修士論文

モンテカルロシミュレーションによる Alの自由エネルギー計算

関西学院大学理工学研究科
情報科学専攻 西谷研究室 M2339 細見 有希

指導教員　西谷滋人　教授

概 要

素材の設計を行う上で系の自由エネルギーを変えることは重要な要素である．第一原理計算におけるフォノン計算のパッケージが開発され，基礎状態の有限温度の予想が示された．第一原理計算は原理的に基底状態の計算である．そして有限温度の計算には，擬調和振動近似に基づいた phonon 計算のパッケージがいくつか開発されている．しかし，半導体や相変態温度近傍での振る舞いには，非調和の影響を取り込むことが不可欠である．だが現在，そのような計算パッケージは提供されていない．

分子動力学シミュレーションでは，自由エネルギーの絶対値を得ることは極めて難しい．一方，モンテカルロ・シミュレーションの分野では自由エネルギーを求める手段として Frenkel 法が存在する．Frenkel 法では自由エネルギーは，系の理想状態である Einstein 結晶から現実の結晶までのエネルギー変化を直接積分することによって得られる．

第一原理計算による Frenkel 法にはいくつかの課題がある．まず，Einstein 結晶において，系全体に起こる rotation がエネルギー計算に影響を与えるため，これを止めたい．この課題に対して，rotation が起きている座標に対し，制限をかけることを提案した．本研究では，モンテカルロ・シミュレーションの移動を行う前と行った後の座標から系全体のモーメントを求め，それが 0 になるようにいくつか選んだ原子の座標移動を行うことで，rotation に制限をかけることを試みた．

また Einstein 結晶のエネルギー計算を行うプログラムである Einstein を開発した．Einstein では，モンテカルロ・シミュレーションの入力として与えた初期座標と座標移動を行った後の各座標の距離の二乗和からエネルギーを求める．

次に，Einstein 結晶のエネルギーから現実の結晶までのエネルギー変化の計算を行うプログラムである Transition によりエネルギーの推移の計算を試みた．しかし，現実の結晶のエネルギーとして，第一原理計算ソフト VASP の振る舞いをまねて，高速に計算する pseudoVASP を用いて予備計算を行ったところ，Einstein 結晶のエネルギー差が異常な振る舞いを示した．この症状を調べたところ，エネルギーが一定の値に収束せずに徐々に下がっていること，ある試行で値で急激に下がっていることがわかった．そこで，それぞれエネルギーが変化する前後の座標を取り出し，原子の挙動を詳しく調べた．

徐々にエネルギーが下がった原因は，系には rotation 以外に drift も起こっているため，移動後の系の rotation に制限をかけただけでは，移動前の系と比べると平均位置がずれてしまい，エネルギーが徐々に下がったためである．これを解決するためには rotation だけでなく drift にも制限をかける必要がある．またエネルギーが急速に下がった原因は，Einstein において，周期的境界条件を考慮したエネルギー計算を行っていなかったため，原子が境界を超えて反対側に移動した時に，

距離が急激に上昇したためである．そのため，Einstein の距離を計算する箇所に周期的境界条件を考慮した条件文を追加する必要がある．

目 次

第1章　諸言

近年，第一原理計算ソフトの操作性，なら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 [1].

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，調和振動子近似値にもとづいたフォノン (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，非調和の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.

一般的にこのような計算には，有限温度での分子動力学 (molecular dynamics) シミュレーションが取られるが，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 [2] が存在する．Frenkel 法は，基準状態として Einstein モデルのエネルギーをとり，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う手法である．しかしながら，これを実際におこなった先行研究では，現実での自由エネルギーが負の無限大に発散してしまい，値は求まらなかった [3]．これは，原子が周囲の原子と力を作用しあうことによって系全体に rotation(回転運動) や drift(並進運動) が起こるためであった.

本研究では，モンテカルロ・シミュレーションの Frenkel 法を行うため，系の rotation に制限を与え，エネルギー計算を行う．本来の系の運動に制限を与えることにより，原子の自由度が凍結することが予想される．そこで，系の rotation に制限のあるものとないものでそれぞれモンテカルロ・シミュレーションを行い，結果を比較することで制限が与える影響を検討した．またモンテカルロ・シミュレーションは，計算するときに与える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，最適なパラメータについても調査する.

第2章　モンテカルロ・シミュレーションのFrenkel法

この章では，本研究で行うモンテカルロ･シミュレーションの原理，及び，Frenkel法の概要について論じる．

2.1　熱平衡モンテカルロ・シミュレーション

統計力学で使われる平衡モンテカルロ法はありそうな状態を次々と生成しながら平均を取るという手法である．

2.1.1　モンテカルロ・シミュレーションの原理

モンテカルロ法は，乱数により系の分子の微視的状態を作成していく手法である．この微視的状態の作成は，分子動力学法の状態点の軌跡に相当するものである．ある微視的状態の出現する確率は，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．しかし現在，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メトロポリス (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，確率密度は，前もって分からないものである．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 [4] にどのようにして正準集団の平均が得られるかが，マルコフ過程と遷移確率，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．

2.1.2　マルコフ過程と推移確率

平衡モンテカルロ法は状態の生成にマルコフ過程を使う．これは，ひとつの状態μから，新しい状態νを生成するルールを定めている．μからνへ移る確率を遷移確率とよび，マルコフ過程においてはすべての遷移確率が現在の状態μ，ν

だけで決まり，それ以前に系がどのような状態を経てきたかにはよらない．遷移確率 $P(\mu \to \nu)$ は，次の総和則

$$\sum_{\nu} U(\mu \to \nu) = 1 \tag{2.1}$$

を満たさなければならない．なぜなら，マルコフ過程は μ という状態が与えられたら，必ず何らかの状態 ν を生成しなければならないからである．モンテカルロ・シミュレーションにおいては，マルコフ過程は μ という状態が与えられたら，必ずなんらかの状態のマルコフ連鎖を生成する．例えば，μ という状態から始めて，次の状態 ν を作り，それをまたマルコフ過程に食わせて次の状態を作る，という操作を繰り返す．このマルコフ過程は，どのような初期状態から始めても最終的な分布状態が正準集団になるように選ぶ必要がある．これらを実現するためにはマルコフ過程に更に二つの条件，エルゴード性と詳細つりあいの条件を課す必要がある．

2.1.3 エルゴード性

エルゴード性条件とは，十分に長い間マルコフ過程を続ければ，系のすべての状態に他の状態からたどり着くことが可能でなければならないという要件である．エルゴード性条件から，直接に一度の遷移でたどり着くマルコフ過程の遷移確率のいくつかはゼロでも構わないことがわかる．ただし，どのような二つの状態を取り出しても，遷移過程を何度か繰り返せばその 2 つの状態を結ぶ確率が有限の値をとる行き方が，少なくともひとつは存在する必要がある．このような条件を満たせば，マルコフ過程は平衡状態に落ち着くことが保証されている．

2.1.4 詳細つりあいの条件

マルコフ過程に課すもう一つの制約は，マルコフ連鎖で生成する分布が平衡状態において正準分布であることを保証するために必要である．これは，平衡状態がどのような状態であるかを記述することから導くことが出来る．

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 が負の値の時は全て採択し，正の値の時には，$\exp(E/kT)$ 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．

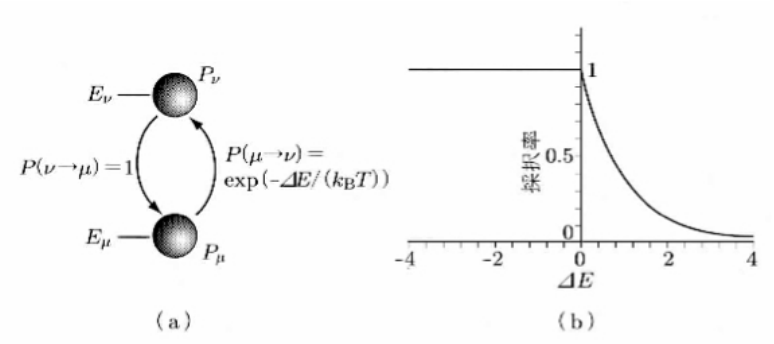


図 2.1: (a) 状態の遷移を示す模式図．2 つの状態 μ，ν で．$E_\nu > E_\mu$ と仮定している． (b) 遷移の採択率.

2.1.5 正準集団

正準集団（カノニカルアンサンブル）は，対象系と等価な多くの複製からなる．それらは，どれも同じ粒子数，温度，体積を持つ．それぞれの系は，集団の他のメンバーとエネルギーをやりとりしながら，それらの制約の範囲内で揺らぐことが許されている．すなわち正準集団の分布は $\exp(-E/(k_B T))$ となる．

2.2 Frenkel 法

Ti は低温においては hcp 構造が，高温においては bcc 構造が安定である．ところが，bcc 構造の phonon 分散曲線は，図 2.2 に示したとおり，負となる分岐が 2 箇所ある．これは，bcc 構造と hcp 構造を連続的に変形させた対応関係を示す Burger's Path(図 2.3) において，原子面が交互に反対方向にずれるシャッフル (suffule) に対応する 1 の変形においてポテンシャルが負の曲率を持つ不安定性に対応している (図 2.4)．したがって，phonon 分散曲線から振動自由エネルギーを見積もる擬調和振動子近似では Ti の bcc 構造の自由エネルギーが計算できない.

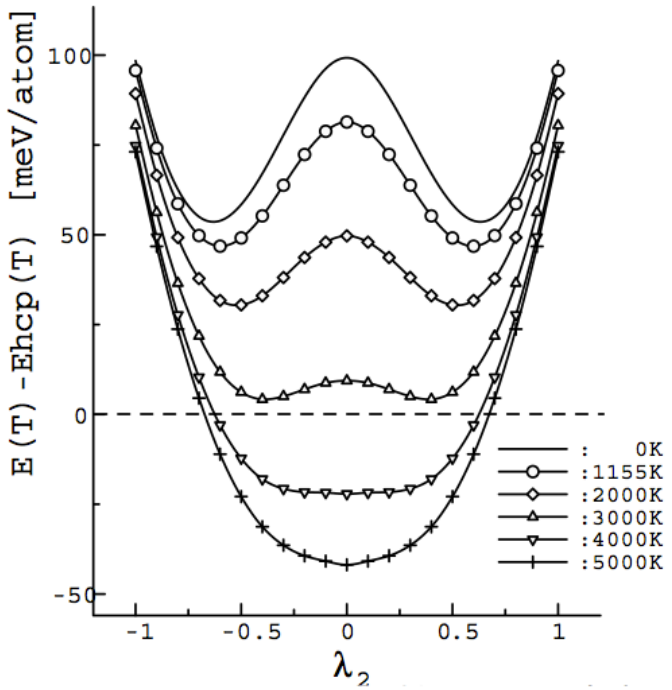


図 2.2: bcc 構造の phonon 分散曲線.

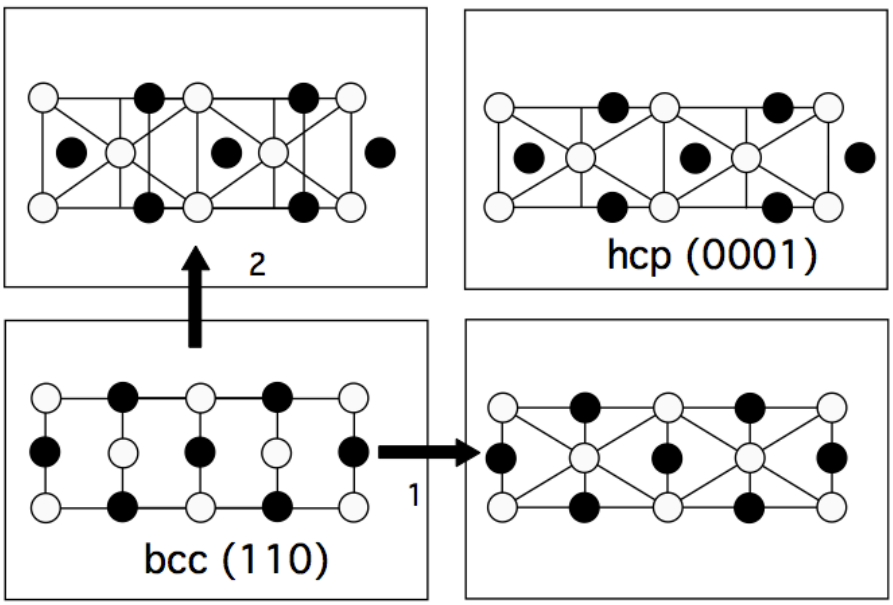


図 2.3: bcc 構造と hcp 構造の変形.

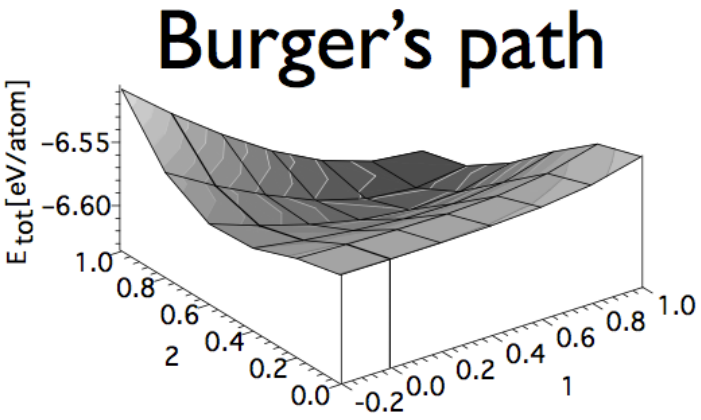


図 2.4: Burger's Path

モンテカルロ・シミュレーションのFrenkel法はEinsteinモデルという理想状態のモデルを用いて，理想状態のエネルギーから推移して現実状態のエネルギーを求める手法をとる．$\lambda = 1$を現実の結晶，$\lambda = 0$をEinstein結晶とする連続した状態量を考えたとき自由エネルギーは式2.3の計算で求まる．Einstein結晶では調和振動子と見なせて，自由エネルギーが解析的に求まる．中間の値は，2つの状態を混ぜ合わせた状態で，調和振動から非調和振動までの遷移状態を生成する事に対応する．

$$U(\lambda) = \lambda U_{\mathrm{true}} + (1 - \lambda) U_{\mathrm{virtual}} \tag{2.2}$$

Fは自由エネルギーを表しており，

$$F(\lambda = 1) = F(\lambda = 0) + \int_0^1 \langle \frac{\partial U(\lambda)}{\partial \lambda} \rangle d\lambda \tag{2.3}$$

式2.3のようにFは$\lambda = 0$のFを基準にして$du/d\lambda$を$\lambda = 0$から1まで積分することで求まる．

$$\langle \frac{\partial U(\lambda)}{\partial \lambda} \rangle = \langle U_{\mathrm{true}} - U_{\mathrm{virtual}} \rangle \tag{2.4}$$

ここで$du/d\lambda$は式2.2を微分して，式2.4のように求まる．これはU_{true}とU_{virtual}の差をあらわし，この値をモンテカルロのシミュレーション中に記録することで，数値積分が可能となる．しかし，実際にFrenkel法の実験を行った結果，現実のエネルギーは求まらなかった．その結果を図2.5に示した．この図の曲線の$\lambda = 1$では，負に発散していることが読み取れる．

2.2.1 Einsteinモデル

固体の熱的振る舞いを表す最も単純なモデルであるEinstein(アインシュタイン)モデルを取り上げる．現実の原子は，それぞれ振動しながら互いがばねで結びついているかのように影響しあっている．そうして，すべての原子は，周りの原子から受ける力によって平衡位置にあり，有限温度ではその周りで揺らいでいる．Einsteinは，すべての原子が相互作用しない仮定したモデルを構築した．これを見直して図2.6の左のように原子を平衡位置にばねで釘付けしてしまうというのがEinsteinモデルである．こうすると原子は1次元の調和振動子とみなすことが出来る．

2.2.2 モーメントの計算による系の制限

先行研究でのFrenkel法で値が求まらなかった原因に，Einsteinモデルで起きる系のrotato(回転運動)とdrift(並進運動)が挙げられる．すなわち，推移状態から現実状態のエネルギーを求める際に，系が平衡位置から移動してしまったために，

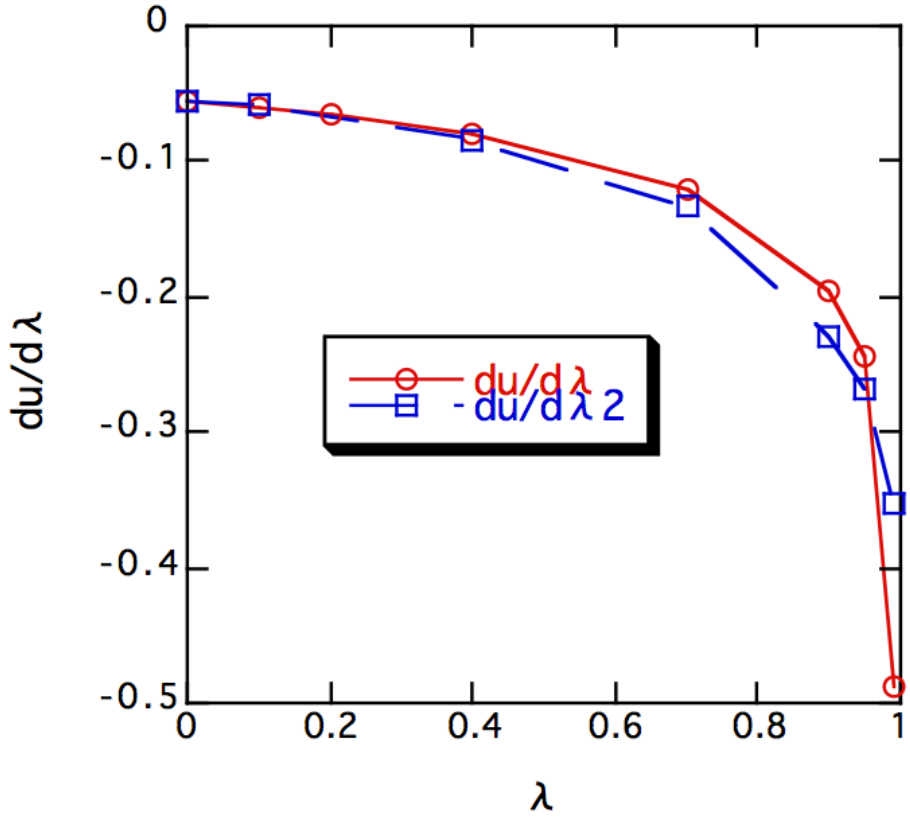


図 2.5: Frenkel 法のシミュレーション結果．λ=1 で値が負の無限大に発散していることが分かる．

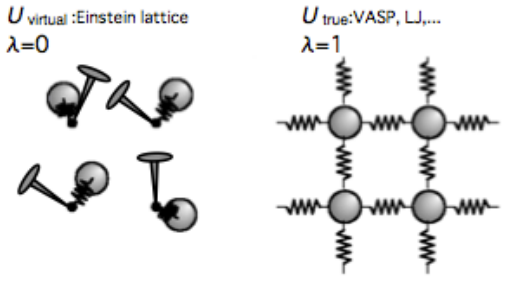


図 2.6: Einstein 結晶（左）と現実の結晶（右）．

値が無限大へと発散してしまったためである (図 2.7)．本研究では，これらの系の運動に制限 (constraint) をかけることで現実状態のエネルギーを求めることを試みる (図 2.8).

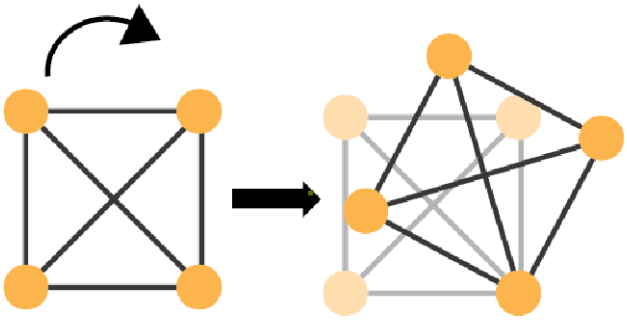


図 2.7: 系に rotation が起こる様子

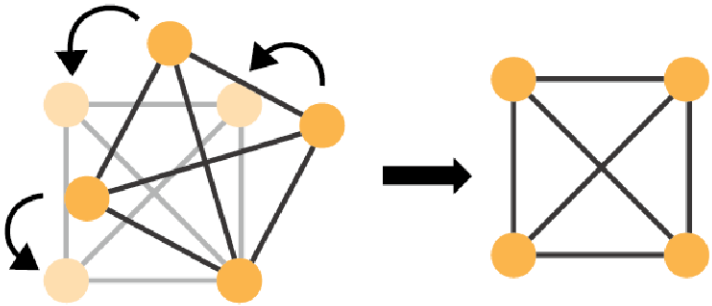


図 2.8: rotation に制限をかける

第3章　計算手法

この章では，実際にモンテカルロ・シミュレーションを行うのに用いたアルゴリズム，パラメータ，計算手法とその結果について説明する．

3.1　モンテカルロのアルゴリズム

本シミュレーションでは京口が開発した NVT 一定のモンテカルロのプログラムを用いる [7]．このアルゴリズムでは，原子の微小移動の並び替えを行い，熱平衡状態のシミュレーションを行う．

3.1.1　NVT 一定のモンテカルロ

本シュミレーションでは NVT 一定のモンテカルロアルゴリズムを用いる．本研究でのモンテカルロ・シミュレーションは原子系の振動エネルギー変化をとっている．すなわち，初期の原子座標からある原子をランダムに微小量移動させ，新たな原子座標を生成する（図 3.1）．

その系のエネルギー変化にしたがって，採択，もしくは棄却を決定する．その際の基準を，規格化した温度 T とする．

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 $\exp(-\Delta E/T) <$ 乱数の確率で採用．
4. ステップ 2 以下を繰り返す．

以上の流れを，図 3.2 に示した．

今回は予備研究として 32 個の Al の原子座標を入力として用いる．図 3.3 はそのモデルを表している．

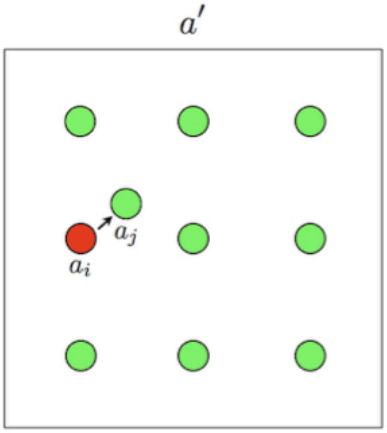


図 3.1: 新たな配置 a' の模式図．a_i 座標を，a_i から微小に離れた a_j 座標へ移動させる．この移動操作をすべての原子座標に行い，変化した配置を $a' + \delta a'$ 座標として出力する．

3.1.2 入力パラメータについて

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．

温度 T

温度 T は $k_{\mathrm{B}}T$ の略記であり k_{B} の相対温度として規格化している．例えば，$T = 0.1$ のとき $k_{\mathrm{B}}T = 0.1$ となる．ここで k_{B} はボルツマン (Boltzmann) 定数とよばれる物理定数で，気体定数 R と Avogadro 数 N_{A} とは

$$k_{\mathrm{B}} = R/N_{\mathrm{A}} = 8.617 \times 10^{-5}\,[\mathrm{eV/K}] \tag{3.1}$$

という関係をもつ．$k_{\mathrm{B}}T = 0.1$ のとき，T について解くと，式 3.1 より

$$T = \frac{0.1}{8.617 \times 10^{-5}} = 1160.5\,[\mathrm{K}] \tag{3.2}$$

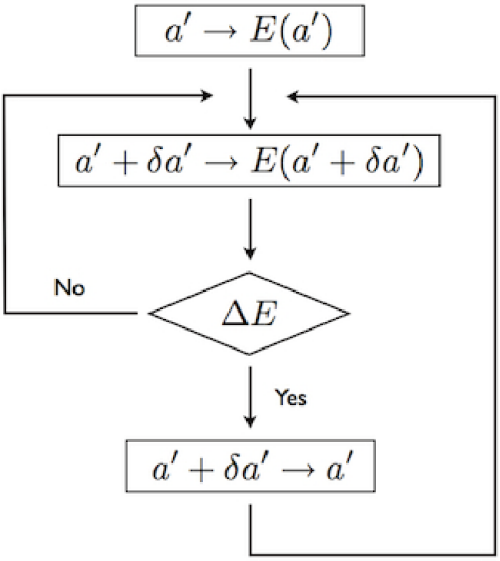


図 3.2: 原子の微小移動のフローチャート

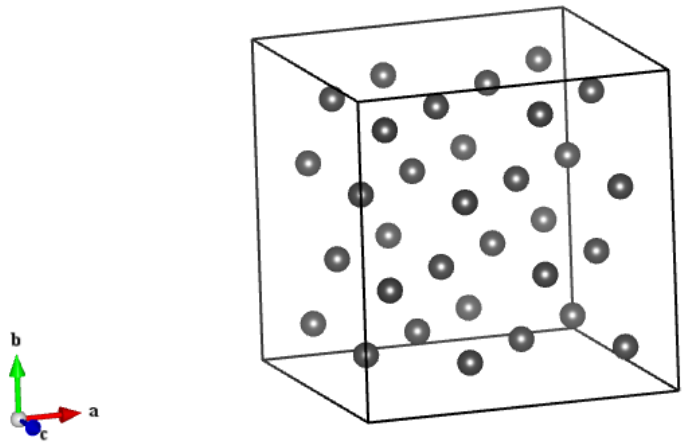


図 3.3: Al のモデル

となる．これより

$$T = 0.1 = 1160.5\,[\mathrm{K}] \tag{3.3}$$

となる．

3.2 入出力ファイル

ここでは入力に使用するファイル，及び出力されるファイルの詳細について説明する．

3.2.1 POSCAR

モンテカルロ・シミュレーションのエネルギー計算をする原子座標の入力には，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．今回初期座標として使用するのは Al の 32 原子の POSCAR であり，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．

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
```

3.2.2 OUTCAR

pseudoVASP では入力された POSCAR を元に Lennard-Jones ポテンシャルでエネルギーを計算し結果を OUTCAR というファイルに出力する．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．以下に，計算終了後に作成される OUTCAR ファイルの一例を示す．OUTCAR の出力形式は pseudoVASP の出力形式になぞらえている．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．本計算では計算終了後のエネルギーを取得している．

OUTCAR

```
1.0
FREE ENERGIE OF THE ION-ELECTRON (eV)
---------------------------------------------------------------------
free energy TOTEN =   -31.9063879 eV

POSITION                                       TOTAL-FORCE (eV/Angst)
---------------------------------------------------------------------
  0.123203  0.124117  0.126056  0.038471 -0.008999 -0.045972
  0.121941  0.123512  0.626011  0.055840 -0.009511 -0.048882
  0.125222  0.622660  0.122869  0.007231  0.028180  0.021592
  0.128122  0.623898  0.623754 -0.042219  0.017667  0.001052
  0.623777  0.127921  0.126610  0.020458 -0.062782 -0.010191
  0.625498  0.123358  0.626233  0.002251  0.045098 -0.054358
  0.627033  0.623055  0.124399 -0.039535  0.037572 -0.026853
  0.374906  0.372410  0.123792 -0.004010  0.040637  0.002524
  0.375369  0.373946  0.622522  0.012087  0.004768  0.042929
  0.375140  0.877187  0.122329 -0.002530 -0.056693  0.042680
  0.374769  0.875992  0.626016 -0.034366 -0.037437  0.007037
  0.877189  0.371112  0.127299 -0.040894  0.060167 -0.061609
  0.877277  0.374436  0.620545 -0.047015  0.017491  0.041155
  0.873387  0.875555  0.127754 -0.013387 -0.018123 -0.071542
  0.875133  0.871819  0.622777  0.007925  0.026394  0.036722
  0.372494  0.124559  0.374708  0.054884  0.008072  0.021112
  0.373307  0.124693  0.874685 -0.010860  0.016863 -0.012869
  0.375710  0.625578  0.373098 -0.064123 -0.000257 -0.050447
  0.377438  0.622461  0.877075 -0.002957  0.022297 -0.089424
  0.874227  0.122147  0.374275 -0.010350  0.041520  0.043740
  0.876419  0.124713  0.874920 -0.050897 -0.031019  0.002753
  0.874511  0.624836  0.370852  0.107245  0.009313  0.020719
  0.875285  0.623618  0.875071 -0.057057 -0.000814 -0.043111
  0.126996  0.374744  0.373003 -0.029081 -0.004304  0.033080
  0.122335  0.374161  0.871542  0.065426  0.004203  0.054362
  0.125756  0.874010  0.373508 -0.001583 -0.014409  0.016764
  0.121574  0.872396  0.875845  0.060206  0.021012 -0.015583
  0.626170  0.380409  0.375263 -0.033997 -0.208201 -0.063661
  0.626864  0.372057  0.873377 -0.005031  0.068308 -0.022128
  0.621798  0.874515  0.378008  0.033220  0.082537 -0.148995
  0.623549  0.876717  0.874637  0.058770 -0.065743 -0.003871
  0.626630  0.626144  0.610806 -0.034123 -0.033809  0.381277
Total CPU time used:         0.01596
```

3.3 原子系の rotation の制限

ここでは，モンテカルロの原子の微小移動の操作の後に起こる，系の rotation を止める手法について論じる．

3.3.1 角運動のモーメント

力学において，原点 O から点 P へ向かう位置ベクトル $\mathbf{x}_0$ と，点 P におけるベクトル量 $\mathbf{dx}$ との外積 $\mathbf{x}_0 \times \mathbf{dx}$ を，O 点まわりの $\mathbf{x}_0$ のモーメント (moment) という．外積は

$$\mathbf{x}_0 = \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} \tag{3.4}$$

$$\mathbf{dx} = \begin{pmatrix} dx \\ dy \\ dz \end{pmatrix} \tag{3.5}$$

としたとき、$\mathbf{x}_1 = \mathbf{x}_0 \times \mathbf{dx}$ をとおくと,

$$\mathbf{x}_1 = \begin{pmatrix} y_0 dz - z_0 dy \\ z_0 dx - x_0 dz \\ x_0 dy - y_0 dx \end{pmatrix} \tag{3.6}$$

で定義される量である．結果がベクトルになるため，$\mathbf{x}_1$ はベクトル積と呼ばれることもある (図 3.4)．この外積 $\mathbf{x}_1$ が 0 のとき，原点 O を中心とした角運動は起きない．

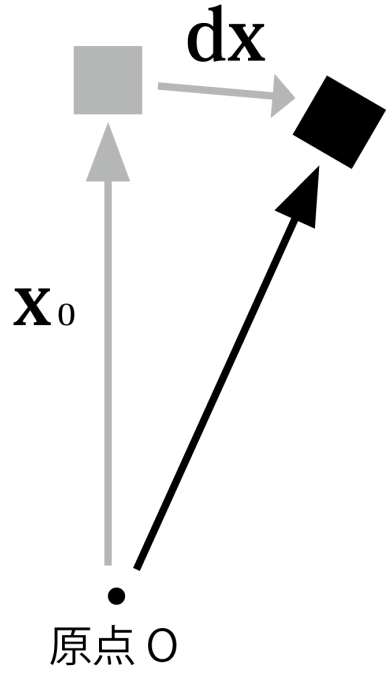


図 3.4: 角運動のモーメント.

そこで，移動後の座標から三点を選びモーメントが 0 になるような座標移動を行うことで系の rotation を止めることを試みる．

具体的には以下の手順により rotation に制限をかける．

1. モンテカルロのアルゴリズムにより，全原子の微小移動を行う．
2. 移動後の原子からランダムに 3 つの原子を選び，各原子の x 座標，y 座標，z 座標を 1 つずつ変数 dx, dy, dz として設定する．
3. 移動前の座標と，変数を含む移動後の座標で，外積を求める計算を行う．
4. 変数 dx, dy, dz を含むこの式の計算結果が 0 になるとき，モーメントは中和される．それらの変数を三元連立方程式として解き，結果として得られた値を，移動後のそれぞれの座標に代入する．

このようにして，系の rotation に対し制限をかけている系 (constraint) とかけていない系 (non-constraint) でシミュレーションを行った．

3.4 エネルギー計算

Frenkel 法では理想状態から現実状態のエネルギーの推移をとって求められる．現実のエネルギーの計算には pseudoVASP，理想状態のエネルギーの計算には Einstein というプログラムを用いた．そしてそれら二つのエネルギーを利用して，理想状態から推移させて現実状態のエネルギーを求める Transition というプログラムを開発した．

3.4.1 pseudoVASP

現実の系のエネルギー計算には，西谷研の中井が開発した pseudo(似非)VASP を用いた [6]．第一原理計算を行うソフトウェアである 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．モンテカルロ・シミュレーションでは，原子を移動する度に系のエネルギーを求めるため，このプログラムを VASP を用いて開発すると，多大な時間がかかる．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．pseudoVASP は，レナードジョーンズの原子間ポテンシャルを用いたエネルギー計算をとる．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 3.5 に示した.

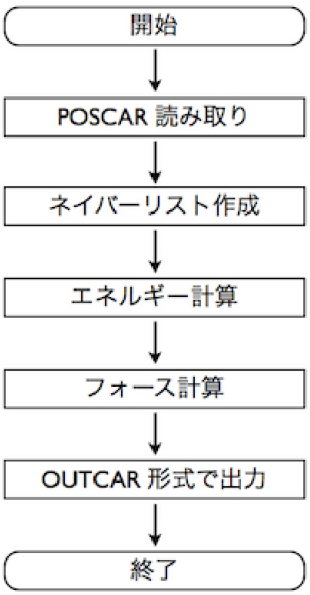


図 3.5: pseudoVASP における計算のフローチャート.

3.4.2 Einstein

系の理想状態のエネルギーは，初期の原子座標から現在の座標への移動距離を元に求められる．これを求める Einstein というプログラムを作成する．挙動は以下のとおり.

1. Al_POSCAR と POSCAR からそれぞれ原子座標を読み込む.
2. それぞれの位置 (site) の原子座標の差を dx，dy，dz とする.
3. その site の原子が持つエネルギーを $ei = -1.0 + 1/2.0k(dx^2 + dy^2 + dz^2)$ の式で求める.
4. すべての site の原子が持つエネルギーの合計 $e\_total = sum(ei) - 32.0$ が系の理想状態のエネルギーとなる.

ここで，ei を求める式に登場するばね定数 k を求める必要がある．単原子を微小移動させてエネルギーを求めるモンテカルロ・シミュレーションのプログラムにより，原子の移動距離 dis を変化させた時のエネルギーの変化を調べた (表 3.1).

表 3.1: dis/E

dis	E
-0.01	-31.9977973
-0.008	-31.9985911
-0.006	-31.9992078
-0.004	-31.999648
-0.002	-31.999912
0	-32.0
0.002	-31.999912
0.004	-31.999648
0.006	-31.9992078
0.008	-31.9985911
0.01	-31.9977973

これを二次曲線にプロットしたものを図 3.6 に示す.
この二次曲線が表す式は,

$$y = 22.03x^2 + 1.106E - 12x - 32 \tag{3.7}$$

となる．この式を

$$y = 1/2kx^2 - 32 \tag{3.8}$$

の形にしたい．後者の式に x と y を代入すれば自ずと k が求まる．つまり,

$$y = 22.03x^2 + 1.106E - 12x \tag{3.9}$$

の式を

$$y = 1/2kx^2 \tag{3.10}$$

の形に直すと.

$$y = 22.02622378x^2 - 32.00000036 \tag{3.11}$$

の式を導きだした． これを

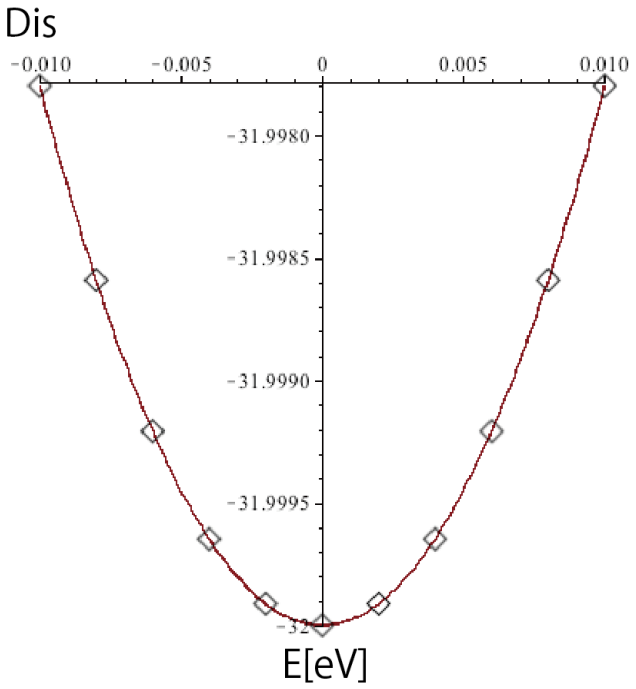


図 3.6: 原子の移動距離ごとにエネルギーをプロットした結果

$$y = 1/2 \times kx^2 - 32 \tag{3.12}$$

の形に直すと

$$y = 1/2 \times 44.06x^2 - 32.00 \tag{3.13}$$

よってばね定数 k は

$$k = 44.06 \tag{3.14}$$

となる．これを先ほどの site のエネルギーを求める式に代入すると，各 site のエネルギーは

$$ei = -1.0 + 1/2.0 \times 44.06 \times (dx^2 + dy^2 + dz^2) \tag{3.15}$$

で求めることが出来る．

第4章　シミュレーション結果

この章ではシミュレーションから得られた結果について記載する．系の rotation に constraint(制限) をかけたものでモンテカルロ・シミュレーションを行い，系が正準集団として正しく機能しているかを調べる．また，理想状態の系のエネルギーを求めるプログラム Einstein でエネルギー計算が正しく行われるかを調べ，更に，系の理想状態から現実状態へのエネルギーの推移を計算する Transition のプログラムを作成し，モンテカルロ・シミュレーションを行った．

4.1　rotationにconstraintをかけた系のモンテカルロ・シミュレーション

本シミュレーションでは温度 (T) と試行回数 (n) と各原子の最大移動距離 (d_max) をパラメータとして与える．

まず，系に rotation をかけた原子座標が，かけていないものと同じように平衡化するかを見る．図 4.1 に T=0.1,n=40000,d_max=0.005 の場合の系に constraint をかけたもののモンテカルロ・シミュレーションの結果を示した．横軸に試行回数を，縦軸に試行回数毎の採択された時のエネルギーをとっている．4 万回の試行に対して，この条件では，約半数の採択率となっている．また，試行回数 500 回程度で平衡化していることがわかる．

続いて，図 4.2 に，それぞれ constraint と non-constraint の系で，モンテカルロ・シミュレーションを温度変化させながら行った結果を示している．このシミュレーションでは，n=10000,d_max=0.005 を与え，それぞれ温度を 0.01〜0.1(T) まで 0.01 刻みで変化させ，平衡化する様子を見た．そして，平衡化した時のエネルギーの平均値をプロットし重ねて，比較した．横軸に設定温度を，縦軸に平衡化したエネルギーの平均を示している．この傾きが比熱となる．ここで行っているシミュレーションでは，比熱が全温度域で一定となることが期待される．この図では，constraint，non-constraint ともに期待通りほぼ一定である．

また，図 4.4，4.2 に constraint と non-constraint の温度を変えた時の揺らぎの幅をそれぞれ示した．温度の上昇とともに，揺らぎの幅も増加していることがわかる．このことから温度の上昇に伴って揺らぎの幅も大きくなっていることが分かる．

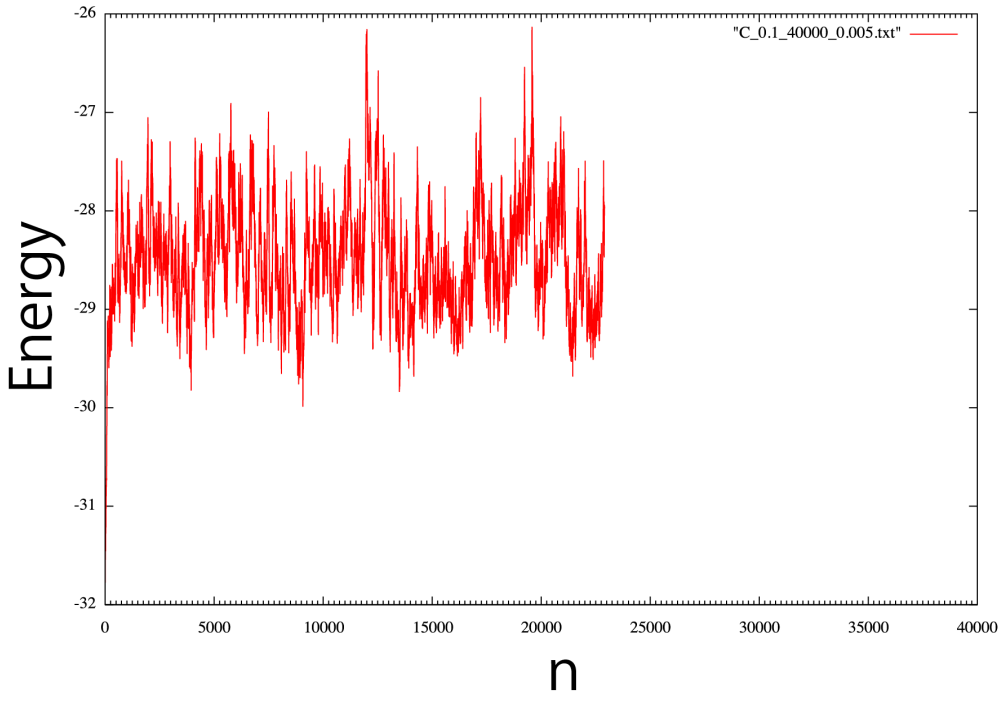


図 4.1: 系に constraint をかけた場合のモンテカルロシミュレーションの結果．横軸に試行回数 n，縦軸にエネルギー E をとっている．n が 2000 程度でエネルギーが一定の値を中心に揺らいでいる．

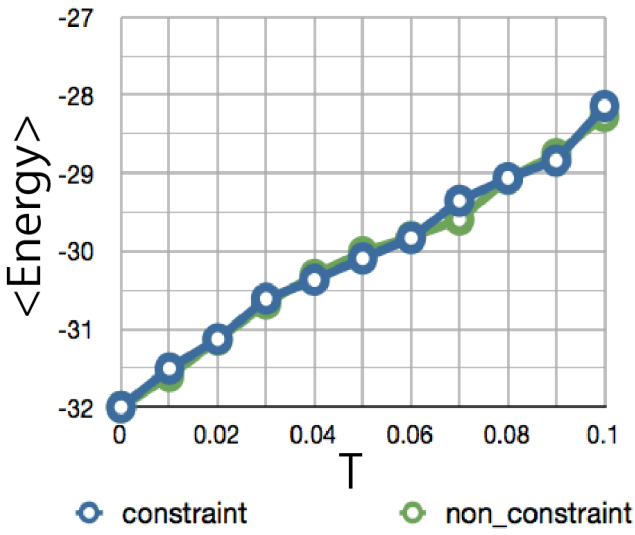


図 4.2: constraint の場合と non-constraint の場合のモンテカルロシミュレーションから得られたエネルギーの平均の比較．横軸に温度 T，縦軸にエネルギーの平均 $< E >$ をとっている．

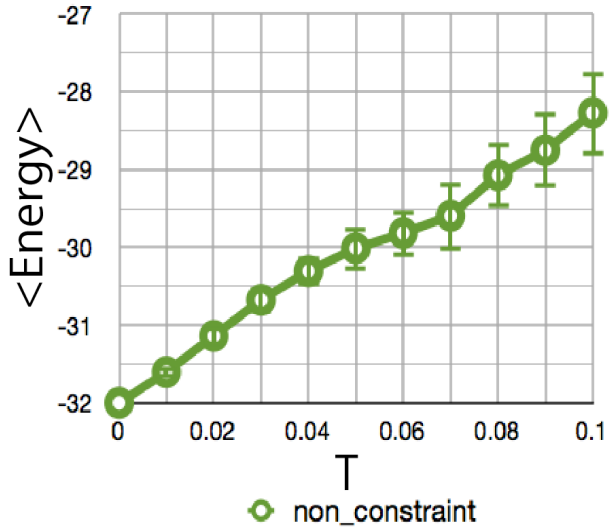


図 4.3: non-constraint の場合の温度ごとのエネルギーの平均と平均二乗誤差．横軸に温度 T，縦軸にエネルギーの平均 $< E >$ をとり，各値の平均二乗誤差をエラーバーとして示している．

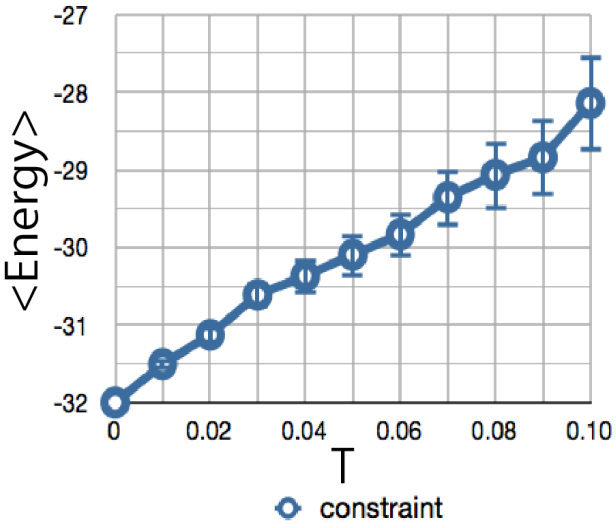


図 4.4: constraint の場合の温度ごとのエネルギーの平均と平均二乗誤差．横軸に温度 T，縦軸にエネルギーの平均 $< E >$ をとり，各値の平均二乗誤差をエラーバーとして示している．

4.2 Einsteinによるエネルギー計算

原子の理想状態のエネルギーの計算を行うEinsteinのプログラムを使いモンテカルロ・シミュレーションを行った．系はrotationにconstraintをかけたものを使い，パラメータはそれぞれ，T=0,0.02,0.04,0.06,0.08,0.1,n=10000,dis=0.005を用いた． 図4.5に，それぞれの温度でのエネルギーのプロット結果を示した．揺らぎの幅に差はあるものの，いずれも試行回数800〜1000回で熱平衡状態へ収束している．図4.6に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．

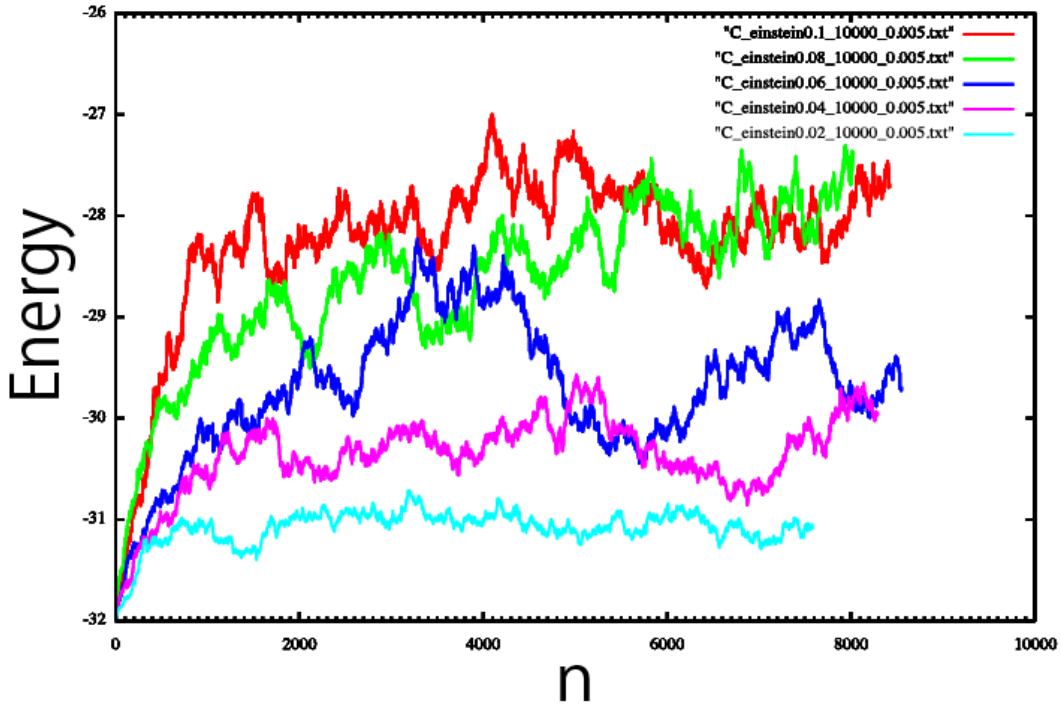


図4.5: Einsteinによるエネルギー計算を用いたモンテカルロ・シミュレーション結果．横軸に試行回数n，縦軸にエネルギーをとっている．

4.3 Transionによるエネルギー計算

系の理想状態から現実状態へのエネルギーの推移を計算するTransitionのプログラムを使いモンテカルロ・シミュレーションを行った．モンテカルロ・シミュレーションのパラメータはそれぞれ，T=0.05,n=10000,dis=0.005とした．理想状態のエネルギーをλ=0.0，現実状態のエネルギーをλ=1.0としている．図4.7に$\lambda = 0.0$から$\lambda = 1.0$に変化させた時の$du/d\lambda$の値をプロットした．λ=0.9まではエネルギーは順調に推移している．しかしλ=1.0では値が負に大きくずれている．

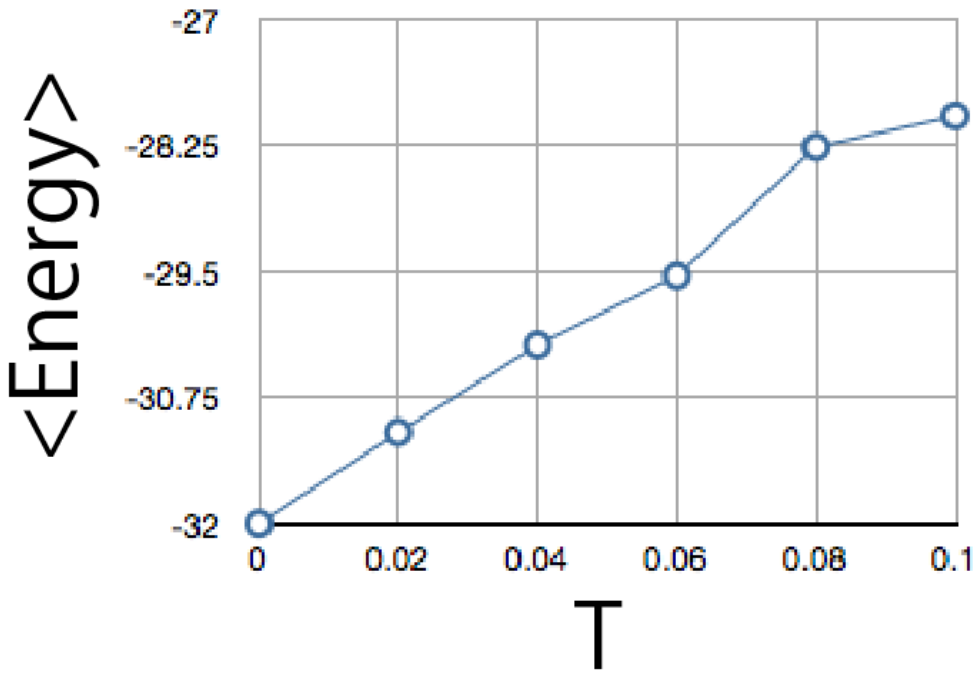


図 4.6: Einstein で求めたエネルギーの平均を温度ごとにプロットした結果．横軸に温度 T，縦軸にエネルギーの平均 $< E >$ をとっている．

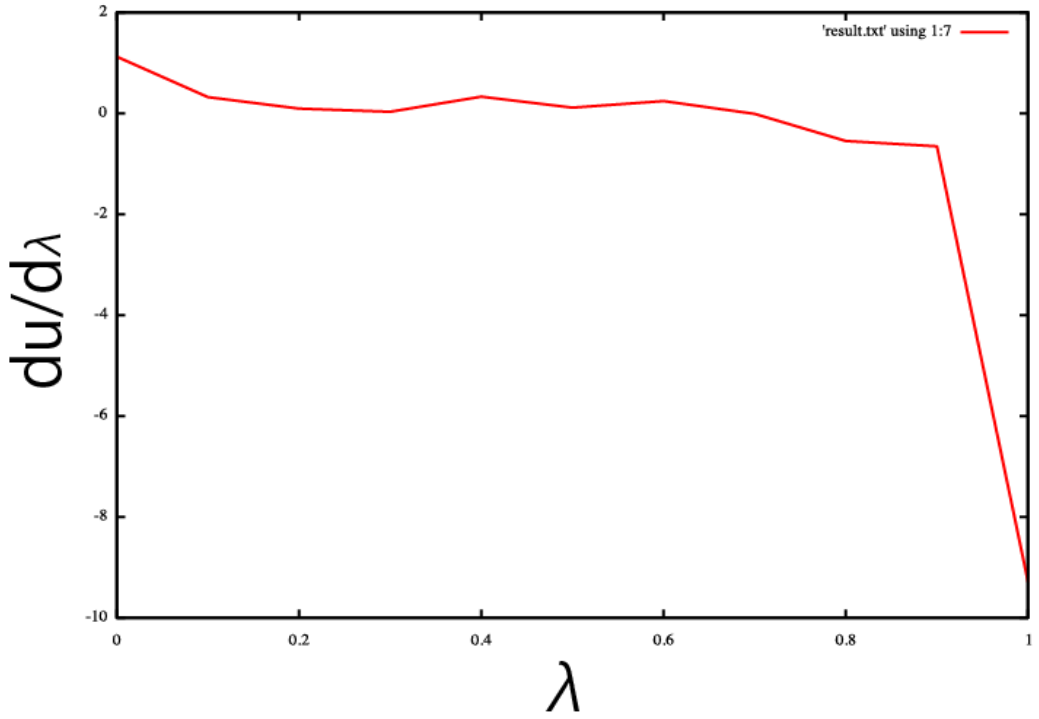


図 4.7: Transition により理想状態から現実状態へのエネルギーの推移の様子を示す．横軸に λ，縦軸に $du/d\lambda$ をとっている．

第5章　議論

この章では前章で示した結果を踏まえた考察を示す．まず，系の rotation に constraint をかけたことによる影響について検討する．4.1 の結果より，系に constraint のかけた状態でもエネルギーが熱平衡状態に達していたことから正準集団を再現していると考えられる．そのため，この結果からは，系に constraint を与えても，エネルギーの振る舞いには影響がないものと推察される．

そして，4.3 より，系の理想状態から現実状態へのエネルギーの推移を計算する Transition のプログラムでモンテカルロ・シミュレーションを行ったが，先行研究のエネルギー計算に同じく，λ=1.0，つまり現実での系のエネルギーが負の無限大に向かってしまい，求まらなかった．しかし，λ=0.9 までは，値が出ているところから，Transition のプログラム自体は正常に動いていることが確認できる．

次に，Transition で値が求まらなかった原因を検討する．図 5.1 に，Transition を λ=1.0 で行った結果を示した．緑線は $du/d\lambda = E_{\mathrm{VASP}} - E_{\mathrm{Einstein}}$，赤線は $u = \lambda E_{\mathrm{VASP}} + (1-\lambda)E_{\mathrm{Einstein}}$ を示している．

緑線は値が収束していないことから．系の重心がずれている可能性が示唆された．初期の原子座標である Al_POSCAR の重心と，実際に Transition のモンテカルロ・シミュレーションにより生成されたある POSCAR の重心，及び Al_POSCAR とのモーメントをそれぞれ求めた結果を図 5.2 に示す．Al_POSCAR の重心は [0.5, 0.5, 0.5]，POSCAR の重心は [0.488..., 0.498..., 0.496...] と．それぞれ 0.01〜0.02 ほどの差が見られる．対して，Al_POSCAR と POSCAR のモーメントはほぼ 0 に近い状態であるから，rotation は constraint がかけられていることがわかる．このことから，rotation を抑えただけで重心移動 (drift) が抑えられると予測したが，実際は移動することが示された．そのため，新たな POSCAR を生成するときに rotation だけでなく drift を明示的に抑える座標を生成する必要がある．

また，λ=1.0 のエネルギーの発散を抑えるために原子の移動距離の上限 dis を 0.04 に変えて Transition を行った結果を図 5.3 に示した．この結果では，緑の線が一部で急激にエネルギーが下がっているため，エネルギーの計算部になんらかのバグがあると考えられた．エネルギーが下がった部分の座標を検出したところ，試行回数 53829 回目で値が下がり， 53841 回目で戻っていることがわかった（図 5.4）．そこで，前後で原子位置をとりだし，VESTA で表示したところ，左上 2 つ目の原子が，境界を超えて，右側に移動していた（図 5.5，図 5.6，図 5.7）．このことから，周期的境界条件を，Einstein のエネルギー計算で考慮していなかったため，原子の移動前と移動後の差が大きくなってしまいエネルギーが急激に下がっ

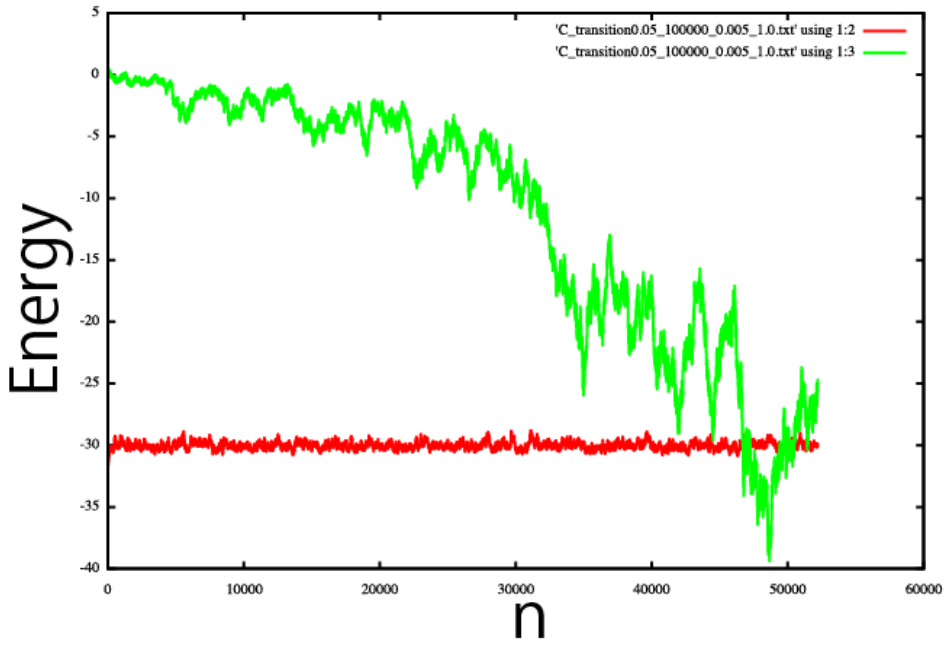


図 5.1: dis=0.005，λ=1.0 で Transition を用いてエネルギー計算した結果．横軸に試行回数 n，縦軸にエネルギーをとっている．

```
/Users/HOSOMIN/Desktop/2014_01% ruby serchBalance.rb Al_POSCAR
0.500000   0.500000   0.500000
/Users/HOSOMIN/Desktop/2014_01% ruby serchBalance.rb POSCAR
0.488392   0.498009   0.496361
/Users/HOSOMIN/Desktop/2014_01% ruby serchMoment.rb POSCAR
Total moments of the system:
   0.0000000005    0.0000000015  -0.0000000007
```

図 5.2: 重心を求めるプログラムにより，Al_POSCAR と POSCAR の重心をそれぞれ調べた．POSCAR は constraint をかけているため，モーメントはほぼ 0 に近い値が出ているが，重心は Al_POSCAR と比べてそれぞれ 0.01 ほどの差がある．

てしまったと考えられる．これを解決するために，Einstein 計算の距離計算において，周期的境界条件を取り入れる必要がある．

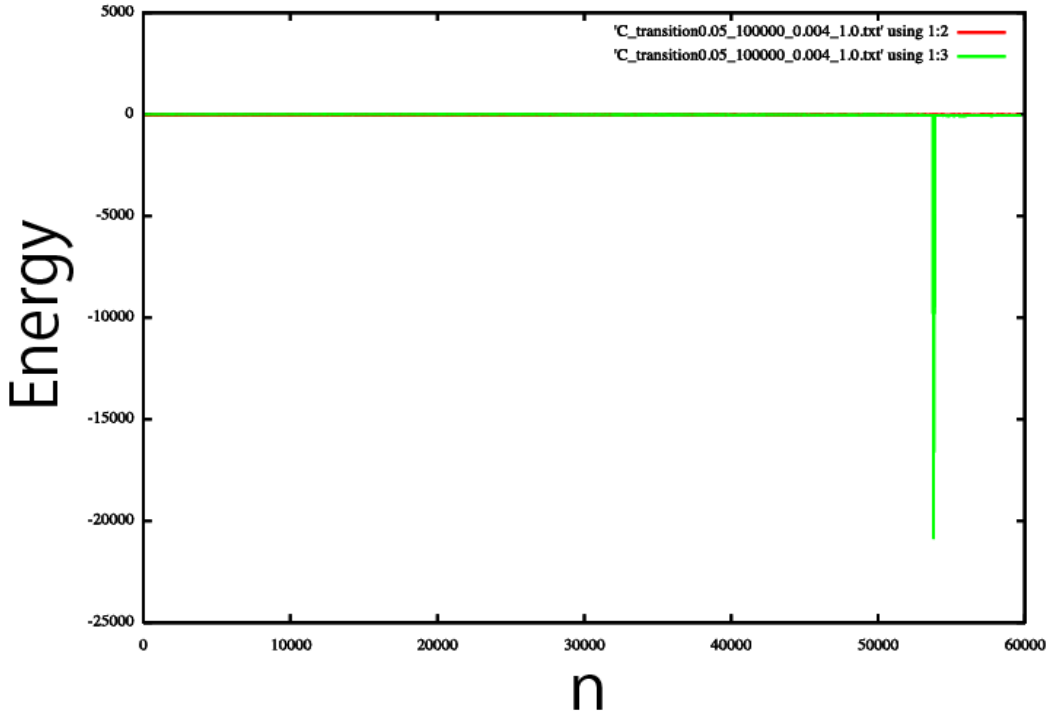
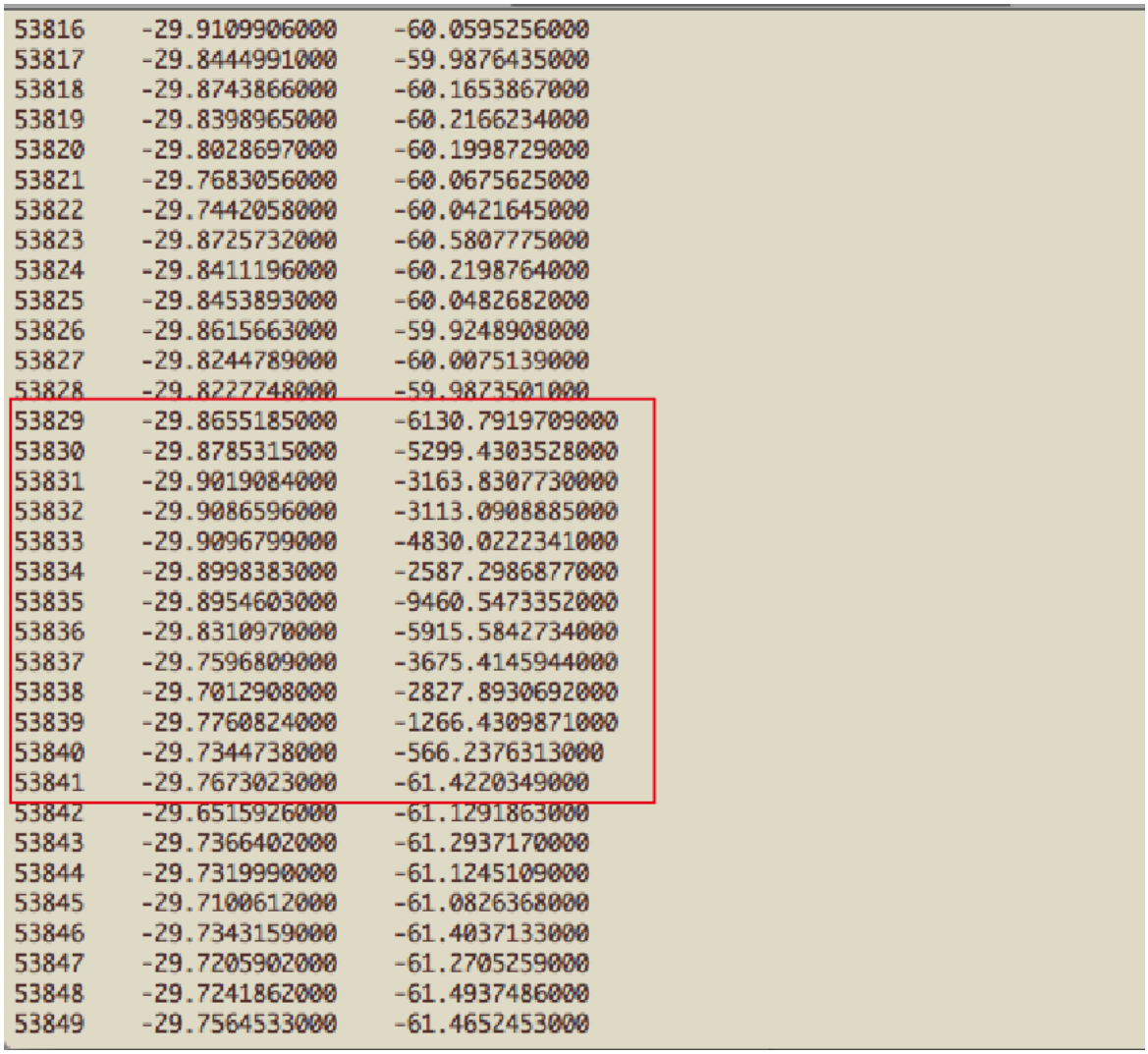


図 5.3: dis=0.004，λ=1.0 で Transition を用いてエネルギー計算した結果．横軸に試行回数 n，縦軸にエネルギーをとっている



```
53816    -29.9109906000    -60.0595256000
53817    -29.8444991000    -59.9876435000
53818    -29.8743866000    -60.1653867000
53819    -29.8398965000    -60.2166234000
53820    -29.8028697000    -60.1998729000
53821    -29.7683056000    -60.0675625000
53822    -29.7442058000    -60.0421645000
53823    -29.8725732000    -60.5807775000
53824    -29.8411196000    -60.2198764000
53825    -29.8453893000    -60.0482682000
53826    -29.8615663000    -59.9248908000
53827    -29.8244789000    -60.0075139000
53828    -29.8227748000    -59.9873501000
53829    -29.8655185000    -6130.7919709000
53830    -29.8785315000    -5299.4303528000
53831    -29.9019084000    -3163.8307730000
53832    -29.9086596000    -3113.0908885000
53833    -29.9096799000    -4830.0222341000
53834    -29.8998383000    -2587.2986877000
53835    -29.8954603000    -9460.5473352000
53836    -29.8310970000    -5915.5842734000
53837    -29.7596809000    -3675.4145944000
53838    -29.7012908000    -2827.8930692000
53839    -29.7760824000    -1266.4309871000
53840    -29.7344738000    -566.2376313000
53841    -29.7673023000    -61.4220349000
53842    -29.6515926000    -61.1291863000
53843    -29.7366402000    -61.2937170000
53844    -29.7319990000    -61.1245109000
53845    -29.7100612000    -61.0826368000
53846    -29.7343159000    -61.4037133000
53847    -29.7205902000    -61.2705259000
53848    -29.7241862000    -61.4937486000
53849    -29.7564533000    -61.4652453000
```

図 5.4: dis=0.004,λ=1.0 でモンテカルロを行った際の試行回数とエネルギー変化. 左から，試行回数，赤線は $u = \lambda E_{\mathrm{VASP}} + (1-\lambda) E_{\mathrm{Einstein}}$ ，$du/d\lambda = E_{\mathrm{VASP}} - E_{\mathrm{Einstein}}$ を表している.

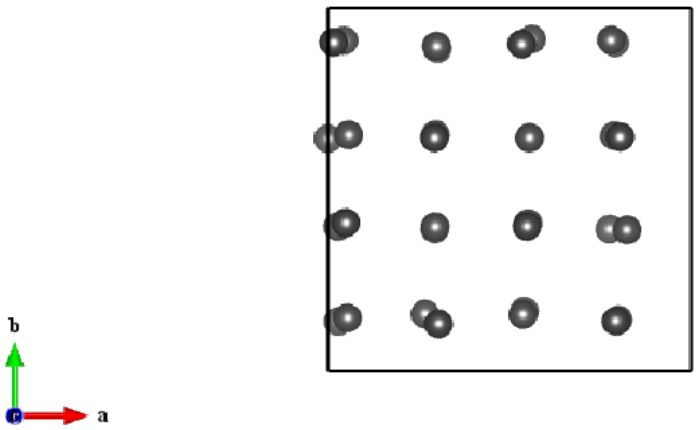


図 5.5: 試行回数 53828 回目の POSCAR を VESTA で可視化した様子.

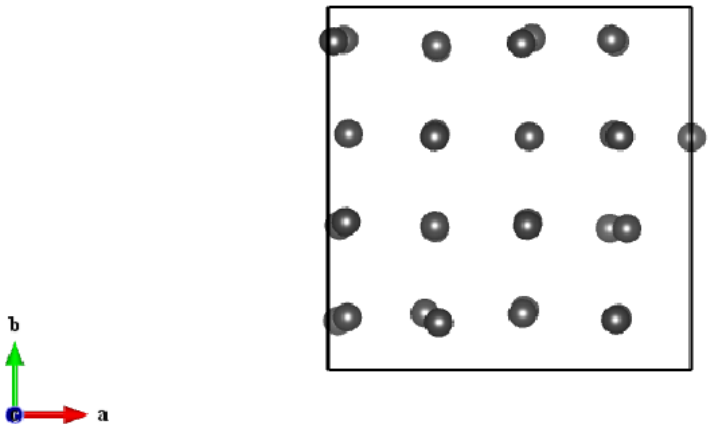


図 5.6: 試行回数 53829 回目の POSCAR を VESTA で可視化した様子．図 5.5 と比較すると，左上から 2 番目の原子が周期的境界条件により右へ移っている.

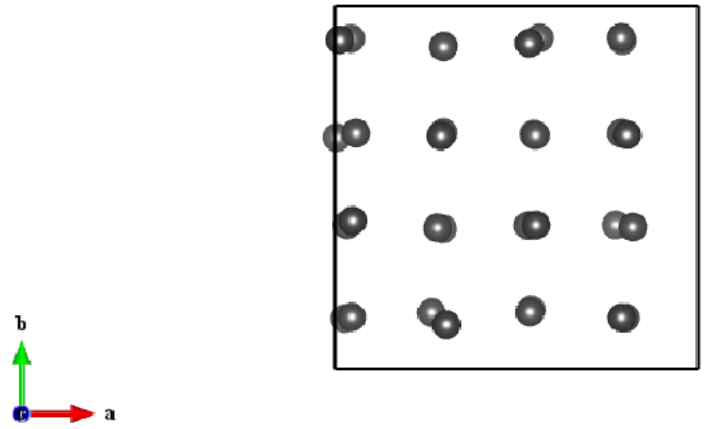


図 5.7: 試行回数 53841 回目の POSCAR を VESTA で可視化した様子．図 5.6 と比較すると，周期的境界条件により右へ移っていた原子が戻っていることが分かる．

第6章　総括

本研究は，第一原理計算を用いたモンテカルロ・シミュレーションのFrenkel法により非調和の影響を取り入れた自由エネルギーの計算を行った．

Frenkel法は，Einsteinモデルという理想状態のモデルを元に，理想状態のエネルギーから推移して現実状態のエネルギーを求める手法をとる．しかしながら，EInsteinモデルのエネルギーは系のrotationやdriftの影響からエネルギーの値が正しく計算できないという問題があった．そこで，系のrotationに制限(constraint)をかけることで前述の問題を解決し，理想状態のエネルギーを導き出し，そこから現実状態のエネルギーを導くことを試みた．

系のrotationにconstraintをかけた状態でモンテカルロ・シミュレーションを行った結果，エネルギーが熱平衡状態に達した．このことから系は，rotationにconstraintをかけた状態でも，正準集団を再現していると確認できた．また，理想状態のエネルギーを求めるEinsteinを開発し，モンテカルロ・シミュレーションで動作確認したところ，熱平衡状態に達したことから動作に問題がないものとした．

しかし，理想状態から推移して現実状態のエネルギーを求めるTransitionによるモンテカルロシミュレーションでは，値が求まらなかった．原因は，Einsteinの計算に周期的境界条件を考慮していなかったこと，また，rotationを抑えただけでdriftを抑えられるした仮定が誤っていたためと考えられる．

よって，今後VASPを用いた現実系の計算にFrenkel法のソフトを提供するには，これらの不具合を修正する必要がある．

関連図書

[1] K. Parlinski, Z. Q. Li and Y. Kawazoe, Phys. Rev. Let., 78, 4063-4066 (1997).

[2] D. Frenkel and A. J. C. Ladd, J. Chem. Phys., 81, 3188-3193(1984).

[3] 伊藤欣也，京都大学，工学部，材料工学科，卒業論文 2001.

[4] 西谷滋人著,「固体物理の基礎」(森北出版,2006) pp.109-115.

[5] キッテル著，宇野良清 他 訳,「固体物理学入門」(丸善 1978)

[6] 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.

[7] 京口周平著,「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 2012 年度卒業論文.

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　プログラム

本研究において作成したプログラムを記載する.

A.0.1　mc_Transition.rb

このコードが本研究で使用したプログラムのメインルーチンとなる．パラメータとして温度T，試行回数n，原子間の距離比dis，現実のエネルギーと理想のエネルギーの割合を示すλを入力すると，計算結果がディレクトリ/result以下のファイルに出力される.

```
# -*- coding: utf-8 -*-
include Math
require "pp"
require "./initial_cPOSCAR.rb"

puts "T="
T=gets.chomp.to_f

puts "n="
n_max=gets.chomp.to_i

puts "Dis="
d_max=gets.chomp.to_f

puts "Lambda="
$lambda=gets.chomp.to_f

$n_multi=1

start = Time.now #計測開始

dEdl = []


def get_str_OUTCAR(x,str) #OUTCARから文字列を読み取る
outcar=[]
iter=0
number=0
        energyCalc=str
   File.open("OUTCAR_#{energyCalc}").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(str) #OUTCARからエネルギーを取り出す
```

```
outcar=get_str_OUTCAR("ION-ELECTRON",str)
return outcar[1].split(" ")[4].to_f
end

def Energy(str) #POSCAR からエネルギー計算
energyCalc=str
system("rm -rf OUTCAR_#{energyCalc}")
system("ruby #{energyCalc}.rb POSCAR > OUTCAR_#{energyCalc}")
return get_energy_OUTCAR(energyCalc)
end

def energy
        ev=Energy("pseudoVASP")
        ee=Energy("Einstein")
        return [$lambda*ev+(1-$lambda)*ee,ev,ee]
end

set_newPOSCAR(d_max)

energy_before = energy()

n_max.times do |i|

if i\%(n_max/10)==0 then printf(".") end
system("cp -f POSCAR sub_POSCAR")

make_newPOSCAR()
make_cancelingPOSCAR()  #canceling_POSCAR 作成

system("cp -f canceling_POSCAR POSCAR")
        energy_new = energy()
dE = energy_new[0] - energy_before[0] #新たな配置の受け入れを計算

        if rand()<exp(-dE/T) then
                dEdl << energy_new[1]-energy_new[2]
                energy_before = energy_new
        else
                system("cp -f sub_POSCAR POSCAR")
        end
end


printf("\n 採択率:\%10.5f\n",dEdl.length/n_max.to_f)

File.open("./result/C_transition" + T.to_s + "_" + n_max.to_s + "_" + d_max.to_s + "_" + $lambda.to_s + ".txt","w"){|fil
dEdl.each do |tmp|
file.puts(tmp)
end
}

File.open("./result/C_transition_result_" + T.to_s + "_" + n_max.to_s + "_" + d_max.to_s + "_" + $lambda.to_s + ".txt","
file.puts(dEdl.length/n_max.to_f)
}

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

#$t0=Time.now

#system("gnuplot command_nvtMC_plot.gpl")
```

A.0.2 POSCARの編集

入力されたオリジナルのAlのPOSCARを編集するプログラム.

initial_cPOSCAR.rb

```
# -*- coding: utf-8 -*
include Math
require './analysis2.rb'
require './analysis3.rb'
require './jordanpivot.rb'
require "pp"

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

end

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

```
def periodic_mv(pos,dd)
for i in 0..2
pos[i]=pos[i].to_f
pos[i]+=dd[i] #新しい原子座標
if pos[i] > 1.0 then
pos[i]-=1.0
elsif pos[i] < 0.0 then
pos[i]+=1.0
end
end
end

def write_new_POSCAR(ajs,poss,text)
poss.each_with_index do |ele,i|
text[ajs[i]]=sprintf("%16.9f %19.9f %19.9f\n",ele[0],ele[1],ele[2])#aj を text に移す
end
n_P = File.open("new_POSCAR","w")#new_POSCAR 新規作成
      text.each do |ele|
            n_P << ele
      end
n_P.close
end

def make_newPOSCAR
ajs,poss,dds = [],[],[]
text = read_POSCAR()
32.times {|i|
aj=i+1+6 #原子番号決め
ajs << aj

pos=text[aj].split(nil)
dd=[]
3.times { dd<<(2.0*rand()-1.0)*$init_vals[0] }
dds << dd

periodic_mv(pos,dd)
poss<<[pos[0],pos[1],pos[2]]
}
write_new_POSCAR(ajs,poss,text)
system("cp new_POSCAR POSCAR")

system("cp POSCAR first_POSCAR")


end

def CrossProduct(a,b)
c=[0.0, 0.0, 0.0]

c[0]=a[1]*b[2]-a[2]*b[1]
c[1]=a[2]*b[0]-a[0]*b[2]
c[2]=a[0]*b[1]-a[1]*b[0]

return c
end

def rewrite_cancelingPOSCAR(chose,tmp)

text = read_POSCAR()
ajs,poss=[],[]
cnt=0

  32.times {|i|
aj=i+1+6
pos=text[aj].split(nil)
ajs << aj
```

```
if i==chose[0] then
pos[0]=tmp[0]
elsif i==chose[1] then
pos[1]=tmp[1]
elsif i==chose[2] then
pos[2]=tmp[2]
end

poss<<[pos[0],pos[1],pos[2]]
}

poss.each_with_index do |ele,i|
text[ajs[i]]=sprintf("\%16.9f \%19.9f \%19.9f\n",ele[0],ele[1],ele[2])
end

n_P = File.open("canceling_POSCAR","w")#canceling_POSCAR 新規作成
        text.each do|ele|
                n_P << ele
        end
n_P.close
end

def make_cancelingPOSCAR()

#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=max

center,s_rxf,poss,dds=getShiftMoment("POSCAR","Al_POSCAR")

#shift_POSCAR(center)
#center,s_rxf,poss,dds=getShiftMoment("POSCAR","Al_POSCAR")

xyz=nil

while xyz==nil do

ai=rand(32)
while true
temp=rand(32)
if temp!=ai
aj=temp
break
end
end

while true
temp=rand(32)
if temp!=ai&&temp!=aj
ak=temp
break
end
end

posi=poss[ai]
ddi=dds[ai]
posj=poss[aj]
ddj=dds[aj]
posk=poss[ak]
ddk=dds[ak]

cc=[0.5, 0.5, 0.5]

ddii=[1, ddi[1], ddi[2]]
ddjj=[ddj[0], 1, ddj[2]]
ddkk=[ddk[0], ddk[1], 1]
```

```
posi_c=[posi[0]-cc[0], posi[1]-cc[1], posi[2]-cc[2]]
posj_c=[posj[0]-cc[0], posj[1]-cc[1], posj[2]-cc[2]]
posk_c=[posk[0]-cc[0], posk[1]-cc[1], posk[2]-cc[2]]

# p posi_c
# p posj_c
# p posk_c

cp_posidd=CrossProduct(posi_c, ddi)
cp_posjdd=CrossProduct(posj_c, ddj)
cp_poskdd=CrossProduct(posk_c, ddk)

# p cp_posidd
# p cp_posjdd
# p cp_poskdd

ci=s_rxf[0]+posj_c[1]*ddjj[2]-cp_posjdd[0]-posk_c[2]*ddkk[1]-cp_poskdd[0]
cj=s_rxf[1]-posi_c[0]*ddii[2]-cp_posidd[1]+posk_c[2]*ddkk[0]-cp_poskdd[1]
ck=s_rxf[2]+posi_c[0]*ddii[1]-cp_posidd[2]-posj_c[1]*ddjj[0]-cp_posjdd[2]
cdxi = 0
cdyi = -posj_c[2]
cdzi = posk_c[1]

cdxj = posi_c[2]
  cdyj = 0
cdzj = -posk_c[0]

cdxk = -posi_c[1]
cdyk = posj_c[0]
cdzk = 0

xyz = gaussjordanpivot([[cdxi, cdyi, cdzi, -ci], [cdxj, cdyj, cdzj, -cj], [cdxk, cdyk, cdzk, -ck]])
#p "!xyz=", xyz

end
xyz = [sprintf("%19.8f",xyz[0]).to_f, sprintf("%19.8f",xyz[1]).to_f, sprintf("%19.8f",xyz[2]).to_f]

dx_tmp=posi[0]+xyz[0]
dy_tmp=posj[1]+xyz[1]
dz_tmp=posk[2]+xyz[2]

chose=[ai,aj,ak]
tmp=[dx_tmp, dy_tmp, dz_tmp]

rewrite_cancelingPOSCAR(chose, tmp)
analysisMoment("canceling_POSCAR","Al_POSCAR")

end
```

A.0.3 エネルギー計算部

エネルギーの計算には西谷研の中井が開発したEAMポテンシャルのエネルギーを出力するpseudoVASP，及び，理想状態のエネルギーを出力するEinsteinを使用した.

pseudoVASP.rb

pseudoVASPのプログラム.

```
# -*- coding: utf-8 -*-
include Math
require 'pp'

$m = 1.000
$lattice=[[0.0,0.0,0.0],[0.0,0.0,0.0],[0.0,0.0,0.0]]
$whole_scale
$cutoff = 4.0* 0.8

class Atom
  attr_accessor :pos, :nl

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

  def energy()
   return lj_energy() #Lennard-Jones
  end

  def force()
    return lj_force()
  end

  def reset_nl()
    @nl=[]
  end


  A0=1.587401051*0.7071067812/2.857701314;
  #E0=-1*4.0/12.0*3.39
  E0=-1*4.0/12.0

  def lj_energy()#LJP
    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

  def lj_force()
    f=[0.0,0.0,0.0]
    nl.each do |j|
      x,y,z=f_distance(@pos,$atom_list[j].pos)
      r=x**2+y**2+z**2
      dedr=-E0*(12/(A0**12*(r)**7)-6/(A0**6*(r)**4))
      f[0] += x*dedr
      f[1] += y*dedr
      f[2] += z*dedr
    end
    3.times do |i|  printf("\%10.6f",pos[i]/$lattice[i][i]) end
    3.times do |i|  printf("\%10.6f",f[i]) end
    printf("\n")
    return f
  end
end

def makeLattice()
  file = open(ARGV[0])
  atom_list=[]
  sw=false
  iline=0
  while line = file.gets do
```

```
    iline+=1
    if iline==2 then
      $whole_scale=line.chomp.to_f
    end
    if (iline==3 or iline==4 or iline==5) then

      lat0=line.chomp.split(" ")
      lat0.each.with_index do |vec,i|
        $lattice[iline-3][i] = vec.to_f
      end
    end

    if /Direct/ =? line then
      sw=true
    elsif sw then
      pos0=line.chomp.split(" ")
      pos1=[]
      pos0.each do |pos2|
       pos1 << pos2.to_f*$lattice[0][0]*$whole_scale
#        pos1 << pos2.to_f
      end

      atom_list << Atom.new(pos1)
    end
  end
  file.close
  p $lattice
  p $whole_scale
  return atom_list
end

$L=[2,2,2]

def distance(a,b)
  r=0.0
  for i in 0..2 do
    x=a[i]-b[i]
    x=x-(x/$lattice[i][i]).round*$lattice[i][i]
    r+=x*x
  end
  return sqrt(r)
end

def f_distance(a,b)
  t=[]
  for i in 0..2 do
    x=a[i]-b[i]
    x=x-(x/$lattice[i][i]).round*$lattice[i][i]
    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

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=$atom_list[j]
      if distance(ai.pos,aj.pos)<$cutoff then
        ai.nl << j
        aj.nl << i
      end
    end
  end
end

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

def force_all()
  $atom_index.each do |i|
    $atom_list[i].force()
  end
  return
end

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\n",total_E())
printf("POSITION                                        TOTAL-FORCE (eV/Angst)\n")
printf("-----------------------------------------------------------------\n")
force_all()
printf("Total CPU time used:      \%10.5f\n",Time.now-t0)
```

Einstein.rb

このプログラムで，Einstein のエネルギーの計算を行う.

```
# -*- coding: utf-8 -*-
include Math
require 'pp'

$m = 1.000
$lattice=[[0.0,0.0,0.0],[0.0,0.0,0.0],[0.0,0.0,0.0]]
$whole_scale
$cutoff = 4.0* 0.8

class Atom
attr_accessor :pos, :nl, :num

def initialize(pos,i)
                @num=i
@nl=[]
@pos=Array.new(pos)
end

def energy()
return einstein_energy() #Einstein
end
```

```
def force()
return lj_force()
end

def reset_nl()
@nl=[]
end

A0=1.587401051*0.7071067812/2.857701314;
#E0=-1*4.0/12.0*3.39
E0=-1*4.0/12.0

        KK=8*44.06

def einstein_energy()
                dx=@pos[0]-$atom_list0[@num].pos[0]
                dy=@pos[1]-$atom_list0[@num].pos[1]
                dz=@pos[2]-$atom_list0[@num].pos[2]
dd2=dx*dx+dy*dy+dz*dz
                return ei=1/2.0*KK*dd2-1.0
end

def lj_force()
f=[0.0,0.0,0.0]
3.times do |i|  printf("\%10.6f",pos[i]/$lattice[i][i]) end
3.times do |i|  printf("\%10.6f",f[i]) end
printf("\n")
return f
end

end

def makeLattice(file_name)
file = open(file_name)
atom_list=[]
sw=false
iline=0
        iatom=0
while line = file.gets do
iline+=1
if iline==2 then
$whole_scale=line.chomp.to_f
end
if (iline==3 or iline==4 or iline==5) then
lat0=line.chomp.split(" ")
lat0.each.with_index do |vec,i|
$lattice[iline-3][i] = vec.to_f
end
end

if /Direct/ =? line then
sw=true
elsif sw then
pos0=line.chomp.split(" ")
pos1=[]
pos0.each do |pos2|
pos1 << pos2.to_f
end
atom_list << Atom.new(pos1,iatom)
                        iatom+=1
end
end
file.close
p $lattice
p $whole_scale
return atom_list
```

```
end

$L=[2,2,2]

def distance(a,b)
r=0.0
for i in 0..2 do
x=a[i]-b[i]
x=x-(x/$lattice[i][i]).round*$lattice[i][i]
r+=x*x
end
return sqrt(r)
end

def f_distance(a,b)
t=[]
for i in 0..2 do
x=a[i]-b[i]
x=x-(x/$lattice[i][i]).round*$lattice[i][i]
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

def makeNL()
        return
end

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

end
return total_E
end

def force_all()
$atom_index.each do |i|
$atom_list[i].force()
end
return
end

t0=Time.now
$atom_list=makeLattice(ARGV[0])
$atom_list0=makeLattice('Al_POSCAR')
$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\n",total_E())
printf("POSITION                                       TOTAL-FORCE (eV/Angst)\n")
printf("---------------------------------------------------------------\n")
force_all()
printf("Total CPU time used:      \%10.5f\n",Time.now-t0)
```

constraint.mw

系に constraint をかける三元一次方程式の解を求める Maple のプログラム.

```
>restart;
>cc:=Vector([0.5,0.5,0.5]):
>rxf:=Vector([   0.0004089012,   0.0000018379,  -0.0001244011]):
>p1:=Vector([   0.8750000000,   0.1250000000,   0.3750000000]):
>f10:=Vector([  -0.0000053570,  -0.0001813910,  -0.0000237270]):
>f1:=Vector([dx,  -0.0001813910,  -0.0000237270]):
>p2:=Vector([   0.1250000000,   0.8750000000,   0.8750000000]):
>f20:=Vector([   0.0001444470,  -0.0000486220,  -0.0000577100]):
>f2:=Vector([   0.0001444470,dy,  -0.0000577100]):
>p3:=Vector([   0.1250000000,   0.1250000000,   0.1250000000]):
>f30:=Vector([   0.0000235240,   0.0001876580,   0.0000114410]):
>f3:=Vector([   0.0000235240,   0.0001876580,dz]):

>with(LinearAlgebra):
>dd:=Vector([dx,dy,dz]);


>pf1:=CrossProduct(p1-cc,f1)-CrossProduct(p1-cc,f10);
>pf2:=CrossProduct(p2-cc,f2)-CrossProduct(p2-cc,f20);
>pf3:=CrossProduct(p3-cc,f3)-CrossProduct(p3-cc,f30);

>eq1:={seq(pf1[i]+pf2[i]+pf3[i]+rxf[i]=0,i=1..3)};

>s1:=solve(eq1,{dx,dy,dz});
{dx = 0.001064923400, dy = 0.0006899221333, dz = 0.0003633000667}
```