

# 卒業論文

## pseudoVASP のモンテカルロ・ シミュレーション・コントローラーの開発

関西学院大学 理工学部  
情報科学科 西谷研究室  
7623 京口周平

平成 25 年 2 月 17 日

## 概 要

熱平衡状態を議論する上で、自由エネルギーを考えると物質の状態がわかる。しかし、true (第一原理計算ソフト) VASP で自由エネルギーを求めるプログラムは今まで無かった。そこで自由エネルギーの計算を行うプログラムを開発した。開発には、中井が開発した pseudo (似非) VASP を使用した。

pseudoVASP は trueVASP よりも短い計算時間で trueVASP のような振る舞いを見せる計算プログラムである。その pseudoVASP のコントローラーとして、*NVT* と *NPT* 一定の熱平衡状態を達成するモンテカルロ・シミュレーションプログラムを、自由エネルギーを求める為の先行研究として開発した。このコントローラーを元に trueVASP で計算回数をできるだけ減らすためのパラメータの探索を行った。対象としたパラメータは移動距離、一度に動かす原子数である。温度に対してこれらのパラメータの最適値を求めた。それぞれのパラメータをプロットすることで比較したところ、*NVT* 一定のモンテカルロ・シミュレーションは、温度に対してエネルギーが線形に増加する、古典的な振る舞いを正しく示した。

*NVT*-MC を用いた trueVASP での動作確認においては、pseudoVASP のように、値を採択することができた。更に、*NPT* 一定のモンテカルロ・シミュレーションプログラムを開発したが、上手く平衡化は行われなかった。*NPT*-MC の検証を行ったところ、体積変化のアルゴリズムと pseudoVASP との連携におけるバグがあることが判ってきた。

# 目次

|       |                          |    |
|-------|--------------------------|----|
| 第1章   | 緒言                       | 2  |
| 第2章   | 方法                       | 3  |
| 2.1   | 熱平衡モンテカルロの原理             | 3  |
| 2.2   | NVT 一定のモンテカルロ法           | 4  |
| 2.3   | NPT 一定のモンテカルロ法           | 5  |
| 2.3.1 | NPT-MC のアルゴリズム           | 5  |
| 2.3.2 | ビリアル状態方程式                | 6  |
| 2.4   | 第一原理計算ソフト VASP           | 11 |
| 2.5   | 似非 VASP                  | 12 |
| 第3章   | 結果                       | 14 |
| 3.1   | NVT-MC のシミュレーション結果       | 14 |
| 3.1.1 | 単原子移動 (mono) NVT-MC の結果  | 14 |
| 3.1.2 | 多原子移動 (multi) NVT-MC の結果 | 16 |
| 3.2   | NPT-MC のシミュレーション結果       | 17 |
| 第4章   | 議論                       | 21 |
| 第5章   | 総括                       | 23 |
| 第6章   | 付録                       | 25 |
| 6.1   | ソースコード                   | 25 |
| 6.1.1 | NVT-MC                   | 25 |
| 6.1.2 | NPT-MC                   | 29 |

# 第1章 緒言

近年の第一原理計算ソフトの操作性並びに計算速度の飛躍的な発展によって、現実の材料開発に活用できるエネルギー計算が可能となってきた。材料開発における欠陥エネルギーの計算はほぼ確立しており、界面、表面および添加元素の影響は信頼できる精度で計算されている。

材料の動作や製造プロセスにおいては有限温度での自由エネルギーが必要となる。第一原理計算は原理的に基底状態の計算であるため、有限温度の計算には、調和振動子近似値もとづいた phonon 計算のパッケージがいくつか開発、提供されている。しかし、半導体や相変態温度近傍での振る舞いには、非調和な影響を取り込むことが不可欠であるが、そのような計算パッケージは提供されていない。

このような計算には、有限温度での分子動力学シミュレーションが一般的に取られるが、残念ながら、自由エネルギーの絶対値を取り出すことはできない。一方、モンテカルロ・シミュレーションには、自由エネルギーを求める Frenkel 法が存在する。Frenkel 法では、基準状態として Einstein 結晶をとり、そこから連続的にポテンシャルを推移させて積分を実行し、現実系の自由エネルギーを求めるといった方法がとられる。このためには、通常の体積一定のモンテカルロ・シミュレーション (*NVT*-MC) ではなく、圧力一定のモンテカルロ・シミュレーション (*NPT*-MC) が必要となる。

本研究では、第一原理計算ソフト VASP を用いた *NPT* 一定のモンテカルロ・シミュレーション・プログラムの開発を目的とする。現実の VASP では計算時間が相当かかるため、中井が開発した pseudo (似非) VASP を用いて、モンテカルロ・シミュレーションのプログラム開発をおこなう。これをコントローラーとして、平衡状態へ達した時の原子のエネルギーを求める。今回は動作確認の為、単純な構造を持つ Al 原子のモデルを用いてシミュレーションを行う。また、モンテカルロ・シミュレーションでは、計算パラメータによって、系が熱平衡に達するまでの計算時間が大幅に変わる。そこで、最適なパラメータおよび原子移動法についても調査する。

## 第2章 方法

### 2.1 熱平衡モンテカルロの原理

モンテカルロ法は，乱数により系の粒子の微視的状态を作成していく手法である．この微視的状态の作成は，分子動力学法の状態点の軌跡に相当するものである．ある微視的状态の出現する確率は，対象としている統計集団の確率密度に従うものでなければならないが，現在，モンテカルロ・シミュレーションに際して圧倒的に広く用いられている Metropolis の方法は，あらかじめ確率密度（分配関数）を知る必要がないように工夫された方法である．実際問題，確率密度は前もってわからないのが普通である．モンテカルロ法は調べようとする体系の取り方によって，小正準モンテカルロ法，正準モンテカルロ法，大正準モンテカルロ法，その他，に分類されるが，その基本的な概念は同一である．西谷 [1] にどのようにして正準集団の平均が得られるかが，Markov 過程と遷移確率，エルゴード性，詳細釣り合いの条件を用いて説明されている．

モンテカルロ・シミュレーションの重要な変数として採択率がある．採択率は図 2.1 に示したとおり，Energy がマイナスの時は全て採択し，プラスの時には， $\exp(E/kT)$  に従って採択率が減少するように取る．このようなエネルギー変化にしたがって系を次々と精製していくことによって，正準集団に近い状態が高い確率で生成し，系が熱平衡状態になる．

採択率は温度を下げると下がり，平衡に達するまでのシミュレーション時間もながくなる．採択率を上げたいときは原子移動の最大値となる刻み幅を小さくする．それにより採択率は増加するが，熱平衡に達するまでの時間も増加する．し

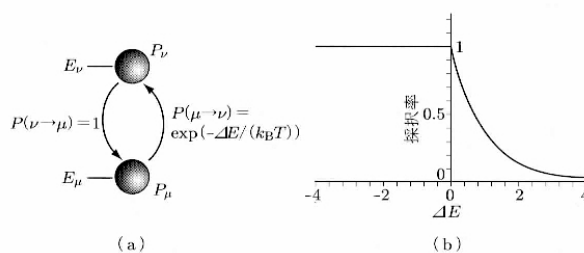


図 2.1: (a) 状態の遷移を示す模式図．2 つの状態  $\mu, \nu$  で  $E_\nu > E_\mu$  と仮定している． (b) 遷移の採択率．

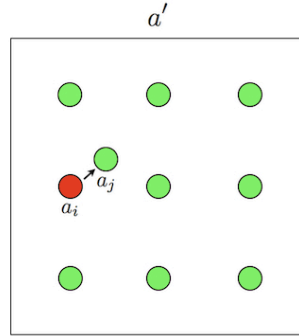


図 2.2: 新たな配置  $a'$  の模式図．ランダムに  $a_i$  座標を選択し， $a_i$  から微小に離れた  $a_j$  座標へ移動させる．この移動によって変化した配置を  $a' + \delta a'$  座標として出力する．

たがって，全体的なシミュレーション時間の短縮にはこれらのパラメータをうまく選択する必要がある．

## 2.2 NVT 一定のモンテカルロ法

$NVT$  一定のモンテカルロ法のアルゴリズムにより，原子の微小移動での並び替えで熱平衡状態のシミュレーションを行う．この  $NVT$ -MC の動作確認の為に，一度に動かす原子の数を，単原子とする mono バージョンと多原子とする multi バージョンを順に開発する． $N$  は粒子数， $V$  は体積， $T$  は温度を表す．熱平衡モンテカルロ法のアルゴリズムは，隣接した位置関係にある原子を並び替えるアルゴリズムである為，微小な距離での原子の並び替えをすることはできない．そこで，熱平衡モンテカルロ法の原子の並び替えに関する箇所を，図 2.2 に示す様に，微小距離で並び替える make  $a_j$  というアルゴリズムに書き換え， $NVT$ -MC のアルゴリズムを作成する．

以下に make  $a_j$  のアルゴリズムを示す．

1. 配置  $a'$  を仮定し， $E(a')$  を求める．
2.  $a'$  から微小に離れた配置  $a' + \delta a'$  を求める（ $a_j$  を作る．）
3. (a)  $\Delta E < 0$  ならステップ 2 を採用．  
 (b)  $\Delta E > 0$  ならステップ 2 を  $\exp(-\Delta E/T) < \text{乱数の確率}$  で採用．
4. ステップ 2 以下を繰り返す．

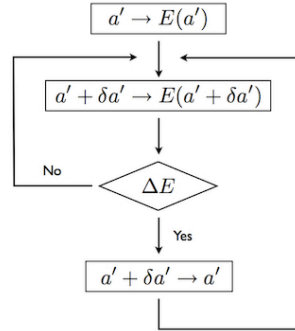


図 2.3: make  $a_j$  のフローチャート .

以上の流れを 図 2.3 に示した .

$NVT$ -MC の動作確認ができれば , 体積を変化させる  $NPT$  一定のモンテカルロ法のアルゴリズムの開発を行っていく .

## 2.3 $NPT$ 一定のモンテカルロ法

最初に ,  $NPT$  一定のモンテカルロ・シミュレーションのアルゴリズムについて記述する . そしてビリアル状態方程式で , 圧力  $P$  を求める為に必要な , 運動量輸送  $P_K$  と粒子間力  $P_U$  の導出を説明する .

### 2.3.1 $NPT$ -MC のアルゴリズム

$NPT$  一定のモンテカルロ法は , 圧力  $P$  を一定にして ,  $NVT$ -MC で一定とした体積  $V$  を変化させて熱平衡状態のシミュレーションを行うアルゴリズムである .  $NPT$ -MC を行う為のアルゴリズムを下記に示す .

1. 初期体積と初期状態を与え , その状態を状態  $i$  とする
2. ポテンシャル・エネルギー  $U_i$  を計算する
3.  $0 \sim 1$  の範囲に分布する一様乱数列から乱数  $R_1$  を取り出し , 体積を  $V_i^*$  から  $V_j^*$  に変化させる . すなわち ,  $V_j^* = V_i^* + (2R_1 - 1)\delta V_{max}^*$
4. 体積  $V_j^*$  の状態に対し , 選び出した一つの粒子を乱数を用いて移動させることにより , 状態  $j$  をつくる

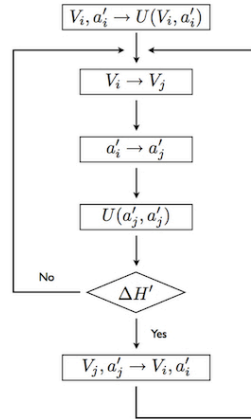


図 2.4:  $NPT$  の make  $a_j$  のフローチャート

5. ポテンシャル・エネルギー  $U_j$  を計算する
6.  $\Delta H' = \{PV_0(V_j^* - V_i^*) + U_j - U_i - NkT \ln(V_j^*/V_i^*)\}$  を計算する
7. もし,  $\Delta H' \leq 0$  ならば, 状態  $j$  をマルコフ連鎖を構成する一つの状態と見なして, ステップ 3 から繰り返す
8. もし,  $\Delta H' > 0$  ならば, 上記の乱数列からさらに乱数  $R_2$  を取り出し,
  - (a)  $\exp(-\Delta H'/kT) > R_2$  のとき, 状態  $j$  をマルコフ連鎖を構成する一つの状態とみなして, ステップ 3 から繰り返す
  - (b)  $\exp(-\Delta H'/kT) \leq R_2$  のとき, 体積を変化させる前の状態, すなわち, 体積が  $V_i^*$  で粒子の位置が  $\underline{s}_i$  の状態を, マルコフ連鎖を構成する一つの状態と見なして, ステップ 3 から繰り返す

以上の流れを 図 2.4 に示した.

上述のアルゴリズムの圧力  $P$  はビリアル状態方程式によって導くことができる.

### 2.3.2 ビリアル状態方程式

ビリアル状態方程式 (virial equation of state) [2] は通常, 粒子間および粒子との容器壁との相互作用を考慮することで導出できるが, 分子シミュレーションの場合, 周期的境界条件を用いるので, 容器壁は存在しない. したがって, ここでは通常とは異なる別の方法でこの方程式を導出する.

いま, 粒子数が  $N$ , 体積が  $V$  (一辺が  $L$  の立方体, つまり  $L^3 = V$ ) なる熱力学的平衡状態にある系を考える.



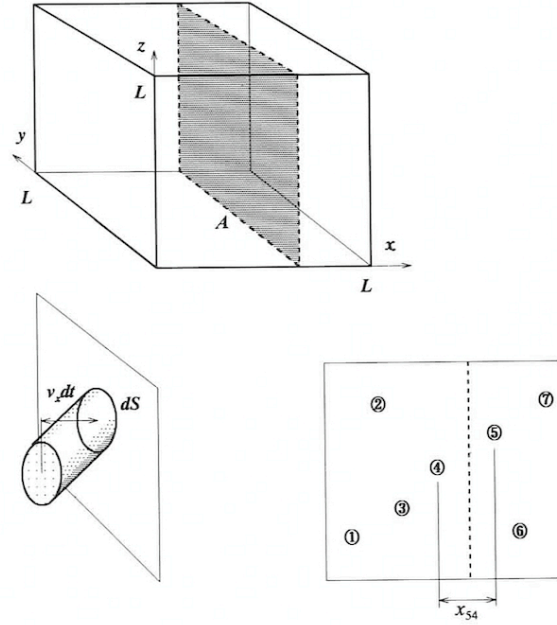


図 2.5: ビリアル状態方程式導出の為の検査面．

圧力は検査面に垂直に作用する単位面積あたりの力として定義される．図 2.5 の場合，検査面の右側部を固体のように考えて，検査面左側の粒子によって引き起こされる，固体の左側面を圧縮する単位面積あたりの力，と考えればわかりやすい．したがって，図 2.5 の場合，圧力は  $x$  方向の向きを正の方向として通常は定義される．圧力は運動量輸送および粒子間力に起因して発生する．まず最初に，運動量輸送に圧力  $P_K$  を導出し，それから粒子間力による圧力  $P_U$  を導出する．

検査面の左側にいる粒子  $i$  に着目し，この粒子が微小時間後に検査面を横切り，右側の領域の粒子と衝突して，また検査面を横切って左側の領域に戻ったとする．この場合の運動量の変化を微小時間で除せば，粒子  $i$  が検査面に作用した力がわかる．しかしながら，系が平衡状態にあるならば，わざわざ個々の粒子の運動を問題にする必要はなく，統計的に処理することができる．すなわち，ある微小時間に検査面に流入する粒子の運動量と流出する運動量を別々に評価することで，検査面に作用する力が計算できる．

図 2.5 に示すように，粒子が速度  $\mathbf{v} = (v_x, v_y, v_z)$  を有して時間  $dt$  の間に微小検査面  $dS$  を横切るとすると，粒子がそのような速度を有する確率は，マクスウェル分布の速度分布  $f$

$$f(v_i) = \left( \frac{m}{2\pi kT} \right)^{3/2} \exp \left\{ -\frac{m}{2kT} (v_{ix}^2 + v_{iy}^2 + v_{iz}^2) \right\} \quad (2.1)$$

を用いて  $f(\mathbf{v})d\mathbf{v}$  で表すことができ，また，そのような速度を有して微小検査面

を横切る粒子数は，平均的に考えれば，図 2.5 の傾斜円柱の体積  $v_x dtdS$  と粒子一個の占める体積  $V/N$  の比で表せる．したがって，速度  $\mathbf{v}$  を有して単位面積単位時間あたりに（左側から）検査面に流入する運動量の  $x$  方向成分  $p_{in}^x$  は，

$$p_{in}^x = \left\{ \frac{v_x dtdS}{V/N} \cdot f(\mathbf{v}) d\mathbf{v} \cdot m v_x \right\} / (dtdS) = m \frac{N}{V} v_x^2 f(\mathbf{v}) d\mathbf{v} \quad (2.2)$$

と表せる．この式を速度に関して積分すれば，

$$\begin{aligned} \sigma_{in}^x &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_0^{\infty} p_{in}^x dv_x dv_y dv_z \\ &= m \frac{N}{V} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_0^{\infty} v_x^2 f(\mathbf{v}) d\mathbf{v} \end{aligned} \quad (2.3)$$

となり， $\sigma_{in}^x$  は単位面積単位時間あたりに流入する  $x$  方向の運動量である．同様にして，検査面から左側へ流出する  $x$  方向の運動量  $\sigma_{out}^x$  は，

$$\sigma_{out}^x = m \frac{N}{V} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^0 v_x^2 f(\mathbf{v}) d\mathbf{v} \quad (2.4)$$

したがって， $\sigma_{in}^x$  と  $\sigma_{out}^x$  のどちらも正になるように取ってあることを注意すれば，圧力  $P_K^x$  は次のように得られる．

$$\begin{aligned} P_K^x &= \sigma_{in}^x + \sigma_{out}^x \\ &= m \frac{N}{V} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} v_x^2 f(\mathbf{v}) d\mathbf{v} \\ &= m \frac{N}{V} \langle v_x^2 \rangle \end{aligned} \quad (2.5)$$

通常，圧力は  $P_K^x$ ， $P_K^y$ ， $P_K^z$  の算術平均として定義されるので，この平均値を  $P_K$  と書けば，

$$P_K = \frac{1}{3} (P_K^x + P_K^y + P_K^z)$$

$P_K^x$  は  $P_K$  の  $x$  方向のみの圧力なので， $PV = 2E/1$  より，

$$\begin{aligned} P_K &= \frac{1}{3} \left( \frac{2E_x}{V} + \frac{2E_y}{V} + \frac{2E_z}{V} \right) \\ &= \frac{2}{3V} (E_x + E_y + E_z) \end{aligned}$$

ここで  $E_x, E_y, E_z$  は単位粒子数当りの運動エネルギーの  $E = mv^2/2$  より ,

$$P_K = \frac{2}{3V} \left\langle \frac{m(v_x^2 + v_y^2 + v_z^2)}{2} \right\rangle$$

$E$  を全粒子数当りの運動エネルギーに対応させるために  $N$  を掛けると ,

$$P_K = \frac{2N}{3V} \left\langle \frac{m(v_x^2 + v_y^2 + v_z^2)}{2} \right\rangle$$

ここで  $m(v_x^2 + v_y^2 + v_z^2)/2 = E$  より ,

$$P_K = \frac{2N}{3V} E$$

$2E/3 = kT$  なので ,

$$P_K = \frac{N}{V} kT \quad (2.6)$$

ここに最後の式は , 式 (2.1) を用いて直接平均することで得られた .

次に , 粒子間力に起因する圧力  $P_U$  を求める . いま , 図 2.5 のように検査面が粒子 4 と 5 の間にあるとする . ただし , 粒子はその位置の  $x$  座標が小さいほど若い番号が着くように番号づけしている . 2 体力近似を用いれば , 検査面をまたいで相互作用する粒子間の力のみが圧力に寄与する . したがって , 図 2.5 の場合 , 粒子 1 と 4 の間で働く力は圧力に寄与しない . 粒子  $i$  に作用する粒子  $j$  の力の  $x$  方向成分を  $f_{ij}^x$  とすれば , 図 2.5 の場合の検査面に作用する力の  $x$  方向成分は ,

$$- \sum_{i=1}^4 \sum_{j=5}^N f_{ij}^x \quad (2.7)$$

となる . この力は  $x_4 \sim x_5$  の座標間で一定である . 検査面を任意の位置に設定し , それらの各位置に対する値を算術平均して , 検査面に作用する単位面積あたりの力の  $x$  方向成分  $\tau^x$  を算出すれば , 次のようになる .

$$\begin{aligned} \tau^x = - \left\{ \left( \frac{x_{21}}{L} \sum_{j=2}^N f_{1j}^x \right) + \left( \frac{x_{32}}{L} \sum_{i=1}^2 \sum_{j=3}^N f_{ij}^x \right) + \right. \\ \left. \cdots + \left( \frac{x_{N,N-1}}{L} \sum_{i=1}^{N-1} f_{i,N}^x \right) \right\} / L^2 \end{aligned} \quad (2.8)$$

ここに ,  $x_{ij} = x_i - x_j$  である . ゆえに , 右辺を整理すれば ,

$$\begin{aligned}
\tau^x V &= - [x_{21}(f_{12}^x + f_{13}^x + \cdots + f_{1N}^x) \\
&\quad + x_{32}(f_{13}^x + f_{14}^x + \cdots + f_{1N}^x + f_{23}^x + f_{24}^x + \cdots + f_{2N}^x) + \cdots \\
&\quad + x_{N,N-1}(f_{1N}^x + f_{2N}^x + \cdots + f_{N-1,N}^x)] \\
&= \{x_{12}f_{12}^x + (x_{12} + x_{23})f_{13}^x + \cdots \\
&\quad + (x_{12} + x_{23} + \cdots + x_{N-1,N})f_{1N}^x\} + \{x_{23}f_{23}^x + (x_{23} + x_{34})f_{24}^x \\
&\quad + \cdots + (x_{23} + x_{34} + \cdots + x_{N-1,N})f_{2N}^x\} + \cdots \\
&= (x_{12}f_{12}^x + x_{13}f_{13}^x + \cdots + x_{1N}f_{1N}^x) \\
&\quad + (x_{23}f_{23}^x + x_{24}f_{24}^x + \cdots + x_{2N}f_{2N}^x) + \cdots \\
&= \sum_{i=1}^{N-1} \sum_{j=i+1}^N x_{ij} f_{ij}^x = \sum_{i=1}^N \sum_{\substack{j=1 \\ (i < j)}}^N x_{ij} f_{ij}^x
\end{aligned} \tag{2.9}$$

よって,  $\tau^x$ ,  $\tau^y$ ,  $\tau^z$  の集団平均の算術平均として  $P^U$  を求めれば,

$$\begin{aligned}
P_U &= \frac{1}{3} \langle \tau^x + \tau^y + \tau^z \rangle \\
&= \frac{1}{3V} \left\langle \sum_i \sum_{\substack{j \\ (i < j)}} (x_{ij} f_{ij}^x + y_{ij} f_{ij}^y + z_{ij} f_{ij}^z) \right\rangle \\
&= \frac{1}{3V} \left\langle \sum_i \sum_{\substack{j \\ (i < j)}} (\mathbf{r}_{ij} \cdot \mathbf{f}_{ij}) \right\rangle
\end{aligned} \tag{2.10}$$

以上より, 圧力  $P$  は式 (2.6) と式 (2.10) の和として次のように表される.

$$P = \frac{N}{V} kT + \frac{1}{3V} \left\langle \sum_i \sum_{\substack{j \\ (i < j)}} \mathbf{r}_{ij} \cdot \mathbf{f}_{ij} \right\rangle \tag{2.11}$$

この式をビリアル状態方程式という.

ここに,  $\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$ ,  $\mathbf{f}_{ij}$  は粒子  $j$  が粒子  $i$  に及ぼす力であり, 右辺第 1 項は粒子の運動による寄与, 第 2 項は粒子間力に起因する項である.

なお,

$$W = \frac{1}{3} \sum_i \sum_{\substack{j \\ (i < j)}} \mathbf{r}_{ij} \cdot \mathbf{f}_{ij} \tag{2.12}$$

とおけば，この  $W$  を内部ビリアル (internal virial) という．この式は，

$$\begin{aligned}
\sum_i \mathbf{r}_i \cdot \mathbf{f}_i &= \sum_i \sum_{\substack{j \\ (i \neq j)}} \mathbf{r}_i \cdot \mathbf{f}_{ij} \\
&= \sum_i \sum_{\substack{j \\ (i \neq j)}} \left( \frac{1}{2}(\mathbf{r}_i \cdot \mathbf{f}_{ij}) + \frac{1}{2}(\mathbf{r}_i \cdot \mathbf{f}_{ij}) \right) = \sum_i \sum_{\substack{j \\ (i \neq j)}} \left( \frac{1}{2}(\mathbf{r}_i \cdot \mathbf{f}_{ij}) + \frac{1}{2}(\mathbf{r}_j \cdot \mathbf{f}_{ji}) \right) \\
&= \frac{1}{2} \sum_i \sum_{\substack{j \\ (i \neq j)}} (\mathbf{r}_i \cdot \mathbf{f}_{ij} + \mathbf{r}_j \cdot \mathbf{f}_{ji}) = \frac{1}{2} \sum_i \sum_{\substack{j \\ (i \neq j)}} (\mathbf{r}_i \cdot \mathbf{f}_{ij} - \mathbf{r}_j \cdot \mathbf{f}_{ij}) \\
&= \frac{1}{2} \sum_i \sum_{\substack{j \\ (i \neq j)}} (\mathbf{f}_{ij}(\mathbf{r}_i - \mathbf{r}_j)) = \frac{1}{2} \sum_i \sum_{\substack{j \\ (i \neq j)}} \mathbf{r}_{ij} \cdot \mathbf{f}_{ij}
\end{aligned} \tag{2.13}$$

ここで， $(i \neq j)$  の関係にある時， $i = (1, 2, 3)$  と  $j = (1, 2, 3)$  は以下の 6 通りである．

$$i, j = \{(1, 2), (1, 3), (2, 1), (2, 3), (3, 1), (3, 2)\}$$

また， $(i < j)$  の関係にある時， $i = (1, 2, 3)$  と  $j = (1, 2, 3)$  は以下の 3 通りである．

$$i, j = \{(1, 2), (1, 3), (2, 3)\}$$

なので， $(i \neq j) = (i < j) \cdot 2$  である．よって，

$$\begin{aligned}
&= \frac{1}{2} \sum_i \sum_{\substack{j \\ (i < j)}} \mathbf{r}_{ij} \cdot \mathbf{f}_{ij} \cdot 2 \\
&= \sum_i \sum_{\substack{j \\ (i < j)}} \mathbf{r}_{ij} \cdot \mathbf{f}_{ij}
\end{aligned} \tag{2.14}$$

を考慮すると，次式のようにも書ける．

$$W = \frac{1}{3} \sum_i \mathbf{r}_i \cdot \mathbf{f}_i \tag{2.15}$$

## 2.4 第一原理計算ソフト VASP

本研究では pseudoVASP のコントローラーを作成するので，その動作確認の為に trueVASP を用いて pseudoVASP との計算結果の比較を行う．trueVASP は固

体物質の電子構造からエネルギーを求める第一原理計算を行うソフトウェアである．trueVASP は平面波基底，擬ポテンシャル法を用いることで，高精度な計算を比較的高速に進めている．

西谷研究室では，この trueVASP 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，モンテカルロシミュレーションなどこれらのコントローラでは，trueVASP 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 trueVASP に入力として与え，再び計算を行う．以上を，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．

## 2.5 似非 VASP

本研究では，中井の開発した pseudo (似非) VASP [3] 上で計算シミュレーションを行う．pseudoVASP は VASP を用いて第一原理計算を行うよりも，短い計算時間でエネルギーやフォースを求めることができる．この pseudoVASP は以下の流れで計算を行う．

1. POSCAR から原子座標を読み取る．
2. 読み取った原子座標から近接原子を選択して，ネイバーリストの配列に格納．
3. Lennard-Jones ポテンシャルでエネルギーを計算．
4. 求めたエネルギーを距離で微分して，force を計算  
(ただし今回の計算で force は使わない．)
5. 結果を OUTCAR 形式で出力．

以上の流れを 図 2.6 に示した．

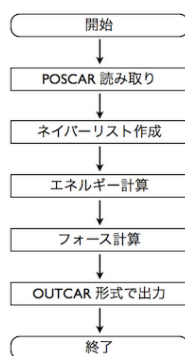


図 2.6: pseudoVASP における計算のフローチャート .

## 第3章 結果

### 3.1 NVT-MC のシミュレーション結果

最初に、*NVT* 一定のモンテカルロ・シミュレーションの結果について記述する。*NVT* 一定のモンテカルロ・シミュレーションプログラムとしては、二つのバージョンを作成した。原子移動の試行に際して、原子一個のみが移動する単原子 (mono) *NVT*-MC と、複数の原子を同時に移動可能な多原子 (multi) *NVT*-MC である。

#### 3.1.1 単原子移動 (mono) *NVT*-MC の結果

*NVT*-MC でいくつかのパラメータを変えて熱平衡にいたるまでの様子を調べた。図 3.1 は横軸に試行回数、縦軸にエネルギーの変化を取りプロットした様子である。試行回数は 10000 回で原子を 1 個移動させている。この際に、目安となる物性値は系全体のエネルギーである。熱平衡状態においては、エネルギーは揺れ動きながらもほぼ一定の値を取る。モンテカルロ・シミュレーションにおいて、シミュレーションがうまく進行しているかどうかは平衡状態、つまりエネルギー一定の状態に達しているかどうかで判断することができる。また、エネルギーは温度に対して、一定の勾配で上昇していく。これが古典的極限と呼ばれる状態である。

ここで、(a) 温度  $0.1\text{eV}$ 、原子間距離比 0.1 の時のエネルギー変化を例に平衡化したときのエネルギーを求める。図 3.2 は温度  $0.1\text{eV}$ 、原子間距離比 0.1 における Cut Avr の始点と終点の位置を示している。図 3.2 を見ると、平衡化は約 700 回目から始まり、約 4900 回目あたりで収束したように見える。そこで e-change.txt の末尾から 4200 番目までのエネルギーを乗算し、4200 で除算すれば平衡化した部分のエネルギーの平均値である、 $-27.69$  が求まった。図 3.3 の通り、この方法で他の原子間距離の値についても、同じように平衡化した時のエネルギーの平均値を求めた。

単原子を動かす mono バージョンが上手くいったので、多原子の multi バージョンでシミュレーションを行った。



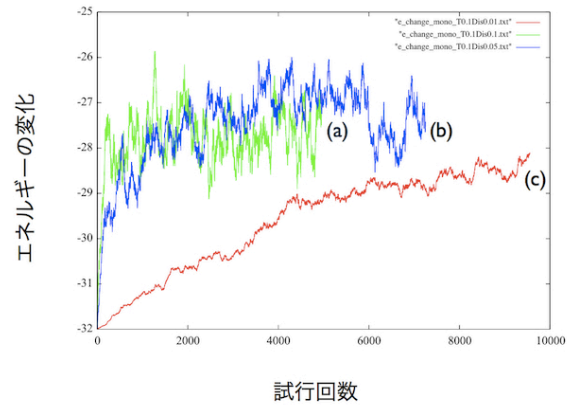


図 3.1: (a) 温度  $0.1eV$  , 原子間距離比  $0.1$  におけるエネルギー変化 . (b) 温度  $0.1eV$  , 原子間距離比  $0.05$  におけるエネルギー変化 . (c) 温度  $0.1eV$  , 原子間距離比  $0.01$  におけるエネルギー変化 .

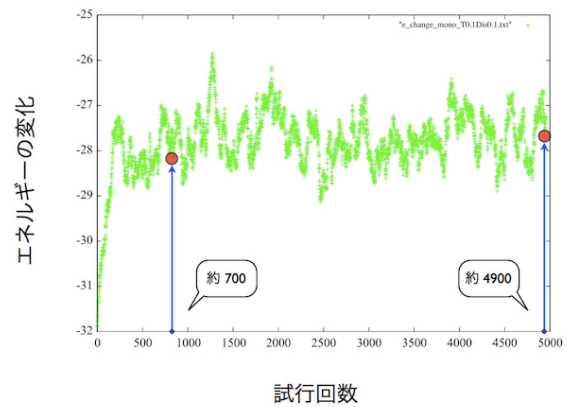


図 3.2: 温度  $0.1eV$  , 原子間距離比  $0.1$  におけるエネルギーのカット位置の決定方法 .

試行回数 10000 回でのエネルギーの平均値 (mono)

| 温度 $T$ [eV] | 距離比 Dis | Cut Avr | getAverage | 採択率  |
|-------------|---------|---------|------------|------|
| 0.1         | 0.1     | 4200    | -27.69     | 0.50 |
|             | 0.05    | 5400    | -27.30     | 0.73 |
|             | 0.01    | 5500    | -28.81     | 0.96 |

| 温度 $T$ [eV] | Cut Avr   |         |
|-------------|-----------|---------|
|             | Cut Start | Cut End |
| 0.1         | 700       | 4900    |
|             | 2000      | 7400    |
|             | 4000      | 9500    |

図 3.3: 温度  $0.1\text{eV}$  における平衡化した時のエネルギーの平均値 .

試行回数 10000 回でのエネルギーの平均値 (multi)

| 温度 $T$ [eV] | 距離比 Dis | Cut Avr | getAverage | 採択率  |
|-------------|---------|---------|------------|------|
| 0.1         | 0.1     | 1200    | -27.60     | 0.15 |
|             | 0.05    | 3800    | -27.84     | 0.55 |
|             | 0.01    | 6000    | -27.74     | 0.88 |
| 0.05        | 0.1     | 450     | -29.63     | 0.07 |
|             | 0.05    | 2500    | -29.45     | 0.34 |
|             | 0.01    | 5700    | -29.76     | 0.85 |
| 0.01        | 0.1     | -       | -          | 0.00 |
|             | 0.05    | 300     | -31.50     | 0.06 |
|             | 0.01    | 6000    | -31.56     | 0.69 |

図 3.4: 4 原子の  $NVT$  一定のモンテカルロシミュレーション .

### 3.1.2 多原子移動 (multi) $NVT$ -MC の結果

$NVT$ -MC で、一度に動かす原子の数を 4 つに増やして、試行回数 10000 回で熱平衡に至るまでの様子を 図 3.4 に示した .

温度  $0.01\text{eV}$  距離比 0.1 の時のシミュレーション結果では値が算出されていない . これは原子の移動が行われず採択されなかったことが原因である . 安定した温度域で一度に多く原子を動かした為、0.1 もの原子間の距離を移動できなかった . これを除けば、他は採択されている . 代表例として、図 3.5 に 4 原子と単原子の熱平衡状態に至るまでの様子を示した . この 2 つを比較すると、4 原子の方は約 3 分の 1 の試行回数で熱平衡状態へ収束している .

それぞれの計算結果の有意性を確認する為に、図 3.6 に 図 3.4 の 温度を 0.01 , 0.05 ,  $0.1\text{eV}$  と変化させた時の、平衡化した時のエネルギーの平均値をプロットした様子を示す . 距離毎に平衡状態に落ち着いているか否か、という観点でこのグラフを見ると、ほぼ直線になっている . よって、 $NVT$ -MC を実行し得られた結果

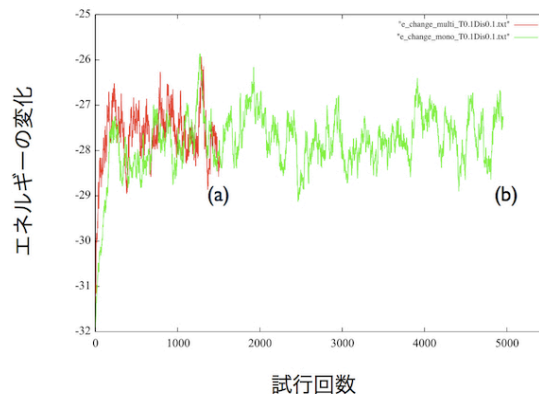


図 3.5: 温度  $0.1\text{eV}$  距離比  $0.1$  でのシミュレーションの様子．(a) 4 原子．(b) 単原子．

から，古典的な振る舞いの特徴を見ることができるので， $NVT$ -MC がうまく進んでいることを示唆している．

しかし，温度  $0.1\text{eV}$  のプロット点のズレは原理的には考えられないものである． $0.1\text{eV}$  は実際の Al ならば融けている温度であるが，融けるという挙動は pseudoVASP にはない．Al が融けているとすれば，それぞれのプロット点は綺麗に並ばない．しかし今回のシミュレーションの結果によると， $0.01\text{eV}$ ， $0.05\text{eV}$  の時に直線上に乗っている．ところが  $0.1\text{eV}$  の時は平衡に達していない．そこで原子間距離比  $0.05$  について，温度の幅を  $0.01$  刻みで  $0.1\text{eV}$  まで増やして，更にシミュレーションを行った．結果をプロットしたものを図 3.7 に，その時の各温度変化を図 3.8 に示す．

一定の直線で外挿できるならば，シミュレーションが十分に平衡に達していることを示唆している．しかし，もし，値が下側に出るならば，それは，もう少し長い時間をとって十分に平衡に達するまで待つ必要があることを示唆している．一般的に，温度が上がると早く平衡に達するので，シミュレーション回数が少ないとは判断できない．しかし，この温度はシミュレーション温度としては高いため，普通の固体では融けている場合もある．したがってより低い温度で，傾向がどのように変化するかを詳細に追いかける必要がある．

## 3.2 NPT-MC のシミュレーション結果

原子移動と体積移動を  $1, 1$  でシミュレーションを行なった．つまり一回 POSCAR を動かして原子移動させる度に，体積変化させている．これが良いのかは分からないので，計算試行回数を  $1000$  回として，平衡状態になるかを確認した．その様子を図 3.9 に示した．

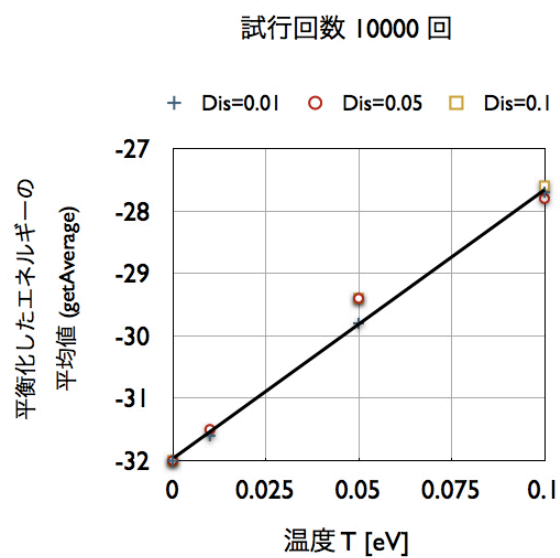


図 3.6: エネルギーの温度依存性 .

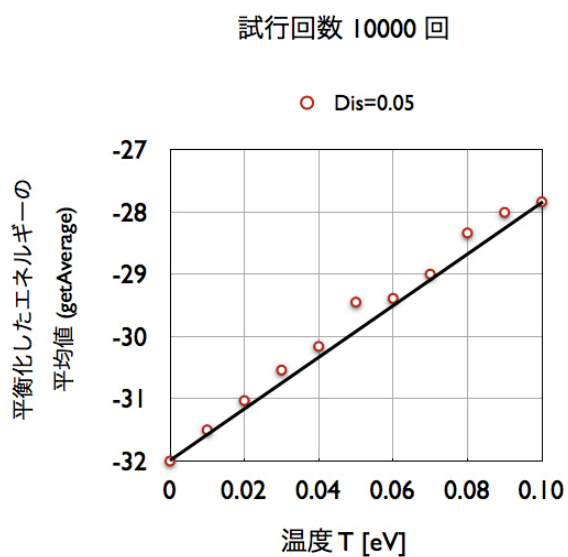


図 3.7: 原子間距離比 0.05 のエネルギーの温度依存性 .

試行回数 10000 回でのエネルギーの平均値 (multi)

| 距離比 Dis | 温度 $T$ [eV] | Cut Avr | getAverage | 採択率  |
|---------|-------------|---------|------------|------|
| 0.05    | 0.1         | 3800    | -27.84     | 0.55 |
|         | 0.09        | 3300    | -28.01     | 0.45 |
|         | 0.08        | 3000    | -28.34     | 0.43 |
|         | 0.07        | 2850    | -29.00     | 0.41 |
|         | 0.06        | 3300    | -29.39     | 0.39 |
|         | 0.05        | 2500    | -29.45     | 0.34 |
|         | 0.04        | 2500    | -30.16     | 0.31 |
|         | 0.03        | 1800    | -30.54     | 0.24 |
|         | 0.02        | 1500    | -31.03     | 0.17 |
|         | 0.01        | 300     | -31.50     | 0.06 |

図 3.8: 原子間距離比 0.05 の各温度変化 .

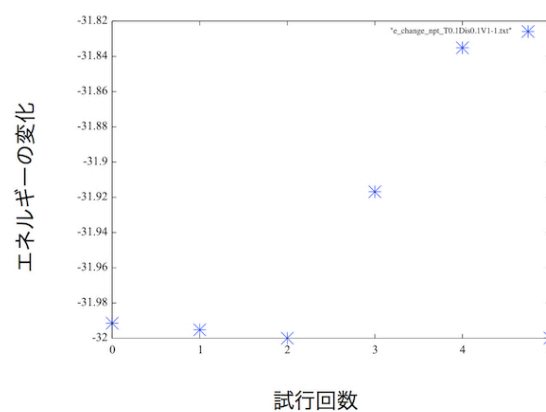


図 3.9: 温度  $0.1\text{eV}$  原子間距離比 0.1 原子移動, 体積移動共に 1, 1 で平衡化した時のエネルギー変化 .

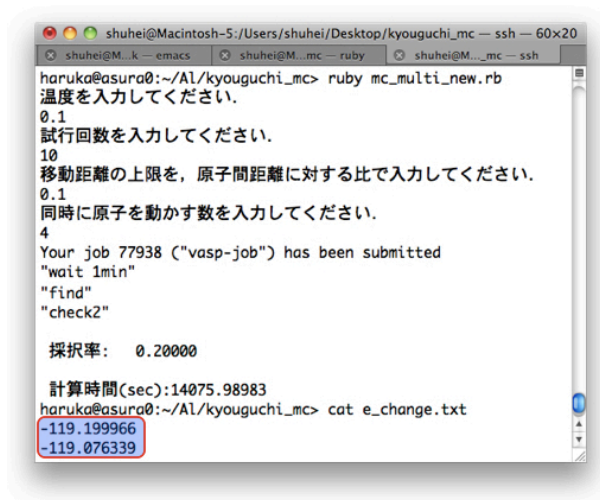
この結果によると平衡化は上手く行われていない．原因としては， $\text{\AA}$  に対して規格化を行なっていたことにより，体積が大きくなりすぎたことにあると考えられる．よって  $V$  を掛けない方が良いということが分かった．このアルゴリズムでは体積を  $V$  としているが， $V$  は規格化している為， $PV$  になる．そして  $P$  は本来ならば  $eV/\text{\AA}^3$  になっている為， $V$  を掛ける必要があった．( $V = \text{\AA}^3$ ) これで  $eV$  になった．ところが， $P$  自身が既に  $r \cdot f$  のときの  $r$  が実は  $\text{\AA}$  ではなく， $\text{\AA}^3$  になってしまっている． $f$  についても  $eV/\text{\AA}^2$  であるが，これも規格化してしまっている．よって両方共消えてしまった．なので VASP で計算するときに，相対座標を絶対座標に変換する関数がプログラムに必要だと考えられる．今は体積全体を動かしている．例えば 0.9 をセットしたとすれば，それは体積を 0.9 倍縮小していることになる．そうすると，現在の psuedoVASP では周期的境界条件が考慮されていない為，シミュレーションが破綻してしまうことが懸念された．

## 第4章 議論

以上の結果から pseudoVASP による 4 原子の NVT-MC は動作確認ができたので、trueVASP についても、コントローラーの動作確認を行った。図 4.1 にその様子を示す。trueVASP での計算は、温度  $0.1\text{eV}$ 、距離比 0.1、計算試行回数 10 回でシミュレーションを行った。pseudoVASP も同じ条件で計算を行ったところ、今回は共に 採択率 0.2 で採択された。採択率は乱数により変動する為、同じ確率で採択されていることに意味はない、pseudoVASP と同様に trueVASP でも値を算出できた事が重要である。結果の数値が異なる理由は、pseudoVASP ではエネルギーを規格化しているが、trueVASP ではエネルギーを規格化せずに求めている為である。よって、pseudoVASP のコントローラーは trueVASP においても正常に動作しているといえる。

本来は、より多くの試行回数シミュレーションによって平衡状態に達しているかを確認する必要があるが、試しにおこなった計算としては、実行が確認できたので十分であろう。しかし、計算時間には問題がある。trueVASP を用いた計算では、pseudoVASP に比べて 14000 倍の計算時間がかかり、10 回の計算で約 4 時間かかっている。したがって、pseudoVASP のシミュレーションから予測される試行回数を 1000 回とすると、400 時間すなわち 17 日あるいは 2 週間程度が必要となる。

今後 trueVASP を用いた精度の高い自由エネルギーを求めるためには、十分な覚悟が必要である。また、モンテカルロ・シミュレーションの実行パラメータはより精度良く絞り込む必要がある。また、trueVASP をより高速に実行するため、電子エネルギーのセルフコンシステントループの収束を加速する CONTCAR を用いる手法など、より多くの工夫をしなければ、現実的な系でのシミュレーションは困難と考えられる。

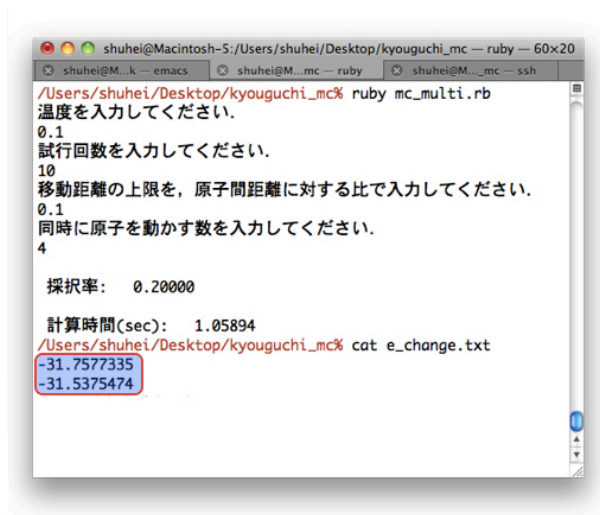


```
shuheideMacintosh-5:/Users/shuheideDesktop/kyouguchi_mc — ssh — 60x20
shuheideM...k — emacs  shuheideM...mc — ruby  shuheideM...mc — ssh
haruka@asura0:~/Al/kyouguchi_mc> ruby mc_multi_new.rb
温度を入力してください。
0.1
試行回数を入力してください。
10
移動距離の上限を、原子間距離に対する比で入力してください。
0.1
同時に原子を動かす数を入力してください。
4
Your job 77938 ("vasp-job") has been submitted
"wait 1min"
"find"
"check2"

採択率: 0.20000

計算時間(sec):14075.98983
haruka@asura0:~/Al/kyouguchi_mc> cat e_change.txt
-119.199966
-119.076339
```

(a)



```
shuheideMacintosh-5:/Users/shuheideDesktop/kyouguchi_mc — ruby — 60x20
shuheideM...k — emacs  shuheideM...mc — ruby  shuheideM...mc — ssh
/Users/shuheideDesktop/kyouguchi_mc% ruby mc_multi.rb
温度を入力してください。
0.1
試行回数を入力してください。
10
移動距離の上限を、原子間距離に対する比で入力してください。
0.1
同時に原子を動かす数を入力してください。
4

採択率: 0.20000

計算時間(sec): 1.05894
/Users/shuheideDesktop/kyouguchi_mc% cat e_change.txt
-31.7577335
-31.5375474
```

(b)

図 4.1: (a) trueVASP での実行結果. (b) pseudoVASP での実行結果 .



## 第5章 総括

本研究では pseudoVASP を用いて、微小な原子移動で一定の熱平衡状態を達成するモンテカルロ・シミュレーションプログラムを開発した。対象としたパラメータは移動距離、一度に動かす原子数である。温度に対してこれらのパラメータの最適値を求めた。

pseudoVASP による *NVT*-MC シミュレーションは、単原子を動かす *mono* バージョンを作成し動作確認した後に、4 原子を動かす *multi* バージョンを作成し、熱平衡のシミュレーションを行った。温度変化ごとに結果をプロットすることで、それぞれのパラメータを一定の直線に外挿できたので、シミュレーションが十分に平衡状態を導いたことを結果付けた。そして trueVASP による *NVT*-MC の動作確認として、同様にパラメータの探索を行ったところ、値が採択されているので、*NVT*-MC の動作には問題がないことがわかった。

しかし、pseudoVASP による *NPT*-MC では、体積変化のアルゴリズムと pseudoVASP との連携の際に発生するバグにより、平衡状態を導くことができなかった。よって今後の課題としてはこのバグを解決することにあると考える。

# 謝辞

本研究を遂行するにあたって，終始多大なる有益なご指導，ご鞭撻，及び心温かい助言を賜りました，関西学院大学理工学部 情報科学科 西谷滋人教授に，深い感謝と共に，心よりの御礼を申し上げます．そして，本研究の進行に伴い，多くのご協力を頂きました，西谷研究室の同輩，先輩方に対して，この場を借りて心から感謝致します．本当にありがとうございました．

## 第6章 付録

### 6.1 ソースコード

本研究で作成した *NVT*-MC と *NPT*-MC のソースコードを示す．

#### 6.1.1 NVT-MC

*NVT*-MC のシミュレーションを行う為のプログラム．同時に動かす原子数を入力させることで，単原子と多原子の切り替えを行う．

```
\begin{enumerate}
\item -*- coding: utf-8 -*-
\end{enumerate}

include Math
require "pp"

\begin{enumerate}
\item order
\end{enumerate}

puts "温度を入力してください．"
T=gets.chomp.to_f
puts "試行回数を入力してください．"
n_max=gets.chomp.to_i

puts "移動距離の上限を，原子間距離に対する比で入力してください．"
$d_max=gets.chomp.to_f
puts "同時に原子を動かす数を入力してください．"
$n_multi=gets.chomp.to_i

start = Time.now #計測開始

n_arng = []
```

```

def get_volume
  vec_a = []
  vec_b = []
  vec_c = []

  vec_a=$poscar_text[2].split(nil) #格子ベクトル
  vec_b=$poscar_text[3].split(nil)
  vec_c=$poscar_text[4].split(nil)
  a_x = vec_a[0].to_f
  a_y = vec_a[1].to_f
  a_z = vec_a[2].to_f
  b_x = vec_b[0].to_f
  b_y = vec_b[1].to_f
  b_z = vec_b[2].to_f
  c_x = vec_c[0].to_f
  c_y = vec_c[1].to_f
  c_z = vec_c[2].to_f
  volume=(a_x*b_y*c_z)+(a_y*b_z*c_x)+(a_z*b_x*c_y)
            -(a_x*b_z*c_y)-(a_y*b_x*c_z)-(a_z*b_y*c_x)
  return volume
end

def get_distance #体積からおよその原子半径を求める
  r_3=(get_volume())/poscar_text[5].to_f/(4*PI/3)
  r=r_3**(1.0/3.0)
  return r
end

def set_newPOSCAR()
  system("cp Al_POSCAR POSCAR")
  text = []
  file =File.open("POSCAR","r")
  file.each do |f|
    text << f
  end
  $poscar_text = text
  $init_vals = []
  $init_vals << get_distance/(get_volume**(1.0/3.0))*$d_max
end

```

```

end

def read_POSCAR()
  text = []
  file =File.open("POSCAR","r")
  file.each do |f|
    text << f
  end
  return text
end

set_newPOSCAR()

def make_newPOSCAR
  pos = []
  text = read_POSCAR()
  $n_multi.times do
    aj=rand(text[5])+1+6 #原子番号決め
    pos=text[aj].split(nil)
    for i in 0..2
      rnd=2.0*rand()-1.0
      dd=rnd*$init_vals[0] #移動させる原子座標のベクトル
      pos[i]=pos[i].to_f+dd #新しい原子座標
      if pos[i] > 1.0 then
        pos[i]=pos[i].to_f-1.0
      elsif pos[i] < 0.0 then
        pos[i]=pos[i].to_f+1.0
      end
    end
    end
    text[aj]=sprintf("%16.9f %19.9f %19.9f\n",pos[0],pos[1],pos[2]) #aj
    を text に移す
  end
  output_poscar = File.open("new_POSCAR","w") #new_POSCAR 新規作成
  output_poscar.write(text)
  output_poscar.close
end

def get_str_OUTCAR(x) #OUTCAR から文字列を読み取る

```

```

outcar=[]
iter=0
number=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

def get_energy_OUTCAR #OUTCAR からエネルギーを取り出す
  outcar=get_str_OUTCAR("ION-ELECTRON")
  return outcar[1].split(" ")[4].to_f
end

def Energy #POSCAR からエネルギー計算
  system("rm -rf OUTCAR")
  system("ruby pseudoVASP.rb POSCAR > OUTCAR")
  return get_energy_OUTCAR()
end

energy_before = Energy()

n_max.times do |i|
  if i%(n_max/10)==0 then printf(".") end
  make_newPOSCAR() #new_POSCAR 作成
  system("cp POSCAR sub_POSCAR")
  system("cp new_POSCAR POSCAR")
  energy_new = Energy()

  #新たな配置の受け入れを計算
  dE = energy_new - energy_before
  if dE<=0 then

```

```

        n_arng << energy_new
        energy_before = energy_new
    elsif dE>0 then
        prb = exp(-dE/T)
        rnd = rand() #0..1 float を作成
        if rnd<prb then
            n_arng << energy_new
            energy_before = energy_new
        else
            system("cp sub_POSCAR POSCAR")
        end
    end
end
end
printf("\n 採択率:%10.5f\n",n_arng.length/n_max.to_f)

File.open("e_change.txt","w"){|file|
  n_arng.each do |tmp|
    file.puts(tmp)
  end
}

stop=Time.now #計測終了
printf("\n 計算時間 (sec):%10.5f\n", stop-start)

```

## 6.1.2 NPT-MC

### *NPT-MC* のプログラム

```

\begin{enumerate}
\item -- coding: utf-8 --
\end{enumerate}
include Math
require "pp"

puts "温度を入力してください。"
$Temp=gets.chomp.to_f
puts "試行回数を入力してください。"
n_max=gets.chomp.to_i

```

```

puts "移動距離の上限を，原子間距離に対する比で入力してください．"
$d_max=gets.chomp.to_f
puts " 体積変化の上限を入力してください．"
$init_vals2 = gets.chomp.to_f
puts "同時に原子を動かす数を入力してください．"
$n_multi=4

start = Time.now #計測開始

n_arng = []

def get_str_OUTCAR(x) #OUTCAR から文字列を読み取る
  outcar=[]
  iter=0
  number=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

def get_force
  f=[]
  posi_f=get_str_OUTCAR("TOTAL-FORCE")
  for i in 0..$poscar_text[5].to_f-1 do
    f_pre=[0.0,0.0,0.0]
    f_pre[0]=posi_f[i].split(" ")[3].to_f
    f_pre[1]=posi_f[i].split(" ")[4].to_f
    f_pre[2]=posi_f[i].split(" ")[5].to_f
    f << f_pre
  end
  return f
end

```



```

def set_r
  r=[]
  for i in 0..$poscar_text[5].to_f-1 do
    r_one=[0.0,0.0,0.0]
    pos=$poscar_text[i+7].split(nil)
    for j in 0..2 do
      r_one[j]=pos[j].to_f-0.5000
      if r_one[j] < 0
        r_one[j]=-r_one[j]
      end
    end
    r << r_one
  end
  return r
end

def virial #Virial の計算
  a=0.0
  for i in 0..$poscar_text[5].to_f - 1 do
    r=set_r()
    f=get_force()
    for p in 0 ..2 do
      a+=r[i][p]*f[i][p]
    end
  end
  return w=a/3
end

def get_volume
  vec_a = []
  vec_b = []
  vec_c = []

  vec_a=$poscar_text[2].split(nil) #格子ベクトル
  vec_b=$poscar_text[3].split(nil)
  vec_c=$poscar_text[4].split(nil)
  a_x = vec_a[0].to_f
  a_y = vec_a[1].to_f
  a_z = vec_a[2].to_f

```

```

    b_x = vec_b[0].to_f
    b_y = vec_b[1].to_f
    b_z = vec_b[2].to_f
    c_x = vec_c[0].to_f
    c_y = vec_c[1].to_f
    c_z = vec_c[2].to_f
    volume=(a_x*b_y*c_z)+(a_y*b_z*c_x)+(a_z*b_x*c_y)
            -(a_x*b_z*c_y)-(a_y*b_x*c_z)-(a_z*b_y*c_x)
    return volume
end

def get_distance #体積からおよその原子半径を求める
    r_3=(get_volume()/$poscar_text[5].to_f)/(4*PI/3)
    r=r_3**(1.0/3.0)
    return r
end

def set_newPOSCAR()
    system("cp Al_POSCAR POSCAR")
    text = []
    file =File.open("POSCAR","r")
    file.each do |f|
        text << f
    end
    $poscar_text = text
    $init_vals = get_distance/(get_volume**(1.0/3.0))*$d_max
end

def read_POSCAR()
    text = []
    file =File.open("POSCAR","r")
    file.each do |f|
        text << f
    end
    return text
end

set_newPOSCAR()

```

```

def make_newPOSCAR
  pos = []
  text = read_POSCAR()
  $n_multi.times do
    aj=rand(text[5])+1+6 #原子番号決め
    pos=text[aj].split(nil)
    for i in 0..2
      rnd=2.0*rand()-1.0
      dd=rnd*$init_vals #移動させる原子座標のベクトル
      pos[i]=pos[i].to_f+dd #新しい原子座標
      if pos[i] > 1.0 then
        pos[i]=pos[i].to_f-1.0
      elsif pos[i] < 0.0 then
        pos[i]=pos[i].to_f+1.0
      end
    end
    text[aj]=sprintf("%16.9f %19.9f %19.9f\n",pos[0],pos[1],pos[2]) #aj
    を text に移す
  end
  output_poscar = File.open("new_POSCAR","w") #new_POSCAR 新規作成
  output_poscar.write(text)
  output_poscar.close
end

def make_newPOSCAR2
  text = read_POSCAR()
  rnd=2.0*rand()-1.0
  dd=1.0+rnd*$init_vals2 #移動させる原子座標のベクトル
  pp dd
  for j in 0..$poscar_text[5].to_i-1 do
    aj=j+1+6 #原子番号決め
    pos=text[aj].split(nil)
    for i in 0..2
      pos[i]=pos[i].to_f*dd #新しい原子座標
      if pos[i] > 1.0*dd then
        pos[i]=pos[i].to_f-1.0*dd
      elsif pos[i] < 0.0 then
        pos[i]=pos[i].to_f+1.0*dd
      end
    end
  end
end

```

```

        end
    end
    text[aj]=sprintf("%16.9f %19.9f %19.9f\n",pos[0],pos[1],pos[2]) #aj
をtextに移す
    end
    output_poscar = File.open("new_POSCAR","w") #new_POSCAR新規作
成
    output_poscar.write(text)
    output_poscar.close
end

def get_energy_OUTCAR #OUTCARからエネルギーを取り出す
    outcar=get_str_OUTCAR("ION-ELECTRON")
    return outcar[1].split(" ")[4].to_f
end

def Energy #POSCARからエネルギー計算
    system("rm -rf OUTCAR")
    system("ruby pseudoVASP.rb POSCAR > OUTCAR")
    return get_energy_OUTCAR()
end

def Rho #POSCARからrhoを計算
    system("rm -rf OUTCAR")
    system("ruby pseudoVASP.rb POSCAR > OUTCAR")
    w=virial()
    n=32.0
    pv=n*$Temp+w
    rho = pv+Energy()
    print w.to_s+", "+rho.to_s+"\n"
    return get_energy_OUTCAR()
end

energy_before = Rho()

n_max.times do |i|
    if i%(n_max/10)==0 then printf(".") end
    make_newPOSCAR() #new_POSCAR作成 原子移動
    make_newPOSCAR2() #new_POSCAR作成 体積変化

```

```

system("cp POSCAR sub_POSCAR")
system("cp new_POSCAR POSCAR")
energy_new = Rho()

#新たな配置の受け入れを計算
dE = energy_new - energy_before
if dE<=0 then
  n_arng << energy_new
  energy_before = energy_new
elsif dE>0 then
  prb = exp(-dE/$Temp)
  rnd = rand() #0..1 float を作成
  if rnd<prb then
    n_arng << energy_new
    energy_before = energy_new
  else
    system("cp sub_POSCAR POSCAR")
  end
end
end
end
printf("\n 採択率:%10.5f\n",n_arng.length/n_max.to_f)

File.open("e_change.txt","w"){|file|
  n_arng.each do |tmp|
    file.puts(tmp)
  end
}

stop=Time.now #計測終了
printf("\n 計算時間(sec):%10.5f\n", stop-start)

```

## 関連図書

- [1] 西谷滋人著「固体物理の基礎」(森北出版, 2006) pp.109-115.
- [2] 神山新一著「モンテカルロ・シミュレーション」(朝倉書店, 1997) pp.107-111.
- [3] 中井遙著「pseudoVASP の開発」, 関西学院大学理工学部 2012 年度卒業論文.