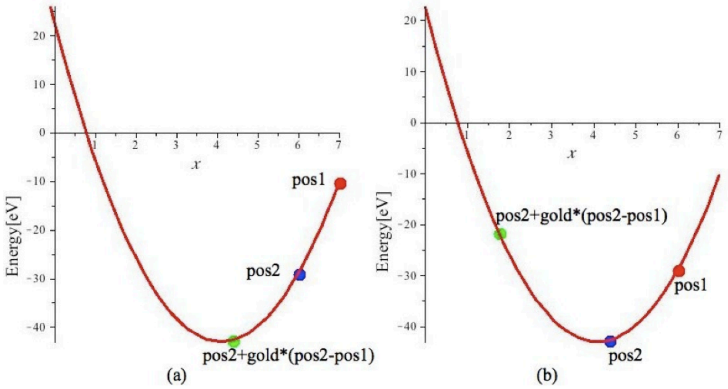

図 2.7: 初期値を極小の右側にとった時の (a) 囲い込み中の 3 点 (b) 囲い込み終了時の 3 点.

黄金分割法　黄金分割は区間 [a,c](a ＜ b ＜ c) に極小値を囲い込み，新しい点 x を a と b の間または b と c の間にとる．ここでも b は a から右に c - a の W 倍 (W ＜ 1) だけ進んだ点とすると，

$$\frac{b-a}{c-a} = W \tag{2.7}$$

$$\frac{c-b}{c-a} = 1-W \tag{2.8}$$

となる．また，x は b から右に c-a の Z 倍 (Z ＜ 1) だけ進んだ点 (図 2.8) とする．

$$\frac{x-b}{x-a} = Z \tag{2.9}$$

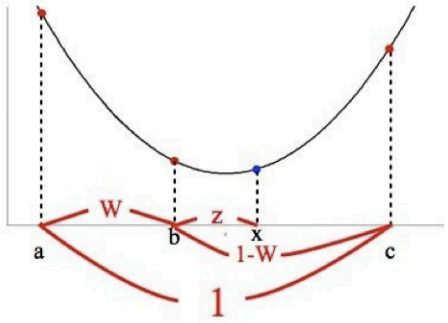

図 2.8: 新しい点の取り方．

次に x と b の関数値を比較して，関数値が大きい方の端点である a または c を除去する．図 2.8 の場合は b の関数値が x の関数値より大きいため，端点 a を除去する．すると，囲い込みの区間幅は，新しい 3 点が a , b , x の場合は現在の W+Z 倍，b , x , c の場合は現在の 1-W 倍となる．この際，W を大きい値と決めると，a を除去した場合は区間の狭まる幅は小さくなってしまう．そこで除去する点によって区間の挟まり方に偏りがなくなるようにするには区間 [a,b] と区間 [x,c] が等しくなるようにすればよい．つまり $|c-x| = W$ となるようにする．よって

$$Z = 1-2W \tag{2.10}$$

となる．これは $|b-a| = |x-c|$ となる位置に新しい点がくることになるため，x は区間 [a,b],[b,c] のうち広い方にくることになる．また区間 [b,c] に対する x の位置関係は，[a,c] に対する b の位置関係と同じになる．つまり，

$$\frac{Z}{1-W} = W \tag{2.11}$$

となる．よって式 (2.10),(2.11) より

$$W = \frac{3-\sqrt{5}}{2} \fallingdotseq 0.38197 \tag{2.12}$$

よって最適な囲い込みの 3 点 a < b < c は，中央の点 b を近い方の端点からの距離と遠い方の端点からの距離が 0.38197:0.61803 になるように選べばよい．この比は黄金比になっている．図 2.9 に黄金分割法が進んでいく様子を示した．図 2.9 の (a) の青で示した点，および図 2.9(b) の緑で示した点が新しい点 x, x' である．また，(a) の b , c は，(b) の a' , b' と更新される．黄金分割では 1 回で区間が全体の 0.61803 倍に減少する．この方法を用いることで，最初に極小値を囲い込んでいれば確実に最小値を求めることができる．

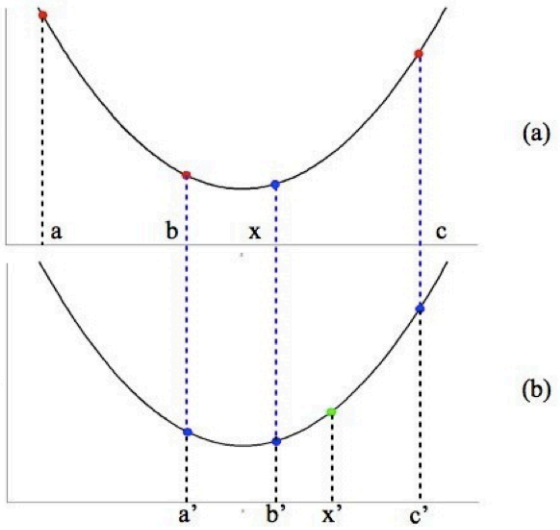

図 2.9: 黄金分割法．

2.5.2 Brent

Brent の方法とは，1 次元における最適化手法の 1 つで，黄金分割法と放物線補間を組み合わせたものである．黄金分割法による関数の最小化は導関数が連続でない場合にも最小かができる方法であるが，収束するまでには時間がかかる．そこで導関数が連続であるような関数の場合は計算時間の早い放物線補間を利用する．Brent の方法では，関数 mnbrak で得られた 3 点を入力とし，最小値を求める．Brent の方法の典型的な最終配置は両端点 a , b が 2 * x * 10^{-5} だけ離れる (式 (2.13)) さらに x は a と b の中点になる [5]．(式 (2.14))

$$b - a = 2 * x * 10^{-5} \tag{2.13}$$

$$x = \frac{b-a}{2} \tag{2.14}$$

よって，相対精度は $\pm 10^{-5}$ になる．

放物線補間放物線補間は初期値として極小を囲い込む 3 点 (図 7 の ①, ②, ③) を与え，関数値を求める．次に選んだ 3 点を通るような放物線をつくり，極小値の x 座標を

$$x = b - \frac{1}{2}\frac{(b-c)^2|f(b)-f(c)| - (b-c)^2|f(b)-f(a)|}{(b-c)|f(b)-f(c)| - (b-c)|f(b)-f(a)|} \tag{2.15}$$

から求める．この時の関数値 (図 7 の ④ と最初の 3 点を比較し，値の小さい 3 点を新たな 3 点として選び再度放物線をつくり，式 (2.15) を利用し極小値の x 座標を求める (図 2.10 の ⑤．この操作を繰り返すことで，もとの関数の極小に近づいていく [5]．Brent の方法で求めたエネルギーの変化を図 2.11(a) に，最安定位置におけるエネルギーとの差の両対数グラフを図 2.11(b) に示した．これらの図から，計算回数に応じてエネルギー値が収束していることがわかる．

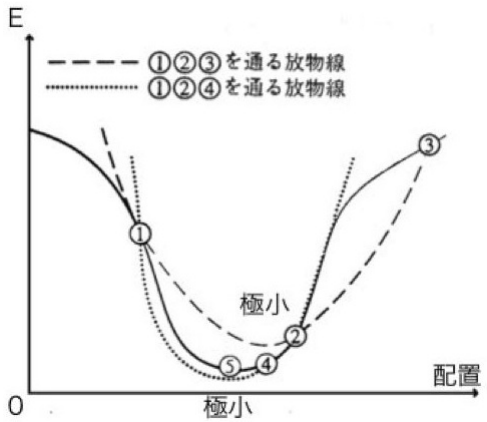

図 2.10: 放物線補間.

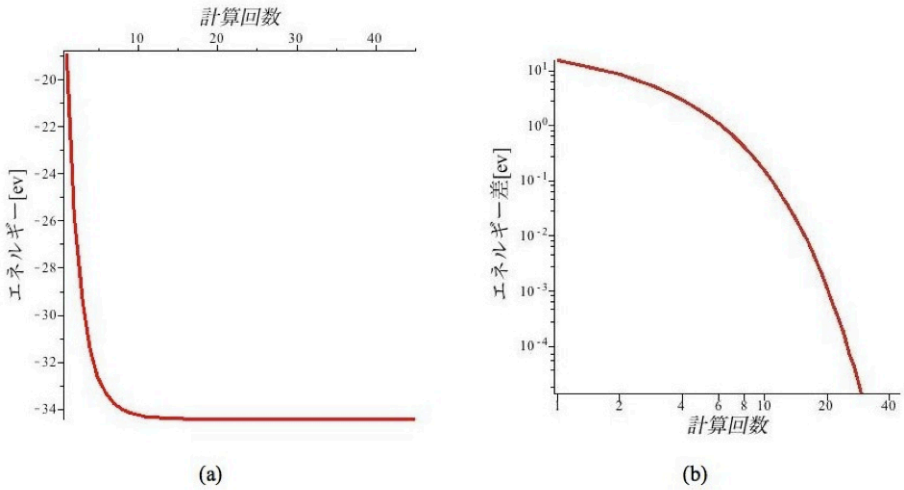

図 2.11: Brent 法の (a) 計算回数とエネルギーの変化　(b) 最小値とのエネルギー差.

2.5.3　CG 法

共役勾配 (Conjugate Gradient :CG) 法は，多次元空間での極小を求める手法で，点 P での関数値　$f(P)$ だけでなく，勾配 (1 階偏導関数) $\nabla f(P)$ も，計算できる場合に使用するものである．実際の計算で，まず初期位置からある方向における 1 次元の極小値を Brent 法を利用して求める．次に得られた極小の位置から再度別の方向に向かって 1 次元の極小値を求める．この作業を繰り返し行い，最終的に多次元での極小値にたどり着くというものである．1 次元の方向を決定する際，前回までのある方向にそった最小化が無駄にならないような方向を採用する必要がある．このような方向を「共役な方向」と呼ぶ．ある点を P_i とし，そこからある方向 u にそって極小 P_{i+1} まで進み終え，次に新しい方向 v にそって進もうとしているとする．この時，P_i から方向 u での最小化が無駄にならないようにするための条件は，$P[i+1$ における勾配ベクトルが u に垂直であることである．勾配ベクトルとは点 P_k おいて関数値を最も大きく増やす方向を示すベクトルである．CG 法では減少する方向のベクトルとして利用するため，この勾配ベクトルを逆向きにしたベクトルを利用する．以下この減少方向のベクトルを勾配ベクトルと表記する.「共役」の数学的な定義は N 次対称行列 A に対して二つのベクトル u, v が

$$u^t A v = 0 \tag{2.16}$$

を満たすとき u と v は互いに共役であるという．さらに A が単位行列のとき上式はベクトルの直行条件となるため,「共役」は「直行」の拡張概念であると考

えることができる．ここで g_iを座標 P_iにおける勾配ベクトルとして，各ループでの 1 次元の探索を行う方向である共役なベクトルの h_iを以下のように定める．

$$g_i = -\nabla f(P_i) \tag{2.17}$$

$$h_{i+1} = g_{i+1} + \gamma_i h_i \tag{2.18}$$

式 (2.18) で得られた共役な方向のベクトル h_{i+1}を利用すると，次の極小の位置 P_{i+2} は

$$P_{i+2} = P_{i+1} + \alpha h_{i+1} \tag{2.19}$$

と表すことができる．ここで αは定数とする．図 2.12 に点 P_iにおける探索の方向のベクトル h_iと点 P_{i+1}における勾配のベクトル g_{i+1}, 次の探索の方向である h_iと共役な方向のベクトル h_{i+1} を示した．

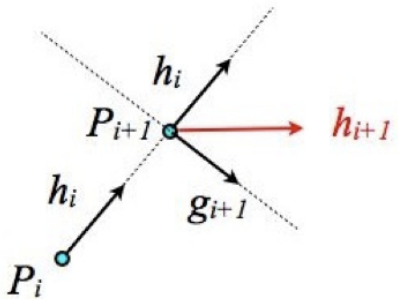

図 2.12: 共役な方向．

式 (2.12) の γ の決め方には

(i) Fletcher-Reeves 法

$$\gamma_i = \frac{g_{i+1} g_{i+1}}{g_i g_i} \tag{2.20}$$

(ii) Polak-Ribiere 法

$$\gamma_i = \frac{(g_{i+1} - g_i) g_{i+1}}{g_i g_i} \tag{2.21}$$

が知られている．最初に Fletcher-Reeves が式（2.20）を用いていた．しかしその後，Polak-Ribiere が式（2.21）を用いることを提案した．これらの式は厳密な 2 次形式の時のみ同値となる．今回用いるプログラムでは，Polak-Ribiere 法を用

いている．以下に 3 次元の関数を用いて実際に CG 法で極小値を求めるまでの過程を示す．図 2.13（a） にその関数のグラフと初期点 P_1 から極小値である P_{16} まで，図 2.13（b） にそのグラフを (x, y) 平面に等高線して示し，(a）と同様に初期点 P_1から P_{16} まで移動する様子を示している．ここで用いたモデルの関数は $f(x, y) = x^2 + y^2 + xy$ で初期位置 P_1 の（x,y）座標は（1.0, 9.0），極小値は，原点 （0.0, 0.0）である．

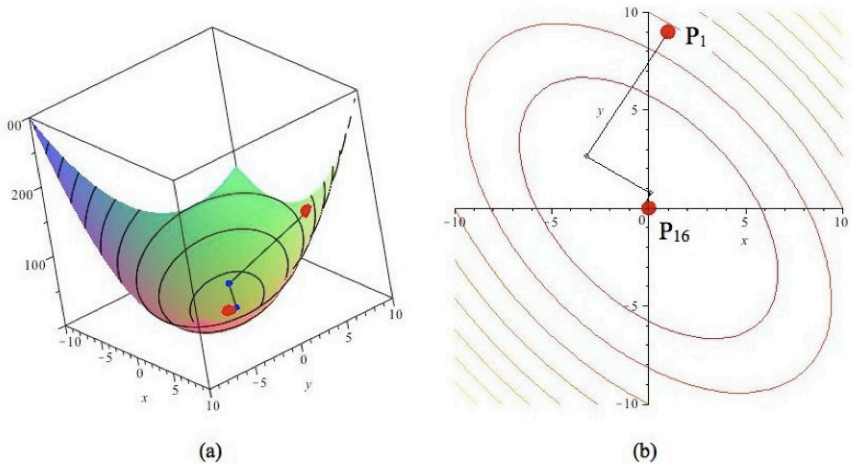

図 2.13: （a）3 次元表示における CG 法, （b）等高線表示における CG 法.

第3章　結果

作成した pseudoVASP を用いて，真嶋の作成した内部緩和プログラム (inner_relax) を実行した．その結果を VASP を用いて行った計算結果と比較し，pseudoVASP の有用性を調べた．なお検証には 2.1 節に述べた $2 \times 2 \times 2$ (32 原子) の Al スーパーセルを用いている．このスーパーセル中の Al 原子一個を最安定位置から移動させたモデルにおいて，内部緩和を行った．

3.1　初期状態

今回の実験では，Al スーパーセルの原子一個を x 軸方向に 0.05 移動させたものを初期状態として与えた．そのモデルを図 3.1 に示す．この原子モデルは，結晶構造の可視化プログラムである VESTA を用いて作成している．黄色で示した原子が，移動させた原子である．

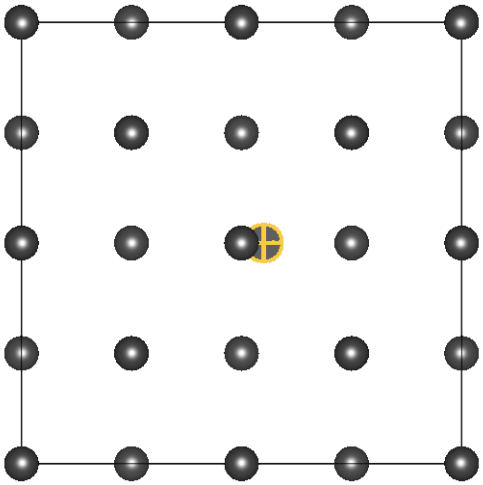

図 3.1: 内部緩和実行前の Al の原子位置．

ここで，対象となる原子における初期状態の座標とエネルギー，フォースを表 3.1 に示した．なお，このエネルギーとフォースの値は pseudoVASP を用いて求めたものである．pseudoVASP では最安定位置において一個の原子が -1.0 eV/atom のエネルギーを持つよう規格化している．今回は 32 個の原子をもつ Al のスー

パーセルを用いているので，系は最安定位置で -32.0 eV のエネルギーをもつことになる．比較のため，対象となる原子の最安定位置における値も表 3.1 に示しておく．その最安定位置を表した原子モデルは，2.1 節の図 2.2 で表した..

表 3.1: 緩和の対象となる原子の初期位置と最安定位置における座標とエネルギー，force の比較.

	初期位置	最安定位置
座標 [x,y,z]	[0.55,0.50.0.50]	[0.50,0.50,0.50]
Energy [eV]	-31.4910152	-32.0000000
force [x,y,z]	[-6.575195 , 0.000000 , 0.000000]	[0.000000 , 0.000000 , 0.000000]

3.2 内部緩和の実行

図 3.1 のモデルにおいて，inner_relax による内部緩和を行った．その実行結果を以下に示す．図 3.2 には pseudoVASP の実行回数と，緩和を行った原子の x 座標との関係を示している．計算の回数が増えるにつれて，x 座標が 0.50 に収束していることがわかる．ここで比較のため，VASP を用いて inner_relax を実行させた時の，VASP の実行回数と緩和を行った原子の x 座標との関係を図 3.3 に示した．この結果では計算を行われた回数は異なるものの，x 座標は pseudoVASP での計算値とほぼ同様の推移をたどり，最後は 0.50 に収束していることがわかる.

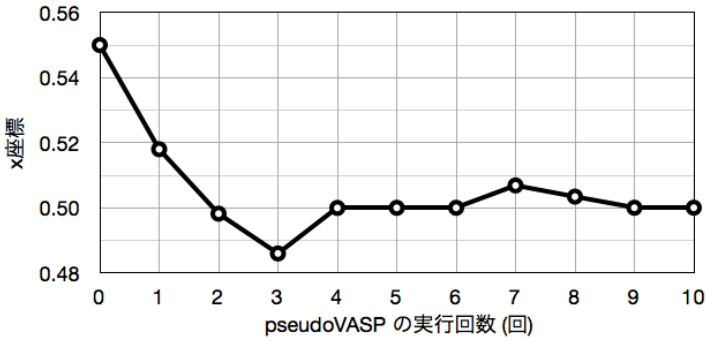

図 3.2: pseudoVASP における x 座標の実行回数による推移.

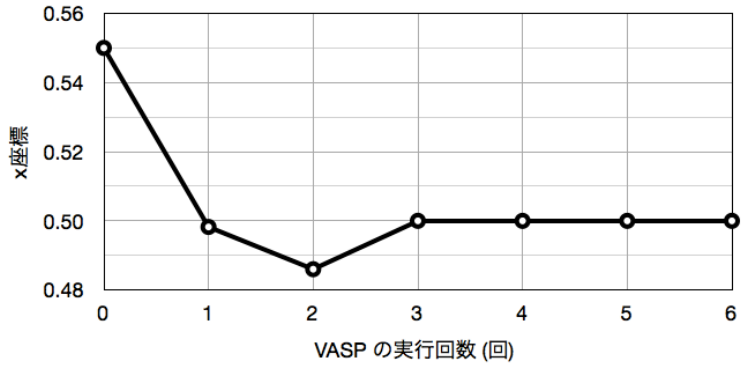

図 3.3: VASP における x 座標の実行回数による推移.

次に，エネルギーについて比較する．図 3.4 は，pseudoVASP の実行回数と，系のもつエネルギーの推移を表している．計算の回数が増えるにつれて，エネルギーの値が -32.0 eV に収束していることがわかる．ここで図 3.5 に VASP を用いた場合のエネルギーの推移を示した．VASP ではエネルギーを規格化せずに計算を行っているため数値は異なるが，pseudoVASP を用いた計算とほぼ同様の推移をたどり，エネルギーが収束している．

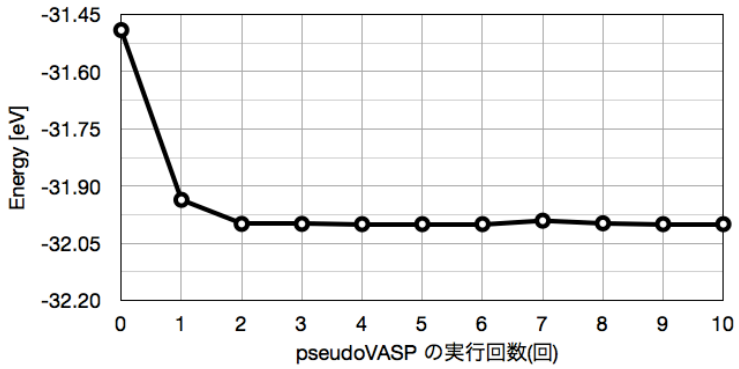

図 3.4: pseudoVASP におけるエネルギーの実行回数による推移.

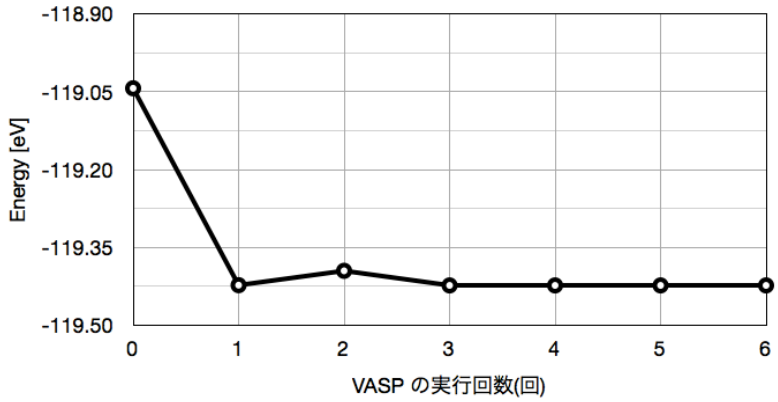

図 3.5: VASP におけるエネルギーの実行回数による推移.

図 3.6 には，pseudoVASP の実行回数と移動させた原子の受ける force の関係を表している．計算の回数が増えるにつれて，原子の受ける force は極めて 0 に近い値に収束した．図 3.7 には，VASP を用いた場合の force の推移を示した．force は，計算で求めたエネルギーの値を用いて行っている．つまり，pseudoVASP では規格化したエネルギーから force を計算しているが，VASP では規格化していないエネルギーから force を求めることになる．そのため計算した force の値は異なっているものの，VASP を用いた計算でも pseudoVASP による計算とほぼ同様の推移をたどり，force は極めて 0 に近い値に収束した.

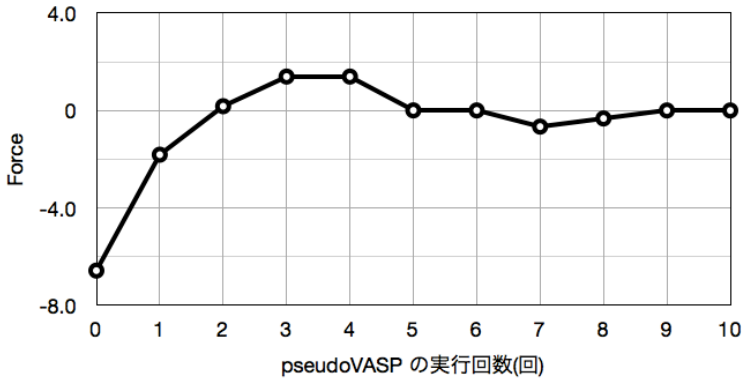

図 3.6: pseudoVASP における force の実行回数による推移.

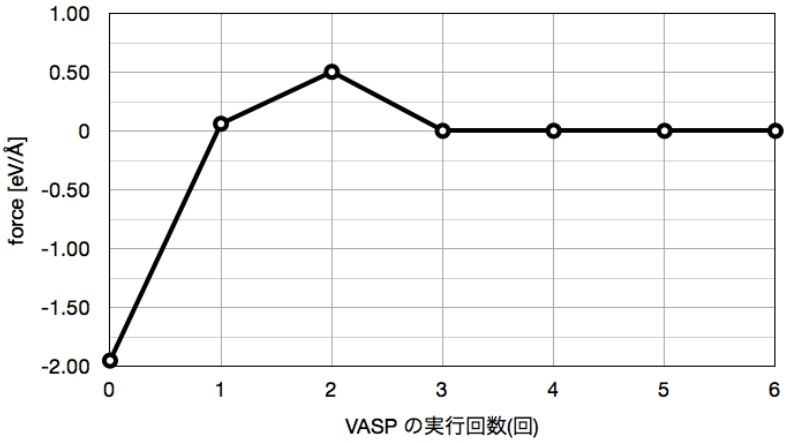

図 3.7: VASP における force の実行回数による推移.

ここで，pseudoVASP を用いた緩和終了後に作成した POSCAR をもとに，VESTA を用いて作成したモデルを図 3.8 に示す．図 3.1 と同様に黄色で印がついている原子が，移動した原子である．図 3.1 の緩和前のモデルと比較して，原子が元の位置に移動してることがわかる．以上の結果より，pseudoVASP を用いた内部緩和でも正しい位置に原子を移動させることができたと考えられる.

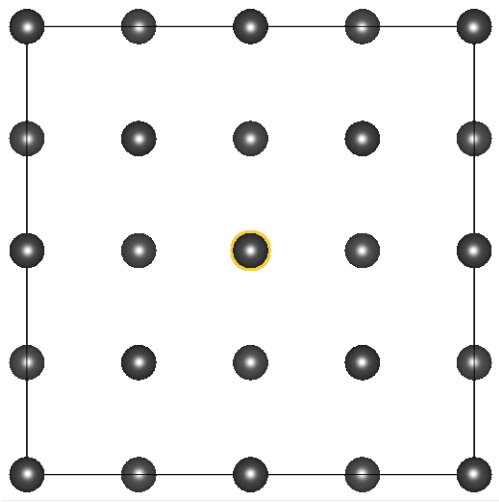

図 3.8: 内部緩和終了後の Al の原子位置.

3.3 VASP と psuedoVASP との計算時間比較

ここで，今回 pseudoVASP を用いて行った内部緩和の計算時間と，VASP を用いて行った同じモデルにおける内部緩和の計算時間との比較を行った．その結果が表 3.2 である．

表 3.2: VASP と pseudoVASP の内部緩和計算時間比較

	VASP	pseudoVASP
計算時間 (sec)	4885.04	69.49

この結果より，VASP の代わりに pseudoVASP を用いることで，作成したコントローラの動作確認時間を大幅に短縮できるということがわかった．

3.4 考察

以上の結果から，VASP を用いて計算を行うよりも短い時間で内部緩和のプログラムの動作を検証することができた.3.2 節で比較した原子座標やエネルギー，force の推移からわかるように，pseudoVASP を用いてコントローラを作動させても，VASP を用いた場合とほぼ同様の動作を行うということが示された．これにより，pseudoVASP が VASP の計算結果に極めて近い値を出力しているということがわかる．最小値探索における計算回数の違いは，1.2.2 節に述べた第一原理計算と原子間ポテンシャルによる計算精度の違いに起因すると考えられる．精度の高い計算を 6 回行うよりも，精度は劣るが短い計算時間で 10 回計算を行うほうが速く解を求めることができ，全体の計算時間も大幅に削減することができるという結果になった．

第4章　総括

第一原理計算ソフト VASP は，その高い計算精度ゆえに膨大な計算時間を要する．西谷研究室で作成している様々なコントローラでは，その計算の際に VASP を何度も実行しているので，本物を使っていてはコントローラ開発に多大な時間が必要となる．コントローラ開発にかかる時間を短くするため，第一原理計算よりも計算速度の速い原子間ポテンシャルを用いてエネルギー計算を行う pseudoVASP を作成した．その動作確認のため，今回は 2011 年度卒業生真嶋の作成した inner_relax を用いて動作の検証を行った．

今回の検証には，Al のユニットセルを $2 \times 2 \times 2$ に拡張したスーパーセルを用いた．そのモデル中の原子を一個移動させて inner_relax を行い，その結果を VASP を用いた時の結果と比較した．

その結果，pseudoVASP を用いた計算でも原子を安定な位置へ移動させることができ，その原子座標やエネルギー，原子にかかる force の推移も VASP を用いた計算とほぼ同様の結果を示した．また，その計算時間は VASP を用いて inner_relax を作動させた時に比べ，約 1/70 の時間に短縮することができた．

今後，コントローラの開発時に VASP の代わりに pseudoVASP を用いて動作確認を行うことで，短時間でのコーディングが可能である．その pseudoVASP の詳しい使用法及び解説は付録 A1 に示した．また現段階での課題点として，pseudoVASP から VASP へ切り替える際に一部コードを書き換えなくてはならないことが上げられる．そのため，今後はその切り替えをスムーズに行えるようにするためのインタフェースを開発することが望まれる．

関連図書

[1] 西谷滋人，西谷研究室 2011 年度在籍生,「はじめての VASP 原理から使用法,適用例まで」，(2011)

[2] 真嶋亨,「第一原理構造緩和法の実装」，(関西学院大学 理工学部　情報科学科　卒業研究 2011)

[3] 独立行政法人物質・材料研究機構 AtomWork http://crystdb.nims.go.jp/ （参照:2013/2/15）

[4] 西谷滋人,「固体物理の基礎」，(森北出版　2006)

[5] William H. Press 他,「ニューメリカルレシピ・イン・シー」，(技術評論社 1993)

謝辞

本研究を遂行する日々において，終始多大なる心温かい御指導，御鞭撻を頂いた西谷滋人教授に対し，深く感謝いたします．また，本研究の成就に至るまで，同研究室に所属する同輩達，ならびに山本洋佑先輩，坂本雄一先輩，大坪秀礎先輩，八幡裕也先輩方からの多くの御協力があり，そして知識の共有を頂いたことをここに記します．最後ではありますが，私を支えてくれた全ての友人，仲間，そして家族に対し，心より深く御礼申し上げます．

付 録A

A.1 pseudoVASP の使用方法

本研究において作成した pseudoVASP とは，基本的には POSCAR から原子座標を読み取りエネルギーや force を OUTCAR に書き込んで出力するプログラムである．コントローラ開発時において，VASP の代わりに pseudoVASP を用いることで，開発に要する時間を大幅に短縮することができる．しかし，VASP を用いた計算と pseudoVASP を用いた計算は同様の推移を示すものの，その計算方法や特性の違いにより，まったく同じコントローラを用いても必ずしも同じ挙動をする訳ではない．コントローラは pseudoVASP による動作確認が終了した後に，VASP に切り替えて計算を行うことを想定して作成される．そのため，pseudoVASP を用いたコントローラ開発の際にはいくつか注意すべき点がある．以下に，開発において pseuoVASP を使用する際の使用方法及び注意点について記載しておく．

A.1.1 pseudoVASP の実行

まず，pseudoVASP を用いて計算を行う際の，ファイルの配置について説明する．pseudoVASP の計算を行うプログラムである pseudoVASP.rb，コントローラのプログラム (ここでは Controller.rb)，POSCAR は同じディレクトリにおいておく．

ターミナル

```
/Users/haruka/Desktop/controller% ls
POSCAR controller.rb pseudoVASP.rb
```

POSCAR の内容は以下のようになっている．pseudoVASP.rb では，Direct 以下に表示されている原子座標を読み取り，配列に格納している．pseudoVASP.rb の詳細については付録 A.2 にて解説を行っている．

ターミナル

```
/Users/haruka/Desktop/controller% cat POSCAR
New structure
1.0
        8.0827999115         0.0000000000         0.0000000000
        0.0000000000         8.0827999115         0.0000000000
        0.0000000000         0.0000000000         8.0827999115
   32
Direct
     0.000000000         0.000000000         0.000000000
     0.000000000         0.000000000         0.500000000
     0.000000000         0.500000000         0.000000000
     0.000000000         0.500000000         0.500000000
     0.500000000         0.000000000         0.000000000
     0.500000000         0.000000000         0.500000000
     0.500000000         0.500000000         0.000000000
     0.250000000         0.250000000         0.000000000
     0.250000000         0.250000000         0.500000000
     0.250000000         0.750000000         0.000000000
     0.250000000         0.750000000         0.500000000
     0.750000000         0.250000000         0.000000000
     0.750000000         0.250000000         0.500000000
     0.750000000         0.750000000         0.000000000
     0.750000000         0.750000000         0.500000000
     0.250000000         0.000000000         0.250000000
     0.250000000         0.000000000         0.750000000
     0.250000000         0.500000000         0.250000000
     0.250000000         0.500000000         0.750000000
     0.750000000         0.000000000         0.250000000
     0.750000000         0.000000000         0.750000000
     0.750000000         0.500000000         0.250000000
     0.750000000         0.500000000         0.750000000
     0.000000000         0.250000000         0.250000000
     0.000000000         0.250000000         0.750000000
     0.000000000         0.750000000         0.250000000
     0.000000000         0.750000000         0.750000000
     0.500000000         0.250000000         0.250000000
     0.500000000         0.250000000         0.750000000
     0.500000000         0.750000000         0.250000000
     0.500000000         0.750000000         0.750000000
     0.550000000         0.500000000         0.500000000
```

次に，用意した 3 つのファイルを使い，pseudoVASP を実行する．実行の過程を以下に示す．

ターミナル

```
/Users/haruka/Desktop/controller% ls
POSCAR  controller.rb pseudoVASP.rb
/Users/haruka/Desktop/controller% ruby controller.rb
/Users/haruka/Desktop/controller% ls
OUTCAR POSCAR  controller.rb pseudoVASP.rb
```

ここで計算終了後に作成された OUTCAR ファイルを，以下に示しておく，OUTCAR ファイルには，計算終了後のエネルギー，各原子座標とそれぞれにかかるフォース，計算所要時間が表示されている．

ターミナル

```
/Users/haruka/Desktop/controller% cat OUTCAR
FREE ENERGIE OF THE ION-ELECTRON (eV)
---------------------------------------------------------------
free energy TOTEN =   -30.9804100 eV

POSITION                                       TOTAL-FORCE (eV/Angst)
-----------------------------------------------------------------
  0.000000   0.000000   0.000000   0.366345   0.245457  -0.552202
  0.000000   0.000000   0.500000  -0.695252  -0.135366  -0.810597
  0.000000   0.500000   0.000000  -0.018752  -0.379694  -0.158079
  0.991035   0.512303   0.488604   0.289987  -0.783470   1.134135
  0.500960   0.001832   0.981616  -0.697994  -0.427772   1.464761
  0.490392   0.007043   0.517273   1.215426  -0.781514  -1.897621
  0.500000   0.500000   0.000000  -0.021804  -0.151253   0.161329
  0.250000   0.250000   0.000000  -0.011915  -0.016517  -0.023442
  0.250000   0.250000   0.500000  -0.160209   0.838411  -0.210219
  0.250000   0.750000   0.000000   0.173685   0.075229  -0.177949
  0.250000   0.750000   0.500000   0.083870  -0.231332  -0.230893
  0.752834   0.235575   0.987648  -0.242530   1.567072   1.319071
  0.750000   0.250000   0.500000   1.442839  -1.790269   0.000615
  0.752078   0.747713   0.988358  -0.642369   0.224816   0.705485
  0.750000   0.750000   0.500000   0.996705   2.339488  -0.084640
  0.250000   0.000000   0.250000   0.187582  -0.039199  -0.076575
  0.237448   0.004947   0.732714   0.600377  -0.553684   2.357078
  0.240240   0.497728   0.262053   1.149135   0.408968  -1.048453
  0.250000   0.500000   0.750000   0.081789   0.007676  -0.365912
  0.750000   0.000000   0.250000  -0.555013   0.098375  -0.056001
  0.750000   0.000000   0.750000   0.447721  -0.440229  -0.948195
  0.750000   0.500000   0.250000   1.045864  -0.141066  -2.448367
  0.750000   0.500000   0.750000   1.472044  -0.258939   1.574204
  0.000000   0.250000   0.250000  -0.273168  -0.168205  -0.092449
  0.000000   0.250000   0.750000  -0.045363   0.484353  -0.390805
  0.991005   0.755977   0.230557   0.640484  -0.473890   1.621199
  0.000000   0.750000   0.750000   0.133897  -0.091424  -0.205451
  0.500000   0.250000   0.250000   0.058411  -0.113454  -0.120693
  0.500000   0.250000   0.750000  -0.191604   0.853255  -0.063766
  0.500000   0.750000   0.250000   0.010497   0.164076  -0.052725
  0.500000   0.750000   0.750000  -0.277836  -0.334197  -0.326967
  0.550000   0.500000   0.500000  -6.562851   0.004296   0.004122
Total CPU time used:         0.01947
```

A.1.2　コントローラ作成における注意点

西谷研究室ではこれまでに様々なコントローラを開発し，来年度以降も開発を行う予定である．ここでは，その開発時における注意点をいくつか説明する．

1. 計算の呼び出し

   pseudoVASP よる計算を行う場合，コントローラ内部の計算を行いたい場所で ruby pseudoVASP.rb POSCAR >> OUTCAR を呼び出す．これにより，POSCAR から原子座標を読み取り，計算したエネルギーとフォースが書き込まれた OUTCAR が出力される．計算を VASP に切り替える場合は，先ほどの記述を qsub run.sh のように VASP の計算を呼び出すコードに書き換える．

ターミナル

```
pseudoVASP の場合
system("ruby pseudoVASP.rb POSCAR >> OUTCAR")

VASP の場合
system("qsub run.sh")
```

2. 計算の待機時間

pseudoVASP は計算が高速なので計算後すぐに OUTCAR を作成し，スムーズに処理を行うことができる．しかし，VASP では OUTCAR 作成に時間がかかるため，計算を実行した後 OUTCAR が作成されるまで処理を行わないようにする必要がある．まず，pseudoVASP の場合，下記のようなプログラムでも正常に動作を行う．ここでは，前回の計算結果を削除し，読み込んだ POSCAR から新たな OUTCAR を作成した後すぐにその OUTCAR を用いた処理を行っている．

ターミナル

```
system("rm -rf OUTCAR")
system("ruby pseudoVASP.rb POSCAR >> OUTCAR")
OUTCAR の処理....
```

しかし，このプログラムをそのまま VASP へ移行しても正常に動作を行うことはない．なぜなら，VASP のエネルギー計算は pseudoVASP に比べて時間がかかりすぎてしまうため，OUTCAR の作成に時間がかかり，すぐに OUTCAR の処理を行うことができないからである．なので OUTCAR が作成されるまで，次の処理を行わないでおく必要がある．よって，プログラムを下記のように書き換える．

ターミナル

```
system("rm -rf vasp-job.*")
system("rm -rf haruka.vasp.*")
system("rm -rf OUTCAR")

sleep(3)
system("qsub run.sh")
sleep(5)

loop do
 if find == true then
   pp "check2"
   break
 end
 pp "wait 1min"
 sleep(60)
end
```

最初の三行は前回の計算結果を削除している．関数 find は以下のように定義しておく．これは OUTCAR が作成完了しているかを調べる関数である．OUTCAR が作成されていればその中身を一列ずつ読み込み， 計算完了後

の OUTCAR に書き込まれる Total CPU time used という文字を確認することで OUTCAR 作成完了と見なしている．関数 find を用いることで，OUTCAR が完全に作成されていれば次の処理へ，そうでなければ 60 秒待機するという処理を繰り返し行えるようになる．これらの処理は pseudoVASP を用いた場合でも不具合無く計算を行うことができるため，pseudoVASP を用いた動作確認時からこの形式でプログラムを作成しておくことを勧める．

ターミナル

```
def find
  if File.exist?("OUTCAR") then
    File.readlines("OUTCAR").reverse_each do |line|
      if line.include?("Total CPU time used")
        pp "find"
        return true
      end
    end
  end
end
```

3. 座標変換

pseudoVASP の計算は，すべて相対座標で行われている．また，POSCAR に表示される原子座標も，相対座標で表示されている．しかし VASP では精確な値を算出しているため，出力される OUTCAR に記載されている原子座標や force は絶対座標を用いた値で表されている．そのため，コントローラにおいて VASP で作成した OUTCAR から座標やフォースを読みとって計算した値も，絶対座標で示される．なので，その値を用いて新たな POSCAR を作成する際には絶対座標を相対座標に変換する必要がある．その変換を行う関数の例として，本研究で用いた inner_relax に使用されている座標変換を行う関数 position を以下に示す．

変数 poscar は POSCAR を一行ずつ読み込んで格納している配列である．そこから基本並進ベクトルを取り出したものを配列 a として作成している．また，配列 move_atom には今回の緩和で移動させた原子座標が格納されていて，ここに a の逆数をかけることで絶対座標への変換を行っている．

ターミナル

```
def position(coord)
  pos=poscar[2][0].to_f
  a=Matrix[[pos,0,0],[0,pos,0],[0,0,pos]]
  ai=a.inv
  move_atom=Vector[coord[0],coord[1],coord[2]]
  m=ai*move_atom
  return m
end
```

pseudoVASP を用いた計算の場合，これらの処理は不要なので注意が必要である．

A.2　コード

ここでは，本研究において作成した pseudoVASP のプログラム，また今回使用した真嶋の inner_relax のプログラムに使用したコードを示しておく．

まず，そのファイルの配置について説明する．inner_relax では，付録 A.1 に示した controller.rb に対応する main.rb の他に，計算に使用する関数を ”ruby” というディレクトリに配置し，require コマンドでそれぞれの関数を呼び出している．main.rb，pseudoVASP，POSCAR は ”vasp” というディレクトリに配置しておく．

ターミナル

```
/Users/haruka/Desktop/pVASP_test1% ls
vasp  ruby

/Users/haruka/Desktop/pVASP_test1/ruby% ls
atom.rb      f1dim.rb      linmin.rb       naighbor.rb
brent.rb      find.rb      makepos.rb    relax1.rb
f1dim.rb     frprmn.rb    mnbrak.rb

/Users/haruka/Desktop/pVASP_test1/vasp% ls
POSCAR  main.rb  pseudoVASP.rb
```

A.2.1　pseudoVASP.rb

今回作成した，原子座標からエネルギー，force を求めて出力するコード．計算の流れは 2.1 節に示している．ここではそのコードを記しておく．

```
include Math
require 'pp'

$m = 1.000

class Atom
  attr_accessor :pos, :pos0, :nl

  def initialize(pos)
    @nl=[]
    @pos=Array.new(pos)
    @pos0=Array.new(pos)
  end

  def energy()
   return lj_energy()
  end

  def force()
    return lj_force()
  end

  def reset_nl()
    @nl=[]
  end

#LJ ポテンシャルにより，エネルギーを計算している
  def lj_energy()
    a0=1.587401051
    e0=-1*4.0/12.0
    ene=0.0
```

```
    nl.each do |j|
      r=distance(@pos,$atom_list[j].pos)
      a=1.0/(a0*r)
      ene+=e0*((a**6)-(a**12))
    end
    return ene
  end

#LJ ポテンシャルにより求めたエネルギーを微分し，force を計算している.
  def lj_force()
    a0=1.587401051
    e0=-1*4.0/12.0
    f=[0.0,0.0,0.0]
    nl.each do |j|
      t=f_distance(@pos,$atom_list[j].pos)
      x=t[0]
      y=t[1]
      z=t[2]
      r=x**2+y**2+z**2
      dedr=-e0*(12/(a0**12*(r)**7)-6/(a0**6*(r)**4))
      f[0] += x*dedr
      f[1] += y*dedr
      f[2] += z*dedr
    end
    printf("%10.6f %10.6f %10.6f %10.6f %10.6f %10.6f\n",pos0[0],pos0[1],pos0[2],f[0],f[1],f[2])
    return f
  end
end

#POSCAR の Direct 以下の部分を読み取り，配列 atom_list に格納している..
def makeLattice()
  file = open(ARGV[0])
  atom_list=[]
  sw=false
  while line = file.gets do
    if /Direct/ =~ line then
      sw=true
    elsif sw then
      pos0=line.chomp.split(" ")
      pos1=[]
      pos0.each do |p|
        pos1 << p.to_f
      end
      atom_list << Atom.new(pos1)
    end
  end
  file.close
  return atom_list
end

$nx=2
$ny=2
$nz=2
$theta=atan(0.0/8.0)
$L=[$nx,$ny,$nz]

#二原子間の距離を求める
def distance(a,b)
  tmp=0
  for i in 0..2 do
      x=a[i]*2-b[i]*2
      x=x-(x/$L[i]).round*$L[i]#周期的境界条件
      tmp+=x*x
  end
  return sqrt(tmp)
end
```

```
#二原子間の距離を x,y,z 各成分ごとに求める
def f_distance(a,b)
  t=[]
  for i in 0..2 do
    x=a[i]*2-b[i]*2
    x=x-(x/$L[i]).round*$L[i]#周期的境界条件
    t << x
  end
  return t
end

def Atom_index
  atom_index=[]
  $atom_list.each_with_index do |i_a,idx|
     atom_index << idx
  end
  return atom_index
end

#ネイバーリストの作成.
def makeNL()
  $atom_list.each do |ai| ai.reset_nl end
  nmax=$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)<0.9*$m then
        ai.nl << j
        aj.nl << i
      end
    end
  end
end

#系のエネルギーを求める
def total_E()
  total_E=0.0
  $atom_index.each do |i|
    total_E+=$atom_list[i].energy()
  end
  return total_E
end

#force の計算
def force_all()
  force_all=0.0
  $atom_index.each do |i|
    force_all=$atom_list[i].force()
  end
  return force_all
end


#OUTCAR への書き出し
$t0=Time.now
$atom_list=makeLattice()
$atom_index=Atom_index()
makeNL()
printf("FREE ENERGIE OF THE ION-ELECTRON (eV)\n")
printf("---------------------------------------------------------------------\n")
printf("free energy TOTEN =   ")
printf("%10.7f eV\n",total_E())
printf("\n")
printf("POSITION                                        TOTAL-FORCE (eV/Angst)\n")
printf("---------------------------------------------------------------------\n")
force_all()
```

```
printf("Total CPU time used:      %10.5f\n",Time.now-$t0)
```

A.2.2 inner_relax

main.rb

作成したコードのメインとなるコード．ディレクトリの整理と変数の初期化，初期値におけるエネルギー計算を行っている．最初に”vasp”ディレクトリの中の”result”に前回の結果が入っている場合はすべて消去されるので注意が必要である．

```
t0=Time.now
require "../ruby/frprmn.rb"
require "../ruby/relax1.rb"
require "../ruby/find.rb"
require 'pp'

#残っている結果の削除
system("rm -rf ./result")
system("mkdir ./result")
system("mkdir ./result/result")
system("rm -rf ./subPOSCAR")
system("cp POSCAR subPOSCAR")

#残っている VASP の出力を削除
system("rm -rf vasp-job.*")
system("rm -rf haruka.vasp.*")
system("rm -rf OUTCAR")
pp "delete OUTCAR"

sleep(3)
system("ruby pseudoVASP.rb POSCAR > OUTCAR")
sleep(5)

loop do
  if find == true then
    pp "check2"
    break
  end
  pp "wait 1min"
  sleep(60)
end

#次元:n
n=3
n=n-1

xfret=[0.0]
#原子座標の取得
p1=outfile
pp p1
$ncom=n
$pcom=Array.new(n+1)
$xicom=Array.new(n+1)

pp frprmn(p1,n)

t1=Time.now
system("rm -rf vasp-job.*")
system("rm -rf haruka.vasp.*")
print t1-t0,"s","\n"
```

frprmn.rb

関数 frprmn は CG 法を実行している．ただし実際にこの関数が行っているのは，前回の探索方向を受け取り，Fletcher-Reeves 法または，Polak-Ribiere 法で新たな探索方向を作成している．

```
require 'pp'
include Math
 require "../ruby/linmin.rb"
 require "../ruby/relax1.rb"
 require "../ruby/f1dim.rb"

def frprmn(p1,n)
  ftol=1.0e-4
  eps=1.0e-4
  delta_e=1.0e-4
  g=Array.new(n+1)
  h=Array.new(n+1)
  xi=Array.new(n+1)

  fp=outene
  xi=outforce

  for j in 0..n do
    g[j]=xi[j]
    h[j]=xi[j]
  end
#関数 linmin の呼び出し
  for its in 0..200 do
    xi,xfret,p1=linmin(p1,xi,n)

    File.open("result.txt","a+"){ |file|
      file.puts(p1,xfret[1])
    }
    system("mv result.txt ./result/result")
    system("mv ./result/result ./result/result#{its+1}")
    system("mkdir ./result/result")
    printf("%10.5f\n%10.5f\n",xfret[1],outene)
    if (xfret[1]-outene).abs <= delta_e then  #エネルギー差の条件判定
      return "fin."
    end
    dgg=0.0
    gg=0.0

    for j in 0..n do
      gg+=g[j]*g[j]
    #  dgg+=xi[j]*xi[j] # Fletcher-Reeves
      dgg+=(xi[j]+g[j])*xi[j] #Polak-Ribiere
    end

    if gg==0.0 then
      print "gg-frp ",xfret,"\n"
      return xfret
    end
    gam=dgg/gg
    for j in 0..n do
      g[j]=-xi[j]
      h[j]=g[j]+gam*h[j]
      xi[j]=g[j]+gam*h[j]
    end
  end
  return p1
end
```

linmin.rb

関数 linmin は main.rb で初期化した大域変数の定義と mnbrak , brent への数値の受け渡しを行う仲介のための関数である.

```
require 'pp'
include Math
 require "../ruby/mnbrak.rb"
 require "../ruby/brent.rb"

def linmin(p1,xi,n)
  for j in 0..n do
    $pcom[j]=p1[j]#座標
    $xicom[j]=xi[j]#探索方向
  end
  ax,xx,cx=mnbrak
  xfret=brent(ax,xx,cx)[0]
  p1=outfile
  xi=outforce
   printf("p1")
   pp p1
  return xi,xfret,p1
end
```

mnbrak.rb

関数 mnbrak は極小を囲い込むためのものである. 計算の詳細については 2.5.1 節にて説明を行っている.

```
include Math
require 'pp'
require "../ruby/makepos.rb"
require "../ruby/naighbor.rb"
require "../ruby/atom.rb"
require "../ruby/f1dim.rb"

#同じ符号の値を返す.
def mysign(a,b)
  if b<0 then
    return -a
  else return a
  end
end

#大きい方の値を返す.
def fmax(a,b)
  if a>b then
    return a
  else return b
  end
end

def mnbrak
  gold=(1+sqrt(5))/2
  p1=0.0
  p2=naighbor[2]/10  #最近接原子間距離の 10 分の 1
  pp  fp1=outene
  system("mkdir ./result/result/mnres1")
  system("cp *CAR ./result/result/mnres1")
  system("cp *rb ./result/result/mnres1")
  pp  fp2=f1dim(p2)
  system("mkdir ./result/result/mnres2")
  system("cp *CAR ./result/result/mnres2")
  system("cp *rb ./result/result/mnres1")
```

```
if fp2>fp1 then
  tmp=p1
  p1=p2
  p2=tmp
  tmp=fp2
  fp2=fp1
  fp1=tmp
end

  p3=p2+gold*(p2-p1)
pp  fp3=f1dim(p3)
pp [p1,p2,p3]
pp "all fp"+fp1.to_s+" "+fp2.to_s+" "+fp3.to_s
pp "----------------------------------------"
system("mkdir ./result/result/mnres3")
system("cp *CAR ./result/result/mnres3")
system("cp *rb ./result/result/mnres1")

i=0

while fp2>fp3 do
  i=i+1
  r=p2-p1*fp2-fp3
  q=p2-p3*fp2-fp1
  u=p2-((p2-p3)*q-(p2-p1)*r)/(2.0*mysign(fmax((q-r).abs,1.0e-20),q-r))
  ulim=p2+100.0*(p3-p2)
  if (p2-u)*(u-p3)>0.0 then #a
    fu=f1dim(u)
    system("mkdir ./result/result/mnres#{i+3}")
    system("cp *CAR ./result/result/mnres#{i+3}")
    system("cp *rb ./result/result/mnres1")

    if fu<fp3 then
      p1=p2
      p2=u
      fp1=fp2
      fp2=fu
      break
    elsif fu>fp2 then
      p3=u
      fp3=fu
      break
    end
    u=p3+gold*(p3-p2)
    fu=f1dim(u)
    system("mkdir ./result/result/mnres#{i+3}")
    system("cp *CAR ./result/result/mnres#{i+3}")
    system("cp *rb ./result/result/mnres1")

  elsif (p3-u)*(u-ulim) >0.0 then #b
    fu=f1dim(u)
    system("mkdir ./result/result/mnres#{i+3}")
    system("cp *CAR ./result/result/mnres#{i+3}")
    system("cp *rb ./result/result/mnres1")
    if fu<fp3 then
      p2=p3
      p3=u
      u=p3+gold*(p3-p2)
      fp2=fp3
      fp3=fu
      fu=f1dim(u)
      system("mkdir ./result/result/mnres#{i+3}")
      system("cp *CAR ./result/result/mnres#{i+3}")
      system("cp *rb ./result/result/mnres1")
    end
```

```
    elsif (u-ulim)*(ulim-p3)>0.0 then #c
      u=ulim
      fu=f1dim(u)
      system("mkdir ./result/result/mnres#{i+3}")
      system("cp *CAR ./result/result/mnres#{i+3}")
      system("cp *rb ./result/result/mnres1")
    else #d
      u=p3+gold*(p3-p2)
      fu=f1dim(u)
      system("mkdir ./result/result/mnres#{i+3}")
      system("cp *CAR ./result/result/mnres#{i+3}")
      system("cp *rb ./result/result/mnres1")


    end
    p1=p2
    p2=p3
    p3=u
    fp1=fp2
    fp2=fp3
    fp3=fu
  end
  return p1,p2,p3
end
```

brent.rb

2.5.2 節にあるような手法で 1 次元の最小値を求めるための関数である．なお，関数 mnbrak と関数 brent における各回の計算終了後毎回 ”vasp ”ディレクトリ内の ”result ”というディレクトリにバックアップとして全ファイルをコピーしている．

```
include Math
require 'pp'
require "../ruby/relax1.rb"
require "../ruby/mnbrak.rb"
require "../ruby/makepos.rb"
require "../ruby/f1dim.rb"

#a が b より大きければ c を，そうでなければ d を返す関数
def hatena(a,b,c,d)
  if a>=b then return c
  else return d
  end
end


def brent(p1,p2,p3)
  fret=[0,0]
  delta_x=1.0e-5
  tol=1.0e-5
  cgold=1.0-(sqrt(5.0)-1)/2
  zeps=1.0e-5
  e=0.0
  d=0.0
  ax=p1
  bx=p2
  cx=p3

  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=f1dim(bx)
fv=fw
fx=fw

count=0

for j in 1 .. 150 do
  count = count + 1
  pp count
  pp outfile
  printf("%10.7f\n",[x,fx][1])
  printf("--------------------------------\n")
  xm=0.5*(a+b)
  tol1=tol*x.abs+zeps
  tol2=2.0*tol1

  #座標の収束判定
  if (x-xm).abs <= delta_x then
    xmin=x
    fret[1]=xmin
    fret[2]=fx
    printf("fin\n")
    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 or p <= q*(a-x) or 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 or (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=f1dim(u)
  system("mkdir ./result/result/brentres#{j}")
  system("cp *CAR ./result/result/brentres#{j}")
  system("cp *rb ./result/result/brentres#{j}")
  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 or w=x then
        v=w
        w=u
        fv=fw
        fw=fu
      elsif fu <= fv or v=x or v=w then
        v=u
        fv=fu
      end
    end
  end

  xmin=x
  fret[1]=xmin
  fret[2]=fx
  return fret[1,2],count#[[position,energy],count]
end
```

f1dim.rb

関数 frprmn によって得られた 1 次元の方向に新たな探索点をとり，pseudoVASP を実行してエネルギーを求める関数である.

```
include Math
require 'pp'
require "../ruby/relax1.rb"
require "../ruby/find.rb"

#関数 func を 1 次元に変換
def f1dim(x)
  ncom=$ncom
  pcom=$pcom
  xicom=$xicom

  xt=Array.new(ncom+1)

  long=0
  for j in 0..ncom do
    long+=xicom[j]**2
  end
  long=sqrt(long)

  for j in 0..ncom do
    xt[j]=pcom[j]+x*(xicom[j]/long)
  end

  mkposfile(xt)
  path=File.expand_path(".")
```

```
  system("rm -rf "+path+"/vasp-job.*")
  system("rm -rf "+path+"/haruka.vasp.*")
  system("rm -rf "+path+"/OUTCAR")
  pp "delete OUTCAR"
  sleep(5)

  #pseudoVASP の呼び出し
  system("ruby "+path+"/distest2.rb POSCAR > OUTCAR")


  loop do
    if find == true then
      pp "check2"
      break
    end
    pp "wait 1min"
    sleep(10)
  end
  return outene
end
```

makepos.rb

このコードで，POSCAR の書き換えを行っている．今回は pseudoVASP の使用に合わせ，相対座標で書き換えを行っているが，VASP で用いる際には付録 A.1 に示した変更を行う必要がある．

```
require 'pp'
require "../ruby/relax1.rb"
require 'matrix'

def position(coord)
  poscar=direct
  pos=poscar[2][0].to_f
  a=Matrix[[pos,0,0],[0,pos,0],[0,0,pos]]
  ai=a.inv
  move_atom=Vector[coord[0],coord[1],coord[2]]
  return move_atom
end

def mkpos(coord)#POSCAR の変更
  poscar=[]
  a=[]
  line=[]
  newpos=""
  n = 0
  num = 0
  File.open("subPOSCAR").each do |l|
    n = n + 1
    poscar.push(l)
    if l == "Direct\n"
      num = n
    end
  end

  d=10**10
  for i in 0 ..2 do
    line[i]=position(coord)[i]
    line[i]=((line[i].to_f*d).floor)/d.to_f
    a[i]=line[i].to_s
  end

  newpos =  ("  ")
  for i in 0..2 do
```

```
    b=""
    for j in 0..17-a[i].size do
      b = b + "0"
    end
    newpos += a[i] + b + (" ")
  end
  poscar[num + infile - 1] = newpos
  return poscar
end


def mkposfile(coord)#ファイルの書き換え
  system("cp POSCAR subPOSCAR")
  File.open("POSCAR","w") {|file|
    file.puts(mkpos(coord))}
end
```

naighbor.rb

関数 naibor は OUTCAR から最近接原子位置と距離を計算するものである.

```
include Math
require 'pp'
require "../ruby/relax1.rb"

def naighbor#OUTCAR から最近接原子位置と距離
outcar=[]#OUTCAR
mainline=[]#基準行指定
compline=[]#比較行指定
position=[]#座標
  File.open("OUTCAR").each do |line1|
    outcar.push(line1)
  end
  for i in 0..outcar.size-1
    if outcar[i] ==  "POSITION                                       TOTAL-FORCE (eV/Angst)\n"
      number = i
    end
  end

  outcar.slice!(0, number)
  mainline = outcar[infile + 1].split(" ")

  for i in 0..infile-1 do
    compline << outcar[infile + 1 - i].split(" ")
  end
  compline.reverse!

  for i in 0..compline.size-1 do
    for j in 0..compline[i].size-1
      compline[i][j] = compline[i][j].to_f
    end
  end

  for i in 0..2 do
    position << mainline[i].to_f
  end

  distance = 100
  count = 1
  for i in 0..compline.size-2
    if distance > sqrt((position[0]-compline[i][0])**2+(position[1]-compline[i][1])**2+(position[2]-compline[i][2])**2)
      distance = sqrt((position[0]-compline[i][0])**2+(position[1]-compline[i][1])**2+(position[2]-compline[i][2])**2)
      count = count + 1
    end
  end
```

```
  return compline[count],position,distance#[最近接原子の座標とエネルギー][着目原子の座標とエネルギー][最
近接原子間距離]
end
```

relax1.rb

ここでは，POSCAR と OUTCAR から情報を取得するための関数がある．関数 str は引数として与えたキーワードとなる文字列を正規表現に変換し，マッチする行数を取得する．

```
require 'pp'

#OUTCAR から文字列を読み込む
def str(x)
  outcar=[]
  iter=0
  File.open("OUTCAR").each do |line1|
    outcar.push(line1)
    iter+=1
    if Regexp.new(x)  =~ line1
      $number = iter
    end
  end
  outcar.slice!(0, $number)
  return outcar
end

#POSCAR の Direct 以降を読み込む
def direct
  poscar=[]
  num=0
  File.open("subPOSCAR").each do |line|
    poscar.push(line.chomp.split(" "))
    if line == "Direct\n"
      break
    end
  end
  return poscar
end

#POSCAR に記載されている原子の数
def infile
  num=0
  poscar=direct
  for i in 0..poscar[-2].size
    num = num + poscar[-2][i].to_i
  end
  return num
end

#OUTCAR に記載されている原子座標
def outfile#position from OUTCAR
  lookatom=[]
  position=[]
  outcar = str("TOTAL-FORCE")
  lookatom = outcar[infile].split(" ")
  for i in 0..2 do
    position[i] = lookatom[i].to_f
  end
  return position
end

#OUTCAR からエネルギーを取り出す
def outene
```

```
  lookene=[]
  position=[]
  outcar=str("ION-ELECTRON")
  lookene = outcar[1].split(" ")
  energy = lookene[4].to_f
  return energy
end

#OUTCAR から force を取り出す
def outforce#force from OUTCAR
  lookatom=[]
  force=[]
  outcar=str("TOTAL-FORCE")
  lookatom = outcar[infile].split(" ")

  for i in 0..2 do
    force[i] = lookatom[i+3].to_f
  end
  return force
end
```