モンテカルロ・シミュレーションによる
高分子鎖のコンホメーションと張力・伸長関係

目次

9.1　目的
9.2　モンテカルロ法
9.3　高分子鎖のモンテカルロ・シミュレーション
9.4　計算機実験のための C プログラミング入門
9.5　ライブラリ lce(Library for Computer Experiments) の使い方
9.6　モンテカルロ・シミュレーション用のプログラム作成
付録 9.A　乱数
付録 9.B　GNUPLOT の使い方
付録 9.C　分布を求めるプログラム

9.1　目的

本節では代表的な分子シミュレーション法の一つであるモンテカルロ (MC) 法について学び，MC 法を用いて高分子鎖のコンホメーションと張力・伸長関係についての考察を行う．特に本実験では，単に与えられたプログラムを実行し，結果を整理するのではなく，C 言語によるプログラミング技法を習得し，シミュレーションのソースコードを各自が作成する．これを用いて課題の計算を行い，必要に応じて解析プログラムを自ら作成し，考察を行う．このような能動的な実習により，高分子鎖の弾性的性質とシミュレーション技法のより深い理解を目指す．

9.2　モンテカルロ法

9.2.1　モンテカルロ法の特徴

計算機シミュレーションは，近年の計算機の演算能力などの飛躍的発達に伴って，高分子系の研究においても有用な研究手法として幅広く用いられるようになってきた．高分子系の場合には，幅広い時空間スケールにわたって階層的に高次構造形成が起こり，これにより多彩な物性・機能が生み出されているので，高分子系に適用される計算手法も非常に多岐に渡っている．その中でも，モンテカルロ (Monte Carlo；MC) 法や分子動力学 (molecular dynamics；MD) 法に代表される分子シミュレーション法は，分子レベルから現象を理解するための手法として幅広い分野で用いられている．

MD 法は，分子の従うニュートンの運動方程式を数値積分することで，系の時間発展を計算する手法であり，平衡状態だけでなく，非平衡状態における系の性質も議論できるという利点がある．これに対して MC 法では，ある特定のアンサンブル (統計集団) に対応する状態を一定の遷移確率の下に次々に発生させ，物理量のアンサンブル平均などを計算する手法である．MD 法では決定論的な方程式により系の振る舞いが決まるのに対して，MC 法では状態間の遷移は確率的に決定されるので，確率論的手法ということができる．MC 法の利点の一つは，通常良く用いられるカノニカル・アンサンブルだけでなく，グランドカノニカル・アンサンブルのように分子数の増減を伴うようなものも含めて様々なアンサンブルを比較的容易に実現できる点である．また，MC 法では，平均操作を効率的に行うために，分子に実際の運動と異なる変位を与えたり，系が自由エネルギー空間の局所極小にトラップされないように工夫されたアンサンブルを用いて計算を行うこともできる．このような特徴をうまく利用すると計算効率を劇的に向上させることも可能であり，MC 法はタンパク質のような複雑な分子の高次構造形成の問題などにも適用されている．

9.2.2　サンプリング

統計力学の問題にモンテカルロ法がどのように適用されるのかをみるために，ここでは，体積 V，粒子 N 個を持つ系が温度 T の熱浴と接触している場合，すなわちカノニカル集団を考えよう [1]．カノニカル集団では粒子の位置 $\boldsymbol{r}_1, \boldsymbol{r}_2, \cdots, \boldsymbol{r}_N$ (まとめて $\{\boldsymbol{r}\}$ と表すことにする) のみに依存する物理量 $A(\{\boldsymbol{r}\})$ の集団平均は

$$\langle A \rangle = \frac{1}{Q} \int d\{\boldsymbol{r}\} A(\{\boldsymbol{r}\}) e^{-\beta U(\{\boldsymbol{r}\})} \tag{9.2.1}$$

と書ける．ここで，U は粒子間に働く相互作用エネルギーであり，$\beta = 1/k_{\mathrm{B}}T$ で，k_{B} はボルツマン定数である．また，Q は配置積分

$$Q = \int d\{\boldsymbol{r}\} e^{-\beta U(\{\boldsymbol{r}\})} \tag{9.2.2}$$

である．この $3N$ 次元空間での配置積分を，状態をランダムに出現させてサンプリングを行う単純サンプリングで評価しようとすれば，n 番目の配置を $\{\boldsymbol{r}^n\}$ として

$$\langle A \rangle = \frac{\sum_{n=1}^{M} A(\{\boldsymbol{r}^n\}) e^{-\beta U(\{\boldsymbol{r}^n\})}}{\sum_{n=1}^{M} e^{-\beta U(\{\boldsymbol{r}^n\})}} \tag{9.2.3}$$

と書ける．ここで，M はサンプリングの総数である．しかし，これは一般に非常に効率が悪いことが知られている[1]．そこで，積分 (9.2.1) の積分関数に任意の重み関数 $w(\{\boldsymbol{r}\})$ をかけて積分すること

$$\langle A \rangle = \frac{1}{Q} \int \frac{A(\{\boldsymbol{r}\})}{w(\{\boldsymbol{r}\})} e^{-\beta U(\{\boldsymbol{r}\})} w(\{\boldsymbol{r}\}) d\{\boldsymbol{r}\} \tag{9.2.4}$$

を考える．カノニカル集団の場合には，重み関数としてカノニカル分布

$$w(\{\boldsymbol{r}\}) = \frac{1}{Q} e^{-\beta U(\{\boldsymbol{r}\})} \tag{9.2.5}$$

を使うことになる．このとき式 (9.2.4) は

$$\langle A \rangle = \int A(\{\boldsymbol{r}\}) w(\{\boldsymbol{r}\}) d\{\boldsymbol{r}\} \tag{9.2.6}$$

と書ける．サンプリングを行う際には，状態の出現確率が $w(\{\boldsymbol{r}\})$ となるように状態を生成する必要がある．この場合には A の平均は

$$\langle A \rangle = \frac{\sum_{n=1}^{M} A(\{\boldsymbol{r}^n\})}{M} \tag{9.2.7}$$

と簡単な形に書ける．この方法を実際に使う場合には，どのようにしてカノニカル分布に従う状態を次々に生成するのかが問題になる．また式 (9.2.5) の $w(\{\boldsymbol{r}\})$ には配置積分 Q が含まれているが，Q は前もって分からないので，状態を生成する手続きの中でこれを使用することはできない．このような条件を満たす手続きを実現する方法が次に述べるメトロポリス法である．

9.2.3　メトロポリス法

メトロポリス法ではマルコフ過程に従ってサンプルが生成される．マルコフ過程では次のステップの状態が現在の状態のみで決まるので，時刻 t における状態 $\{\boldsymbol{r}\}$ の確率分布関数 $P(\{\boldsymbol{r}\}, t)$ の時間発展方程式は次

[1] サンプリングの実例は 9.6.1 節で実習する

のように書くことができる：

$$\frac{\partial}{\partial t}P(\{\boldsymbol{r}\},t)=\sum_{\{\boldsymbol{r}'\}}W(\{\boldsymbol{r}\}\leftarrow\{\boldsymbol{r}'\})P(\{\boldsymbol{r}'\},t)-\sum_{\{\boldsymbol{r}'\}}W(\{\boldsymbol{r}'\}\leftarrow\{\boldsymbol{r}\})P(\{\boldsymbol{r}\},t) \tag{9.2.8}$$

ここで，$W(\{\boldsymbol{r}\}\leftarrow\{\boldsymbol{r}'\})$ は状態 $\{\boldsymbol{r}'\}$ から状態 $\{\boldsymbol{r}\}$ への遷移確率である．時間が十分に経った後の平衡状態では，平衡分布 $P_{\rm eq}(\{\boldsymbol{r}\})$ に対して

$$\sum_{\{\boldsymbol{r}'\}}W(\{\boldsymbol{r}\}\leftarrow\{\boldsymbol{r}'\})P_{\rm eq}(\{\boldsymbol{r}'\})=\sum_{\{\boldsymbol{r}'\}}W(\{\boldsymbol{r}'\}\leftarrow\{\boldsymbol{r}\})P_{\rm eq}(\{\boldsymbol{r}\}) \tag{9.2.9}$$

が成立する．カノニカル集団に対しては，$P_{\rm eq}$ がカノニカル分布になるように遷移確率を式 (9.2.9) から決めれば良いが，このままでは自由度が多すぎて W が一意的に決まらない．そこで，$\{\boldsymbol{r}'\}$ の和をとる前の 2 つの状態 $\{\boldsymbol{r}\}$，$\{\boldsymbol{r}'\}$ について個別に条件

$$W(\{\boldsymbol{r}\}\leftarrow\{\boldsymbol{r}'\})P_{\rm eq}(\{\boldsymbol{r}'\})=W(\{\boldsymbol{r}'\}\leftarrow\{\boldsymbol{r}\})P_{\rm eq}(\{\boldsymbol{r}\}) \tag{9.2.10}$$

が成立しているものとして W を決める．これは微視的な状態間の遷移の可逆性を示しており,「詳細釣り合い条件」と呼ばれる．しかし，詳細釣り合い条件を満たす W の選び方にはまだかなりの自由度が残っているので，さらに

$$W(\{\boldsymbol{r}'\}\leftarrow\{\boldsymbol{r}\}) = \begin{cases}\alpha(\{\boldsymbol{r}\},\{\boldsymbol{r}\}') & \Delta U\le 0\\ \alpha(\{\boldsymbol{r}\},\{\boldsymbol{r}\}')e^{-\beta\Delta U} & \Delta U>0\end{cases} \tag{9.2.11}$$

$$W(\{\boldsymbol{r}\}\leftarrow\{\boldsymbol{r}\}) = 1-\sum_{\{\boldsymbol{r}'\}(\neq\{\boldsymbol{r}\})}W(\{\boldsymbol{r}'\}\leftarrow\{\boldsymbol{r}\}) \tag{9.2.12}$$

と選ぶことにする．ここで $\Delta U=U(\{\boldsymbol{r}'\})-U(\{\boldsymbol{r}\})$ である．また，α は次の条件を満たさなければならない．

$$\begin{aligned}\alpha(\{\boldsymbol{r}\},\{\boldsymbol{r}'\}) &= \alpha(\{\boldsymbol{r}'\},\{\boldsymbol{r}\}) \quad \text{(対称性)}\\ \alpha(\{\boldsymbol{r}\},\{\boldsymbol{r}'\}) &\ge 0 \quad \text{(定正値性)}\end{aligned} \tag{9.2.13}$$

この $\alpha(\{\boldsymbol{r}\},\{\boldsymbol{r}'\})$ は状態 $\{\boldsymbol{r}'\}$ が状態 $\{\boldsymbol{r}\}$ からどのように生成されるかを決めるものであり，対称性と定正値性だけではその選び方に多くの自由度がある．通常用いられている方法を図 9.1 に示している．この図では状態 $\{\boldsymbol{r}\}$ にある粒子 i が $2\delta r_{\rm max}$ を一辺とする正方形（三次元では立方体）の領域 $\mathcal{R}$ 内の一点に等確率で変位することにより，状態 $\{\boldsymbol{r}'\}$ が生成されている．このとき $\alpha(\{\boldsymbol{r}\},\{\boldsymbol{r}\}')$ は $\delta\boldsymbol{r}_{\rm max}$ の関数であるが，実際のアルゴリズムでは，$\alpha(\{\boldsymbol{r}\},\{\boldsymbol{r}\}')$ の具体的な形は必要はなく，以下に述べる試行的な配置の採択率が適当な値になるように $\delta r_{\rm max}$ をパラメータとして調節する．

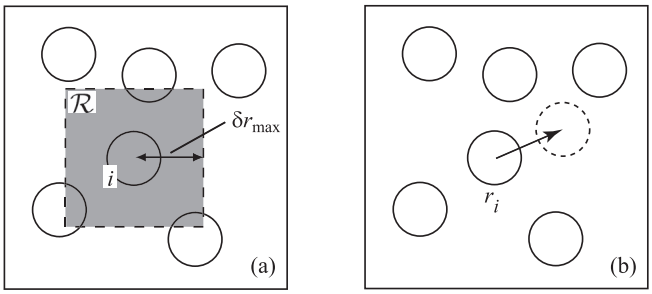

図 9.1: α の決め方．(a) 状態 $\{\boldsymbol{r}\}$ にある粒子 i を領域 $\mathcal{R}$ 内へ変位させることにより状態 $\{\boldsymbol{r}\}'$ が生成される．(b) 粒子 i の変位．

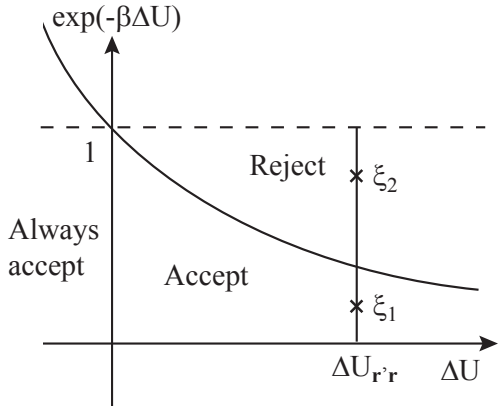

図 9.2: モンテカルロ法における判定.

以上をまとめるとカノニカル集団に対するメトロポリス法のアルゴリズムの主要部分は次のようになる．まず系は状態 $\{\boldsymbol{r}\}$ にあるとする．

1. ランダムに粒子を一つ選ぶ.
2. 粒子を領域 $\mathcal{R}$ 内の位置にランダムに変位させ，試行的な配置 $\{\boldsymbol{r}'\}$ を生成する (図 9.1(b)).
3. 配置 $\{\boldsymbol{r}'\}$ での系のポテンシャルエネルギー $U(\{\boldsymbol{r}'\})$ を計算し，$\Delta U = U(\{\boldsymbol{r}'\}) - U(\{\boldsymbol{r}\})$ を計算する.
4. $\Delta U \leq 0$ であれば，配置 $\{\boldsymbol{r}'\}$ を次の新しい配置として採用し，ステップ 1 から繰り返す.
5. $\Delta U > 0$ であれば，$[0,1)$ 区間の乱数 ξ を生成し
   (a) $e^{-\beta\Delta U} > \xi$ ならば，配置 $\{\boldsymbol{r}'\}$ を次の新しい配置として採用し，ステップ 1 から繰り返す.
   (b) $e^{-\beta\Delta U} \leq \xi$ ならば，配置 $\{\boldsymbol{r}'\}$ を棄却し，変位前の配置 $\{\boldsymbol{r}\}$ を次の配置として採用しステップ 1 から繰り返す.

ステップ 4，5 の判定の様子を図 9.2 に示している．基本は，ボルツマン因子 $e^{-\beta\Delta U}$ と $[0,1)$ 区間の値をとる乱数 ξ を比較して，

$$e^{-\beta\Delta U} > \xi \tag{9.2.14}$$

ならば配置 $\{\boldsymbol{r}'\}$ を新しい状態として採用する．$\Delta U \leq 0$ の時には必ず式 (9.2.14) を満たすので，乱数を生成したり，指数関数を計算したりする手間を省いて，配置 $\{\boldsymbol{r}'\}$ を採用することができる．これがステップ 4 である．$\Delta U > 0$ の場合には，乱数を一つ生成し，発生した乱数が図 9.2 に示された ξ_1 のように $e^{-\beta\Delta U} > \xi_1$ の時には状態 $\{\boldsymbol{r}'\}$ を新しい配置として採用し，逆に ξ_2 のように $e^{-\beta\Delta U} \leq \xi_2$ の場合には配置 $\{\boldsymbol{r}'\}$ は棄却される．これがステップ 5 である．

モンテカルロ法では上記の一つの粒子に対する試行 N 回 (N は系内の全粒子数) を 1 モンテカルロステップ (MCS) とし，試行の回数を表現することが多い.

9.3 高分子鎖のモンテカルロ・シミュレーション

9.3.1 高分子鎖の統計的性質

まず，高分子鎖の統計力学的取り扱いについて考える．ここでは温度 T の熱浴中にある一本の高分子鎖を考える．鎖は $N+1$ 個の統計単位 (セグメント) から成り，i 番目のセグメントの位置ベクトルを $\boldsymbol{r}_i$ とす

る．熱平衡状態にあるこの高分子鎖の統計的性質を考察するために，末端ベクトル $\boldsymbol{R}$ を一定に保った場合のカノニカル分配関数を考える (図 9.3)．これは

$$Z(\boldsymbol{R},T)=\int\cdots\int d\boldsymbol{r}_1 d\boldsymbol{r}_2\cdots d\boldsymbol{r}_N e^{-\beta U(\{\mathbf{r}\})} \tag{9.3.1}$$

で与えられる．ここで，U はセグメント間に働く相互作用を表すポテンシャルエネルギーである．計算機シミュレーションでは U として

$$U=U_{\rm bond}(l)+U_{\rm bend}(\theta)+U_{\rm tortion}(\phi)+U_{\rm non-bond}(r_{ij}) \tag{9.3.2}$$

という形がよく用いられる．ここでは相互作用ポテンシャルの具体的な関数形には言及しないが，ポテンシャルはそれぞれ隣接するセグメント間の結合距離 l，連続する 3 個の原子の作る角 θ，二面角 ϕ，高分子鎖に沿って隣接していないセグメント i,j 間の距離 r_{ij} の関数となっている．また，高分子電解質の場合には，静電相互作用の効果も取り入れる必要がある．

さて，分配関数 Z を求めることができれば，高分子鎖の慣性半径などの物理量を計算することができる．実際，理想鎖の場合にはボンドの結合相互作用のみを考えれば良いので，計算を厳密に実行でき，様々な物理量を具体的に計算できる（詳しくは参考文献 [1, 2] 参照）．しかし，一般の場合にはポテンシャル U は分子間の相互作用を全て含むので実際に式 (9.3.1) の積分を厳密に実行するのは困難である．以下では，MC 法を用いて高分子鎖のコンホメーションや弾性的性質を計算してみよう．

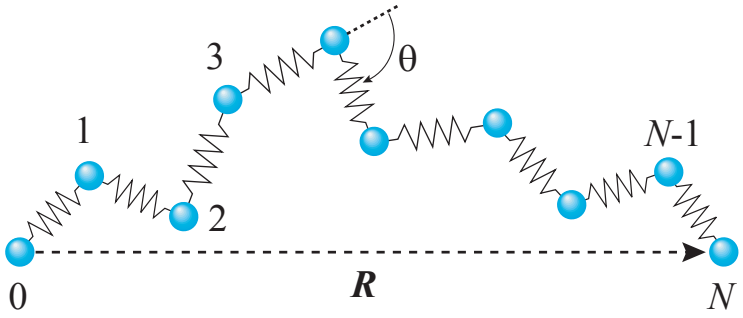

図 9.3: 高分子鎖のモデルの模式図.

9.3.2　モンテカルロ法による高分子のコンホメーション

最初に，末端ベクトル $\boldsymbol{R}$ を固定しない場合の高分子のコンホメーションを特徴付ける物理量を MC 法により計算する．9.2 節での MC 法の説明では粒子を扱ったが，以下ではこれを高分子鎖を構成するセグメントに対応させる．シミュレーションでは前述した全ての種類の相互作用を考慮することができるが，本節では簡単のために理想鎖を取り扱い，結合バネのポテンシャルのみを考える．バネのポテンシャルとしては次の有限の伸びきり長を持つバネポテンシャル (finitely extensible nonlinear elastic potential，FENE ポテンシャルと略す) を用いる．

$$U_{\rm bond}(l_i)=-\frac{1}{2}k_{\rm bond}\left(l_{\rm max}-a\right)^2\ln\left[1-\left(\frac{l_i-a}{l_{\rm max}-a}\right)^2\right] \tag{9.3.3}$$

ここで，$k_{\rm bond}$ はバネ定数，$l_i\equiv|\boldsymbol{r}_i-\boldsymbol{r}_{i-1}|$ はボンド i のボンド長，$l_{\rm max}$ はボンドの伸び切り長，a は平衡ボンド長である．シミュレーションではエネルギーは $k_{\rm B}T$，長さは a を単位として測ることにする．バネ定数，伸び切り長は $\beta k_{\rm bond}a^2=50$，$l_{\rm max}=1.2a$ の値に固定する[2]．

[2]バネでセグメントを結んだこのようなモデルはよくバネ・ビーズモデルと呼ばれる．以下の解説でもセグメントをビーズと呼ぶことがある．

次の手順で計算，考察を行う：

〈 **実習 1** 〉

1. 以下の 9.6 節の解説に従ってプログラム `polymer_free.c` を作成する．
2. 平均 2 乗末端間距離 $\langle R^2 \rangle \equiv \langle |\boldsymbol{r}_N - \boldsymbol{r}_0|^2 \rangle$，慣性半径の 2 乗平均 $\langle s^2 \rangle$ を計算する部分をプログラムに追加する．ただし，慣性半径の 2 乗平均は，重心 $\boldsymbol{X}_G \equiv (N+1)^{-1} \sum_i \boldsymbol{r}_i$ からみた i 番目のセグメントの相対位置ベクトルを $\boldsymbol{s}_i \equiv \boldsymbol{r}_i - \boldsymbol{X}_G$ としたときに

$$\langle s^2 \rangle \equiv \frac{1}{N+1} \sum_i \langle s_i^2 \rangle \tag{9.3.4}$$

で定義される．
3. 書き直したプログラムを用いて計算を行う．このときボンド数 N を変えて計算を行う．

計算結果を作図し，以下の考察を行う．

〈 **課題 1** 〉

1. 平均 2 乗末端間距離 $\langle R^2 \rangle$ と慣性半径の 2 乗平均 $\langle s^2 \rangle$ を種々の N の値に対してプロットし，どのような依存性を示すかを考察せよ．

〈 **発展的課題 1** 〉

1. 末端間距離分布を計算する部分をプログラムに追加せよ．分布の求め方は付録 9.C に示しているプログラムを参考にすること．
2. $N = 20$ の場合に末端間距離分布をグラフにプロットし，ガウス鎖の場合と比較せよ．また，末端間距離分布のグラフに平均 2 乗末端間距離，慣性半径を記入せよ．

9.3.3 モンテカルロ法による高分子の張力と伸長の関係

次に，図 9.4 に示しているように，高分子の末端を距離 R に保ったときに末端に働く張力 $\boldsymbol{f}$ をモンテカルロ・シミュレーションにより求めてみよう．理想鎖の場合には，末端のセグメントに働く力は結合されたバネからの力だけなので，図 9.4 に示すように末端間距離を固定してモンテカルロ計算を行い，バネから末端のセグメントに働く力の平均値を求めればよい．

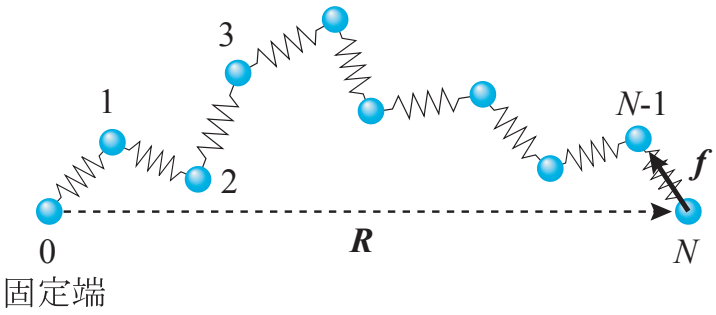

図 9.4: 鎖の配置．末端間ベクトルを $\boldsymbol{R}$ に保ったときに鎖の末端に働く力を $\boldsymbol{f}$ とする．

次の手順で解析プログラムの作成，計算，考察を行う：

〈 **実習 2** 〉

1. 高分子鎖の末端を固定するようにプログラム `polymer_free.c` を書き換えて新たにプログラム `polymer_fix.c` を作成する．末端ベクトル $\boldsymbol{R}$ の方向は x 軸方向にとる．
2. 力の平均値を計算する部分をプログラム `polymer_fix.c` に追加する．力は x，y，z 方向の成分を計算し，それぞれの平均値 $\langle f_x \rangle$，$\langle f_y \rangle$，$\langle f_z \rangle$ を求める．
3. 書き直したプログラムを用いて計算を行う．このときセグメント数を一定 ($N = 20$) とし，末端間距離 R を変えて計算を行う．エネルギーが $k_{\mathrm{B}}T \equiv \beta^{-1}$，長さが a でスケールされていることを反映して，計算により得られる力，末端間距離の値は，$\beta \boldsymbol{f} a$，R/a となっていることに注意する．

計算結果を用いて，次の手順で張力の分子論的な起源についての考察を行う：

〈 **課題 2** 〉

1. 末端に働く力の x，y，z 方向の成分の平均値から，張力の働く方向を特定せよ．また，張力の働く方向と末端ベクトルの方向の関係を考察せよ．
2. 上記 1 で求めた方向の張力の大きさ f と末端間距離 R の関係をグラフにプロットせよ．このとき無次元化した変数 $\beta f a$ と R/Na を用いること．また，ガウス鎖，ランダム・フライト鎖の張力・伸長関係をシミュレーション結果と同じグラフにプロットし，それらの間の共通点，相違点について考察せよ．
3. 張力の起源について考察する．

   (a) 高分子鎖の取り得るコンホメーションの数は末端間距離と共にどのように変化すると考えられるか．プログラムを実行したとき三次元表示される高分子のコンホメーションをよく観察して考察せよ．

   (b) 項目 (a) の考察をもとにエントロピーをキーワードとして高分子鎖の張力・伸長関係を説明せよ．

〈 **発展的課題 2** 〉

1. 末端間距離分布と張力はどのような関係にあるか．関係式を示し，その意味を論じよ．また，発展的課題 1 で計算した末端間距離分布から実際に張力を計算し，MC シミュレーションにより直接計算された結果と比較せよ．

9.4 計算機実験のための C プログラミング入門

ここでは，MC 法のプログラムを作成するために必要な C 言語の文法事項を簡単なプログラムを例にして解説をする．まず，注意事項をまとめておく．

1. UNIX の基本的なコマンド，エディタ emacs は使えるものと仮定する．「基礎情報処理演習」で用いたテキストを必ず持参し参照すること．

2. ここでの文法事項の説明は必要最小限にしてある．本実験に必要な知識は本節の説明で十分であるが，更に進んで勉強したい人は文法書を参照すること．

3. ここで示したプログラムは教育目的のために敢えて効率の悪い計算を行っている部分がある．そのような部分については各自で改良してよいが，その場合には，まず解説に従って正しく動作するプログラムを作成し，それを保存しておくこと．その後で自分で考えたプログラムを別の名前のプログラムとして作る．勝手にプログラムを改変してしまうと，うまく動作しなくなった場合に指導が困難になる．

9.4.1 hello, world

まず最も簡単な C のプログラムを示す.

プログラム：hello.c

```
1  #include <stdio.h>
2
3  int main(){
4
5    printf("hello, world \n");
6
7    return 0;
8  }
```

- 1 行目は標準入出力関数を宣言しているヘッダファイル stdio.h を読み込んでいる．これは，5 行目の printf を使うのに必要．標準ヘッダファイルを読み込むときはその名前を<>で囲む.
- C のプログラムは関数からなる．main 関数はプログラムの本体である．関数には型があり，3 行目では main 関数は整数型であることを示していて，7 行目では return 文で 0 を返している．関数については後でもう少し詳しく説明するので，ここではこれと同じように書いておけばよい.
- main() の後の{と}で囲まれた部分が関数の中身.
- 5 行目の printf() は，""で囲まれた部分を標準出力に出力．\n は改行．最後のセミコロンは文の終わりを示している.

プログラムを実行するには，まず gcc コマンドを使ってコンパイルする．次のように書くと hello.c をコンパイルし，hello という実行形式のファイルを作る.

```
gcc␣-o␣hello␣hello.c
```

実行するには

```
./hello
```

とする.

9.4.2 四則演算

プログラム：calci.c

```
1  #include <stdio.h>
2
3  int main(){
4    int i;
5
6    i = 2;
7    printf("i=%d \n",i);
8    i = i + 3;
9    printf("i=%d \n",i);
10   i += 3;
11   printf("i=%d \n",i);
12
13   return 0;
14 }
```

- データや計算結果を入れておくのに変数を用いる．変数は「型」を持っている．プログラムで使う変数は最初に型を宣言しなければならない．4 行目では整数型の変数 i を宣言している.
- 6 行目では i に 2 を代入している．ここの ”= ”は i が 2 等しいという数学記号ではないので注意する．C 言語で等しいことを表す記号は ”= ”ではなく ”== ”である．これについては条件文のところで説明する.
- 8 行目は，i+3 を計算し，それを左辺の i に代入するという意味である．=を左辺と右辺が等しいことを表す数学記号だと思うとおかしなことになる.
- 10 行目の書き方は，8 行目を短くした書き方である．引算，掛算，割算についても，-=，*=，/=などを使うことができる.

- printf 中での%d は整数型変数を書き出す際の書式である．

問 9.4.1: プログラム calci.c で+=の代わりに/=を用いて書き直し，実行せよ．結果はどうなるか．

問 9.4.1 の実行結果は i=1 となったはずである．これは，5/3 を計算し，小数点以下を切り捨てて左辺に代入したからである．整数型演算では小数点以下は切り捨てられるので注意が必要である．実数を扱うために次のプログラムでは，実数型変数を使っている．

プログラム：calcd.c

```
 1  #include <stdio.h>
 2
 3  int main(){
 4    double x;
 5
 6    x = 2.0;
 7    printf("x=%f \n",x);
 8    x /= 3.0;
 9    printf("x=%f \n",x);
10
11    return 0;
12  }
```

- 4 行目では，倍精度実数型変数として x を宣言している．実数型変数には，精度によって単精度型 float と倍精度型 double があるが，普通は倍精度型を使うと思っておけばよい．
- printf 中での%f は実数型変数を出力する際の書式である．

問 9.4.2: 次のプログラム calct.c を実行するとどのような結果が出力されると考えられるか．プログラムを実行する前に自分で考えてみること．その後でプログラムを実行し，自分の考えが合っていたかどうかを確認せよ．

プログラム：calct.c

```
 1  #include <stdio.h>
 2
 3  int main(){
 4    int    i,j;
 5    double x,y,z;
 6
 7    x = 2.2;
 8    y = 5.6;
 9
10    i = x;
11    printf("i=%d \n",i);
12    i = y;
13    printf("i=%d \n",i);
14
15    i = x * y;
16    printf("i=%d \n",i);
17
18    i = 20;
19    j = x / i;
20    z = x / i;
21    printf("j=%d, z=%f \n",j,z);
22
23    j = 30;
24    z = i / j;
25    printf("z=%f \n",z);
26
27    y = x;
28    x = y;
29    printf("x=%f, y=%f \n",x,y);
30
31    return 0;
32  }
```

問 9.4.3: プログラム calct.c の 24 行目では，本当は 20/30 の計算結果を倍精度にして出力したかったのだが，うまく行かなかったはずである．どうすればよいか．

問 9.4.4: プログラム calct.c の 27，28 行目では，本当は x と y の値を入れ替えたかったのだがうまく行かなかったはずである．どうすればよいか．

9.4.3 条件文と標準入力

これまでは変数の値はプログラムの中で設定してきたが，それを標準入力（キーボード）から入力し，その値に応じて動作するプログラム例を示す．

プログラム：if.c

```
 1  #include <stdio.h>
 2
 3  int main(){
 4    double x;
 5
 6    scanf("%lf",&x);
 7    if(x<0.0){
 8      printf("%f < 0.0 \n",x);
 9    }
10    else if( (x>=0.0) && (x<2.0) ){
11      printf("0.0 <= %f < 2.0 \n",x);
12    }
13    else{
14      printf("%f >= 2.0 \n",x);
15    }
16
17    return 0;
18  }
```

- 6 行目では scanf 命令で倍精度実数型変数に値を読み込んでいる．%f ではなく%lf としているのは倍精度型だからである．書式にはいろいろあるが，とりあえず倍精度変数のときはこのように書くと覚えておく．整数型の変数のときは%d．&x の&は変数 x へのポインタを示す演算子であるが，ここではポインタについては説明しないので，これも書式であると考えておく．

- if 文は条件によってプログラムの実行を制御する文であり，

  ```
  if(条件 1){
    文 1
  }
  else if(条件 2){
    文 2
  }
  else{
    文 3
  }
  ```

  という形式で書く．この場合，条件 1 が満たされれば文 1 が実行され，条件 1 が満たされず条件 2 が満たされれば文 2 が実行される．条件 1 も 2 も満たされなければ文 3 が実行される．

- 条件を記述する関係演算子には，

==	左辺と右辺の値が等しい
!=	左辺と右辺の値が等しくない
>	左辺が右辺より大きい
<	左辺が右辺より小さい
>=	左辺が右辺以上
<=	左辺が右辺以下

  がある．

- 10 行目では，複数の条件の論理積 (&&) を使って条件を指定している．論理和 (||) なども使うことができる．

9.4.4 簡単なループ

何度も同じ計算を繰り返すときはループを使う．

プログラム：loop.c

```
 1  #include <stdio.h>
 2
 3  int main(){
 4    int    i,sum;
 5
 6    sum=0;
 7    for(i=1; i<=10; i++){
 8      sum += i;
 9    }
10    printf("sum=%d \n",sum);
11
12    return 0;
13  }
```

- 7 行目には for 文を使っている．for 文の書式は

  for(式 1; 式 2; 式 3)

  であり，

  1. ループの初期化：式 1 を評価して，繰り返しに用いる変数の初期値を代入．
  2. 繰り返しの判定：式 2 を評価し，その値が 0 なら for 文の実行を終了．そうでなければループ本体の実行．
  3. ループ本体の実行後，式 3 を評価し，繰り返しを制御する変数の値を増加，もしくは減少させ，2 へ戻る．++は増分演算子といい，i++は i+=1 と同じ意味である．

9.4.5 関数

プログラム loop.c では整数の和を求めるプログラムを作った．和を求める計算はプログラムの他の部分でも使用できるかもしれないので，そういう決まりきった作業をする部分はひとまとめにしておけば便利である．以下のプログラムでは和を求める部分を一つの関数としている．計算結果はプログラム loop.c と全く同様である．

プログラム：function.c

```
 1  #include <stdio.h>
 2
 3  int Calc_summation(int,int);          /* 和を計算する関数のプロトタイプ宣言 */
 4
 5  int main(){
 6    int answer;
 7
 8    answer = Calc_summation(1,10);      /* 和を計算する関数の呼び出し */
 9    printf("answer=%d \n",answer);
10
11    return 0;
12  }
13                                        /* 和を計算する関数の定義 */
14  int Calc_summation(int lower_limit, int upper_limit){
15    int i,sum;
16
17    sum = 0;
18    for(i=lower_limit; i<=upper_limit; i++){
19      sum += i;
20    }
21    return sum;
22  }
```

- 3 行目では，Calc_summation という名前の整数型の関数を宣言している．(int,int) の中は整数型の引数を二つとることを示している．これを関数プロトタイプ宣言という．
- 14 行目から 22 行目までが関数の定義である．ここでは，和の上限と下限を引数で指定している．

- 21 行目の return 文が実行されると，return の後ろに書かれた式が評価されて，その値が関数の値として返される．
- 関数を呼び出す際は，8 行目にあるように引数を指定して呼び出す．この例では，関数が実行されれば，1 が lower_limit に，10 が upper_limit に渡されて，和の計算が実行され，結果が answer に返される．
- C 言語では/* */で囲めばその部分は注釈として解釈される．本節のプログラムにも理解を助ける目的で注釈を入れている．

9.4.6 配列

配列は通し番号を付けた変数をひとまとめにしたものである．配列を使ったプログラム例を示す．

プログラム：array.c

```
 1  #include <stdio.h>
 2
 3  int main(){
 4    int    i;
 5    double a, x[10];
 6
 7    for(i=0; i<10; i++){
 8      x[i] = 2.0 * i;
 9      printf("i=%d, x[%d]=%f \n",i,i,x[i]);
10    }
11
12    a = x[1] + x[5];
13    printf("a=%f \n",a);
14
15    return 0;
16  }
```

- 5 行目では配列を宣言してる．[] の中は配列の要素数を示している．C 言語では，要素の番号は 0 から始まるので，この例では，x[0]，x[1]，x[2]，$\cdots$，x[9] までの要素が用意される．
- 個々の要素は添字で指定できる．8 行目では変数 i で要素を指定し，値を代入している．12 行目では x[1] と x[5] の値の和を求めて，a に代入している．

次のプログラムは配列の要素の平均値を計算するプログラムである．

プログラム：average.c

```
 1  #include <stdio.h>
 2
 3  int main(){
 4    int    i,sum;
 5    double x[10],average;
 6
 7    for(i=0; i<10; i++){
 8      x[i] = 2.0 * i;
 9      printf("i=%d, x[%d]=%f \n",i,i,x[i]);
10    }
11
12    sum = 0;
13    average = 0.0;
14    for(i=0; i<10; i++){
15      average += x[i];
16      sum++;
17    }
18    average /= sum;
19    printf("average=%f \n",average);
20
21    return 0;
22  }
```

次のプログラムは，平均値を計算する部分を関数にしている．

プログラム：averagef.c

```
 1  #include <stdio.h>
 2
 3  double x[10];                              /* 外部配列などの宣言 */
 4  double Calc_average(void);
 5
 6  int main(){
 7    int    i;
 8    double average;
 9
10    for(i=0; i<10; i++){
11      x[i] = 2.0 * i;
12      printf("i=%d, x[%d]=%f \n",i,i,x[i]);
13    }
14
15    average = Calc_average();
16    printf("average=%f \n",average);
17
18    return 0;
19  }
20
21  double Calc_average(void){                 /* 平均値を計算する関数 */
22    int    i,sum;
23    double ave;
24
25    sum = 0;
26    ave = 0.0;
27    for(i=0; i<10; i++){
28      ave += x[i];
29      sum++;
30    }
31    return ave/sum;
32  }
```

- この例では，3 行目で配列 x[10] が main 関数の外側で宣言されている．これを外部配列という．こうしておけばと x いう配列を main と Calc_average 両方の関数で共通して使用できる．
- この例では，10 行目から 13 行目において x に値が設定され，15 行目で Calc_average 関数が呼び出されると，Calc_average 関数内で x の平均値が計算され，その値が倍精度で average に返される．

9.4.7 数学関数

プログラム：sugaku.c

```
 1  #include <stdio.h>
 2  #include <math.h>                /* 標準の数学関数用ヘッダファイルのインクルード */
 3
 4  int main(){
 5    int    i;
 6    double x[10],y[10];
 7
 8    for(i=0; i<10; i++){
 9      x[i] = 0.1 * i;
10      y[i] = tanh(x[i]);
11      printf("x[%d]=%f, y[%d]=%f \n",i,x[i],i,y[i]);
12    }
13
14    return 0;
15  }
```

- 2 行目では，sin などの数学関数をプログラム中で使うために，数学関数を宣言したヘッダファイル math.h をインクルードしている．

- コンパイル時には

  gcc␣-o␣sugaku␣sugaku.c␣-lm

  として，標準の数学ライブラリをリンクする．

- 使える数学関数は，sin(x)，cos(x)，tan(x)，sinh(x)，cosh(x)，tanh(x)，exp(x)，log(x)，log10(x)，pow(x,y)，sqrt(x)，floor(x) など．ここで x，y は倍精度実数型で，三角関数の角度はラジアン．pow(x,y) は x^y，sqrt(x) は $\sqrt{x}$，floor(x) は x より大きくない最大の整数である．

9.4.8 ファイルへの出力

計算結果をファイルに書き出す必要も生じると思われるので，プログラム例を示しておく．

プログラム：io.c

```
 1  #include <stdio.h>
 2  #include <math.h>
 3  #include <stdlib.h>
 4
 5  int main(){
 6    int    i;
 7    double x[10],y[10];
 8    FILE   *fp;                                  /* FILE 型ポインタ変数 fp の宣言 */
 9
10    if( (fp = fopen("file_name.dat", "a")) == NULL ) {   /* ファイルのオープン */
11      perror("wfp");
12      exit(1);
13    }
14
15    for(i=0; i<10; i++){
16      x[i] = 0.1 * i;
17      y[i] = tanh(x[i]);
18      printf("x[%d]=%f, y[%d]=%f \n",i,x[i],i,y[i]);
19    }
20
21    fprintf(fp,"# x, y \n");
22    for(i=0; i<10; i++){
23      fprintf(fp,"%f, %f \n",x[i],y[i]);
24    }
25
26    fclose(fp);                                          /* ファイルのクローズ */
27    return 0;
28  }
```

- ここの説明の詳細はポインタなどを理解していないと難しいと思われるので，とりあえずこのような書式で書けばよいと思っておけばよい．

- 8 行目で fp を FILE 型のポインタ変数として宣言．

- 10 行目で，file_name.dat という名前のファイルをオープンしている．この名前は各自必要に応じて書き換える．"a"はモードを指定していて，a はテキストファイルを追加モードでオープンするという指定である．ここを w とすると，書込みモードでオープンする．存在しないファイルを書込みあるいは追加モードでオープンすると指定した名前のファイルが新しく生成される．存在するファイルを書込みモードでオープンすると，以前のファイルの内容は消去される．

- 23 行目では，ファイルポインタ fp が指すファイルに出力している．printf では，出力先が標準出力 (ディスプレイ) になっていたので，fprintf では，その出力先をファイル名で指定したファイルに変更したと考えればよい．

- 26 行目では，ファイルのクローズを行っている．

- プログラムを実行して 10 行目で指定したファイルにデータが出力されていることを確認する．

9.5 ライブラリ lce(Library for Computer Experiments) の使い方

本実験では，実習がスムーズに進むように C プログラムから簡単に利用できるライブラリ，数値計算用ライブラリ `nlce`(Numerical Library for Computer Experiments) とグラフィックライブラリ `glce`(Graphic Library for Computer Experiments) を用意している．入手方法は実験の際に説明するが，ここでは，その使用方法をサンプルプログラムを用いて簡単に解説する．

9.5.1 乱数

9.2 節で解説したように MC 法では乱数を用いる．本実験では，乱数を発生させる関数 `nlce_random()` を用意している．使い方を以下に示している．

プログラム：random.c

```
 1  #include <stdio.h>
 2  #include <math.h>
 3  #include "lce.h"              /* ライブラリ lce のヘッダファイルのインクルード */
 4
 5  int main(){
 6    int    i;
 7    double x;
 8
 9    init_nlce_random(4711);     /* 乱数発生関数の初期化 */
10
11    for(i=1;i<=10;i++){
12      x = nlce_random();        /* 乱数発生関数の呼び出し */
13      printf("%f \n",x);
14    }
15
16    return 0;
17  }
```

- 3 行目でライブラリのヘッダファイル lce.h を読み込んでいる．これは標準ライブラリではないので""で名前を囲んでいる．`lce` は Library for Computer Experiments の略．これ以降のプログラムでは必ずインクルードする．
- 9 行目で乱数の初期値を与える関数 `init_nlce_random()` を呼び出して，初期値を与えている．乱数を使い始める際には必ず呼び出す．
- 12 行目で `nlce_random()` を呼び出している．`nlce` は Numerical Library for Computer Experiments の略．この関数は $[0, 1)$ 区間の一様乱数を乗算合同法で生成する（付録 9.A 参照）．一度呼び出すと一個の乱数を生成し，倍精度でその値を返す．
- 実行の仕方がこれまでと違うので注意する．ライブラリをリンクするために次のようにして実行する．

  `gcc␣-o␣random␣random.c␣-L./␣-llce␣-lm`

  `-L./`はライブラリをカレントディレクトリから検索することを指定し，`-llce` で liblce.a を指定している．

問 9.5.1: プログラム random.c で発生させた乱数の平均値，分散を計算するプログラムを作成し，実際に計算せよ．`x` を配列にして，平均値の計算の部分は，前節で作成した関数をコピー＆ペーストすると便利．サンプル数 `M` を変えてどの程度の精度で理論値と一致するか確認すること．

9.5.2 グラフィックス

簡単な図形の三次元表示

これから計算する高分子のコンフォメーションなどを描画するためにグラフィック・ライブラリを使う．まず球とシリンダを描くプログラムを示す．

プログラム：graphics.c

```
 1  #include <stdio.h>
 2  #include <math.h>
 3  #include "lce.h"                        /* lce のヘッダファイルのインクルード */
 4
 5  int main(){
 6    double x,y,z,x1,y1,z1;
 7
 8    glce_init(1,1);                       /* glce の初期化 */
 9
10    x=0.0;y=0.0;z=0.0;
11    glce_color(0,255,0);                  /* 色 RGB の指定 */
12    glce_sphere(x,y,z,2.0);               /* 球の描画 */
13
14    x1=5.0;y1=5.0;z1=-5.0;
15    glce_color(100,100,255);
16    glce_cylinder(x,y,z,x1,y1,z1,1.0); /* シリンダの描画 */
17
18    x=5.0;y=5.0;z=-5.0;
19    glce_color(255,0,0);
20    glce_sphere(x,y,z,2.0);
21
22    glce_color(255,255,255);
23    glce_boundary();                      /* 境界を示す立方体の描画 */
24
25    glce_display();                       /* 描画したデータのディスプレイへの表示 */
26
27    getchar();
28
29    glce_end();                           /* glce の終了処理 */
30
31    return 0;
32  }
```

- 3 行目では，ヘッダファイルを読み込んでいる．これは，数値計算ライブラリと共通である．
- 8 行目では，グラフィック・ライブラリを初期化．引数はとりあえず (1,1) のままにしておく．関数名の glce は Graphic Library for Computer Experiments の略．
- 11 行目：描画する色の指定．色は三原色 RGB（赤，緑，青）を使って表現されており，それぞれの色を 0 から 255 の数で指定する．
- 12 行目：球の描画．球の中心の座標と半径を与える．このライブラリ中での座標系は，水平方向が x 軸，垂直方向 y 軸，手前に向かって z 軸の右手系であり，ウィンドウの中に x, y, z 方向それぞれ -14 から 14 までの領域が描かれるようになっている．描画はバックグラウンドで行われるので，関数を呼び出した時点では何も表示されない．
- 16 行目：シリンダの描画．シリンダの始点と終点の座標および太さ（断面の半径）を与える．
- 23 行目：一辺の長さが 20 の立方体 ($-10 < x < 10$，$-10 < y < 10$，$-10 < z < 10$) の辺を描く．
- 25 行目：描画したデータをディスプレイに表示．
- 27 行目：getchar() はキーボードから文字を読み込むためのライブラリ関数であり，この関数が実行されるとキーボードからの入力待ちになる．ここでは，この性質を利用して処理を一時停止するために使っている．この行がないと描画のためにポップアップしたウィンドウが 29 行目の終了処理が実行されて一瞬で消えてしまう．
- 29 行目：終了処理．作図したウィンドウも消去される．
- グラフィック・ライブラリ glce を用いる場合には，X-Window のライブラリをリンクするために次のようにして実行する．

  ```
  gcc␣-o␣graphics␣graphics.c␣-L./␣-llce␣-L/usr/X11R6/lib␣-lX11␣-lm
  ```

- プログラムを実際に動かして，画面にウィンドウがポップアップし，図が描かれていることを確認する．パラメータを変えていろいろな図を描いてみる．

酔歩

乱数とグラフィックスの簡単な例題として，酔歩 (ランダム・ウォーク) を取り上げる．

プログラム：random_walk.c

```
 1  #include <stdio.h>
 2  #include <math.h>
 3  #include "lce.h"
 4
 5  int main(){
 6    int    i,M;
 7    double x,y,z;
 8
 9    glce_init(1,1);
10    init_nlce_random(4711);
11
12    M=100;
13
14    x = 0.0; y = 0.0; z = 0.0;
15    for(i=1;i<=M;i++){
16      x += 2.0 * nlce_random() - 1.0;
17      glce_color(80,80,200);
18      glce_sphere(x,y,z,1.0);
19      glce_color(255,255,255);
20      glce_boundary();
21      glce_display();
22    }
23    getchar();
24
25    glce_end();
26    return 0;
27  }
```

このプログラムでは現在の位置からの x 方向の移動距離を -1 から 1 の範囲から乱数で決めている．プログラムの意味は分かると思うので，実際に動かしてみる．

問 9.5.2: プログラム random_walk.c で，y, z 方向にも移動するようにプログラムを書きかえよ．

発展問題 9.5.1: 始点と終点の間の距離 R の 2 乗平均 $\langle R^2 \rangle$ と移動した回数 M の関係はどうなるか．

9.6 モンテカルロ・シミュレーション用のプログラム作成

ここでは，簡単なプログラムを段階的に発展させ，実習 1，2 のモンテカルロ・シミュレーション用のソースコードを作成する．

9.6.1 単純サンプリング法

高分子のモンテカルロ・シミュレーション用のプログラムを作成する手始めとして，まず単純サンプリング法で二つのセグメントがバネで結合された二量体の末端間距離の 2 乗平均 $\langle R^2 \rangle \equiv \langle |\boldsymbol{r}_1 - \boldsymbol{r}_0|^2 \rangle$ を計算するプログラムを考える[3]．ここではボンドのポテンシャルエネルギーは

$$U_{\rm bond}(l) = \frac{k_{\rm bond}}{2} l^2 \tag{9.6.1}$$

で与えられるとする．ここで $l = |\boldsymbol{r}_1 - \boldsymbol{r}_0|$．シミュレーションでは，エネルギーと長さはそれぞれ $k_{\rm B}T$ と a で無次元化されている．ここでは二量体一分子のみを考えるので，最初のセグメントの位置 $\boldsymbol{r}_0$ が原点に固定されているとする (図 9.3 参照)．単純サンプリングではもう一方のセグメントの位置をランダムに選ぶことになるが，ここでは計算の都合上，末端間距離 R は $R \leq R_{\rm max}$ に制限されているとして計算を行う．具体的には，セグメント 1 の位置を一辺の長さが $2R_{\rm max}$ の立方体 ($-R_{\rm max} < x < R_{\rm max}$，$-R_{\rm max} < y < R_{\rm max}$，$-R_{\rm max} < z < R_{\rm max}$) の領域内からランダムに選び，$R \leq R_{\rm max}$ の時に末端間距離を計算し，そのサンプル平均を求める．以下のプログラムを参考にして，random_walk.c を変更し，プログラムを作成せよ．

[3] 二量体なので末端間距離とボンド長は同じになるが，後の便宜上ここでは末端間距離と呼んでおく．

プログラム：dimer_simple.c

```
1  #include <stdio.h>
2  #include <math.h>
3  #include "lce.h"
4
5  int    N;
6  double rx[2],ry[2],rz[2];
7
8  double Calc_rij2(double,double,double,double,double,double);
9  void   Draw_dimer(void);
10
11 int main(){
12   int    iter,M;
13   double k_bond,U_bond,U,beta;
14   double R2,Ave_R2,Q;
15   double R_max,R2_max;
16
17   glce_init(1,1);
18   init_nlce_random(4711);
19
20   M      = 1000;                                  /* 変数などの初期化 */
21   N      = 1;
22   k_bond = 10.0;
23   beta   = 1.0;
24
25   R_max  = 10.0;
26   R2_max = R_max * R_max;
27
28   Ave_R2 = 0.0;
29   Q      = 0.0;
30
31   rx[0] = 0.0;
32   ry[0] = 0.0;
33   rz[0] = 0.0;
34   for(iter=0;iter<M;iter++){
35     rx[1] = R_max * (2.0 * nlce_random() - 1.0); /* セグメント 1 の位置の決定 */
36     ry[1] = R_max * (2.0 * nlce_random() - 1.0);
37     rz[1] = R_max * (2.0 * nlce_random() - 1.0);
38
39     R2 = Calc_rij2(rx[1],ry[1],rz[1],rx[0],ry[0],rz[0]); /* R2 の計算 */
40
41     if(R2<=R2_max){                            /* R2<=R2_max かどうかの判定 */
42       U_bond = 0.5*k_bond*R2;                     /* U_bond の計算 */
43       U      = U_bond;
44
45       Ave_R2 += R2*exp(-beta*U);
46       Q      += exp(-beta*U);
47
48       if(iter%10==0) Draw_dimer();             /* 10 ステップ毎に二量体の描画 */
49     }
50   }
51
52   Ave_R2 /= Q;
53   printf("Ave_R2=%f \n",Ave_R2);
54
55   glce_end();
56   return 0;
57 }
58                                                /* 二点間の距離の 2 乗を計算する関数 */
59 double Calc_rij2(double x1,double y1,double z1,double x0,double y0, double z0){
60   return (x1-x0)*(x1-x0)+(y1-y0)*(y1-y0)+(z1-z0)*(z1-z0);
61 }
62                                                /* 二量体の描画を行う関数 */
63 void Draw_dimer(void){
64   int i;
65
66   for(i=0;i<=N;i++){
67     glce_color(80,80,200);
68     glce_sphere(rx[i],ry[i],rz[i],0.2);
69   }
70   glce_color(150,150,180);
71   glce_cylinder(rx[0],ry[0],rz[0],rx[1],ry[1],rz[1],0.1);
72   glce_color(255,255,255);
73   glce_boundary();
74   glce_display();
75 }
```

- 35-37 行目では，一辺の長さが $2R_{\max}$ の立方体 ($-R_{\max} < x < R_{\max}$，$-R_{\max} < y < R_{\max}$，$-R_{\max} < z < R_{\max}$) の領域内からランダムにセグメント 1 の位置を選んでいる．
- 41 行目：39 行目で計算した R^2 の値を使って $R^2 \leq R_{\max}^2$ かどうかの判定．
- 42 行目：ボンドのポテンシャルエネルギー $U_{\rm bond}$ の計算．
- 45-46 行目：単純サンプリングで平均を計算する式 (9.2.3) の分子，分母の計算．
- 48 行目：10 ステップ毎にスナップショットの描画 (`Draw_dimer`) を行うための `if` 文．モジュロ演算子%は，割り算した余りを与える演算子．描画には時間がかかるので，MC ステップが多くなってきたら，それに応じて描画間隔を調節すること．
- 63-75 行目：ビーズとビーズ間のボンドをそれぞれ球とシリンダーで描画．

問 9.6.1: 末端間距離に制限がないとして $\langle R^2 \rangle$ の理論値を導出せよ．サンプル数 `M` を 100 から 10 万程度にまで増加させて理論値に近づくかどうか確認せよ．

9.6.2 重みつきサンプリング

さて，二量体の問題で単純サンプリング法は非常に効率が悪いことが分かったはずである．これは，セグメントの位置をランダムに選んでしまうために，平均値の計算に寄与の大きいボンドエネルギーが小さい配置のサンプリングを効率的に行えないためである．これをメトロポリス法によって解決しよう．以下のプログラムを参考にして，単純サンプリングのプログラムをメトロポリス法のプログラムに変更する．9.2.3 節のメトロポリス法のアルゴリズムとの対応を考えながらプログラミングすること．

dimer_simple.c からの主な変更点：

- 34-38 行目：初期値の設定．
- 46 行目：1 番目のビーズを選ぶ．本来，メトロポリス法のランダムにビーズを選ぶステップ 1 がここに対応するが，ここでは選ぶべきビーズが一個しかないので，そのステップが省略されている．
- 48-50 行目：変位の計算．メトロポリス法のステップ 2．
- 52-54 行目：ここでは，末端間距離に $R \leq R_{\max}$ の制限があるので，試行的な配置が $R > R_{\max}$ の場合にはエネルギーが無限大になるとして，フラグ `flg` に 1 を与えて，乱数との比較をすることなく，試行的な配置を棄却している．
- 57-59 行目：ΔU の計算．メトロポリス法のステップ 3．
- 61-72 行目：試行的な配置を採択するかどうかの判定．メトロポリス法のステップ 4，5．
- 65 行目：試行した配置が採用された割合を `acceptance_ratio` として計算している．一般に，$\delta r_{\max}$ が大きいと一回の試行での変位量が大きいので配置を大きく変えることができるが，ポテンシャルエネルギーが大きくなってしまい，試行配置が棄却される可能性が大きくなる．逆に，$\delta r_{\max}$ が小さいと試行配置が採用される確率は高くなるが，配置を大きく変えるのに多くのステップが必要になる[4]．シミュレーションでは $\delta r_{\max}$ を調節して試行配置が採用される割合が 5 割程度になるようにするとよいと言われている．
- 75-79 行目：平均値の計算．試行の回数が 10 回以上場合に平均の計算を行っている点に注意する．初期配置は平衡状態から大きく外れている可能性があるので，ある程度モンテカルロ試行を行い，配置が平衡状態に近づいてから平均を取るようにする．この平衡化に要するモンテカルロステップはパラメータなどにより変わってくるので注意を要する．平衡化に要するステップ数は，本来は系の緩和時間などを検討して決定する必要があるが，ここでは便宜的にスナップショットを観察するなどして各自判断することにする．

[4] ここでは詳しく議論しないが，ここで使っているメトロポリス法ではある時点の配置に微小変位を加えて次の状態の配置を生成しているので，順次生成される配置の間には相関がある．位相空間でのサンプリングを効率的に行うためには，なるべく互いに相関のない独立な配置を生成する必要があるが，$\delta r_{\max}$ を小さくしすぎると，相関のない配置を生成するのに多くの MC ステップが必要になってしまう．

プログラム：dimer_importance.c

```
 1  #include <stdio.h>
 2  #include <math.h>
 3  #include "lce.h"
 4
 5  int    N;
 6  double rx[2],ry[2],rz[2],k_bond;
 7
 8  double Calc_U_bond(double,double,double,double,double,double);
 9  double Calc_rij2(double,double,double,double,double,double);
10  void   Draw_dimer(void);
11
12  int main(){
13    int    i,iter,M;
14    int    index_bead,flg;
15    double U_bond,beta;
16    double Delta_r_max,Delta_U;
17    double rx_new,ry_new,rz_new,U_bond_new;
18    double R2,Ave_R2,Q;
19    double R_max,R2_max;
20    double acceptance_ratio;
21
22    glce_init(1,1);
23    init_nlce_random(4711);
24
25    M      = 1000;
26    N      = 1;
27    beta   = 1.0;
28    k_bond = 10.0;
29    Delta_r_max = 0.3;
30
31    R_max  = 10.0;
32    R2_max = R_max * R_max;
33
34    for(i=0;i<=N;i++){
35      rx[i] = 1.0*i;
36      ry[i] = 0.0;
37      rz[i] = 0.0;
38    }
39    Draw_dimer();
40
41    Ave_R2 = 0.0;
42    Q      = 0.0;
43    acceptance_ratio = 0.0;
44
45    for(iter=0;iter<M;iter++){
46      index_bead = 1;
47                                                  /* 試行配置の計算 */
48      rx_new = rx[index_bead] + Delta_r_max * (2.0 * nlce_random() - 1.0);
49      ry_new = ry[index_bead] + Delta_r_max * (2.0 * nlce_random() - 1.0);
50      rz_new = rz[index_bead] + Delta_r_max * (2.0 * nlce_random() - 1.0);
51
52      flg=0;
53      R2 = Calc_rij2(rx_new,ry_new,rz_new,rx[0],ry[0],rz[0]);   /* R2 の計算 */
54      if(R2>R2_max) flg=1;                        /* R2>R2_max の時 flg=1 */
55
56      if(flg==0){                                 /* flg が 0 かどうかの判定 */
57        U_bond_new = Calc_U_bond(rx_new,ry_new,rz_new,rx[0],ry[0],rz[0]);
58        U_bond = Calc_U_bond(rx[index_bead],ry[index_bead],rz[index_bead],
59     rx[0],ry[0],rz[0]);
60        Delta_U = U_bond_new - U_bond;            /* エネルギー変化の計算 */
61
62        if(Delta_U<=0.0){                         /* 試行配置の採択の判定 */
63          rx[index_bead] = rx_new;
64          ry[index_bead] = ry_new;
65          rz[index_bead] = rz_new;
66          acceptance_ratio += 1.0;
67        }
68        else if(exp(-beta*Delta_U)>nlce_random()){
69          rx[index_bead] = rx_new;
70          ry[index_bead] = ry_new;
71          rz[index_bead] = rz_new;
72          acceptance_ratio += 1.0;
73        }
74      }
```

```
75                                              /* 10 ステップ以上の場合に平均値の計算 */
76      if(iter>=10){
77        R2 = Calc_rij2(rx[1],ry[1],rz[1],rx[0],ry[0],rz[0]);
78        Ave_R2 += R2;
79        Q      += 1.0;
80      }
81
82      if(iter%10==0) Draw_dimer();
83    }
84
85    Ave_R2 /= Q;
86    acceptance_ratio /= M;
87    printf("Ave_R2=%f, accept_ratio=%f \n",Ave_R2,acceptance_ratio);
88
89    glce_end();
90    return 0;
91  }
92                                              /* ボンドエネルギーを計算する関数 */
93  double Calc_U_bond(double x1,double y1,double z1,double x0,double y0, double z0){
94    double rij2;
95
96    rij2 = Calc_rij2(x1,y1,z1,x0,y0,z0);
97    return 0.5*k_bond*rij2;
98  }
99                                              /* 二点間の距離の 2 乗を計算する関数 */
100 double Calc_rij2(double x1,double y1,double z1,double x0,double y0, double z0){
101   return (x1-x0)*(x1-x0)+(y1-y0)*(y1-y0)+(z1-z0)*(z1-z0);
102 }
103                                             /* 二量体の描画を行う関数 */
104 void Draw_dimer(void){
105   int i;
106
107   for(i=0;i<=N;i++){
108     glce_color(80,80,200);
109     glce_sphere(rx[i],ry[i],rz[i],0.2);
110   }
111   glce_color(150,150,180);
112   glce_cylinder(rx[0],ry[0],rz[0],rx[1],ry[1],rz[1],0.1);
113   glce_color(255,255,255);
114   glce_boundary();
115   glce_display();
116 }
```

問 9.6.2: 単純サンプリングの場合と同様にサンプル数 `M` を増加させて理論値に近づくかどうか確認せよ．単純サンプリングとどちらが効率的か．

9.6.3 高分子のプログラムへの拡張

さて，二量体の問題でメトロポリス法の基本的なプログラムはできたが，ビーズ数が 3 以上の分子を扱うにはプログラムを拡張しておく必要がある．拡張のポイントを説明しているので，以下のプログラムを参考にして dimer_importance.c を書き換え，高分子用のプログラム polymer_free.c を作成する．

dimer_importance.c からの主な変更点：

- 39-43 行目：N 個のビーズの初期値の設定．
- 50 行目：N 個のビーズからランダムに一つのビーズを選ぶ．メトロポリス法のステップ 1．
- 56 行目：フラグ `flg` の計算．関数 `check` については最後に説明している．
- 91-108 行目：エネルギー差の計算．ここで注意するのは，高分子の場合にはどの位置のビーズが選択されたかによって計算が変わる点．高分子鎖の端のビーズが選ばれた場合には，高分子鎖に沿って隣接するビーズは一個しかないので，二量体の時と同様にそのビーズとのバネポテンシャルを計算すれば良い．そうでない場合には隣接するビーズは二つあるので，その二つのビーズとのバネポテンシャルを計算する．

- 110-116 行目：バネポテンシャルの計算．ここでは，式 (9.3.3) に与えられている FENE ポテンシャルを使っている．
- 122-135 行目：`Draw_dimer` を `Draw_polymer` に変更．
- 137-152 行目：バネで結合されているビーズ間の FENE ポテンシャルは，ビーズ間の距離 l が $l \leq l_{\min}$ もしくは $l \geq l_{\max}$ で発散しているので，もしその場合には 1 を返すようになっている．56 行目でこの関数から返された値がフラグ `flg` に入り，それがもし 1 ならば，乱数との比較をすることなく試行的な配置は棄却される．

プログラム：polymer_free.c

```
 1  #include <stdio.h>
 2  #include <math.h>
 3  #include "lce.h"
 4
 5  #define N_MAX 100
 6
 7  int    N;
 8  double rx[N_MAX+2],ry[N_MAX+2],rz[N_MAX+2];
 9  double k_bond,a,lmax,lmin;
10  double rx_new,ry_new,rz_new;
11
12  double Calc_Delta_U_bond(int);
13  double Calc_U_bond(double,double,double,double,double,double);
14  double Calc_rij2(double,double,double,double,double,double);
15  void   Draw_polymer(void);
16  int    Check(int);
17
18  int main(){
19    int    i,iter,M;
20    int    index_bead,flg;
21    double beta,Delta_r_max,Delta_U;
22    double R2,Ave_R2,Q;
23    double acceptance_ratio;
24
25    glce_init(1,1);
26    init_nlce_random(4711);
27
28    M = 1000000;                                /* 変数などの初期化 */
29
30    N      = 10;
31    k_bond = 50.0;
32    beta   = 1.0;
33    Delta_r_max = 0.3;
34
35    a    = 1.0;
36    lmax = 1.2;
37    lmin = 0.8;
38
39    for(i=0;i<=N;i++){                          /* 高分子の初期配置の設定 */
40      rx[i] = i;
41      ry[i] = 0.0;
42      rz[i] = 0.0;
43    }
44    Draw_polymer();
45
46    Ave_R2 = 0.0;
47    Q      = 0.0;
48    acceptance_ratio = 0.0;
49    for(iter=0;iter<M;iter++){
50      index_bead = floor((N+1)*nlce_random()); /* 変位させるセグメントの選択 */
51                                                /* 試行配置の計算 */
52      rx_new = rx[index_bead] + Delta_r_max * (2.0 * nlce_random() - 1.0);
53      ry_new = ry[index_bead] + Delta_r_max * (2.0 * nlce_random() - 1.0);
54      rz_new = rz[index_bead] + Delta_r_max * (2.0 * nlce_random() - 1.0);
55
56      flg = Check(index_bead);                  /* ボンド長のチェック */
57      if(flg==0){
58        Delta_U = Calc_Delta_U_bond(index_bead);  /* エネルギー差の計算 */
59
60        if(Delta_U<=0.0){                       /* 試行配置を採択するかどうかの判定 */
61          rx[index_bead] = rx_new;
62          ry[index_bead] = ry_new;
```

```
63          rz[index_bead] = rz_new;
64          acceptance_ratio += 1.0;
65        }
66        else if(exp(-beta*Delta_U)>nlce_random()){
67          rx[index_bead] = rx_new;
68          ry[index_bead] = ry_new;
69          rz[index_bead] = rz_new;
70          acceptance_ratio += 1.0;
71        }
72      }
73
74      if(iter>=100000){                       /* 100000 ステップ以上かどうかの判定 */
75        R2 = Calc_rij2(rx[N],ry[N],rz[N],rx[0],ry[0],rz[0]);
76        Ave_R2 += R2;
77        Q      += 1.0;
78      }
79
80      if(iter%10000==0) Draw_polymer();
81    }
82
83    Ave_R2 /= Q;
84    acceptance_ratio /= M;
85    printf("Ave_R2=%f, accept_ratio=%f \n",Ave_R2,acceptance_ratio);
86
87    glce_end();
88    return 0;
89  }
90                                              /* エネルギー差を計算する関数 */
91  double Calc_Delta_U_bond(int i){
92    int    j;
93    double U_bond,U_bond_new;
94
95    U_bond     = 0.0;
96    U_bond_new = 0.0;
97    if(i!=N){
98      j = i+1;
99      U_bond_new += Calc_U_bond(rx_new,ry_new,rz_new,rx[j],ry[j],rz[j]);
100     U_bond += Calc_U_bond(rx[i],ry[i],rz[i],rx[j],ry[j],rz[j]);
101   }
102   if(i!=0){
103     j = i-1;
104     U_bond_new += Calc_U_bond(rx_new,ry_new,rz_new,rx[j],ry[j],rz[j]);
105     U_bond += Calc_U_bond(rx[i],ry[i],rz[i],rx[j],ry[j],rz[j]);
106   }
107   return U_bond_new - U_bond;
108 }
109                                             /* ボンドエネルギーを計算する関数 */
110 double Calc_U_bond(double x1,double y1,double z1,double x0,double y0, double z0){
111   double rij;
112
113   rij = sqrt(Calc_rij2(x1,y1,z1,x0,y0,z0));
114   return -0.5*k_bond*(lmax-a)*(lmax-a)
115     *log(1.0 - (rij-a)*(rij-a)/(lmax-a)/(lmax-a));
116 }
117                                             /* 二点間の距離の 2 乗を計算する関数 */
118 double Calc_rij2(double x1,double y1,double z1,double x0,double y0, double z0){
119   return (x1-x0)*(x1-x0)+(y1-y0)*(y1-y0)+(z1-z0)*(z1-z0);
120 }
121                                             /* 高分子の描画を行う関数 */
122 void Draw_polymer(void){
123   int i;
124
125   for(i=0;i<=N;i++){
126     glce_color(80,80,200);
127     glce_sphere(rx[i],ry[i],rz[i],0.2);
128     if(i!=N){
129       glce_color(150,150,180);
130       glce_cylinder(rx[i],ry[i],rz[i],rx[i+1],ry[i+1],rz[i+1],0.1);
131     }
132   }
133   glce_color(255,255,255);
134   glce_display();
135 }
```

```
136                                                    /* ボンド長のチェックを行う関数 */
137  int Check(int i){
138    int    j;
139    double rij;
140
141    if(i!=N){
142      j = i+1;
143      rij = sqrt(Calc_rij2(rx_new,ry_new,rz_new,rx[j],ry[j],rz[j]));
144      if(rij>=lmax || rij<=lmin) return 1;
145    }
146    if(i!=0){
147      j = i-1;
148      rij = sqrt(Calc_rij2(rx_new,ry_new,rz_new,rx[j],ry[j],rz[j]));
149      if(rij>=lmax || rij<=lmin) return 1;
150    }
151    return 0;
152  }
```

以上で，高分子のモンテカルロ・シミュレーション用のプログラムが作成できたので，このプログラムをもとにして 2 節で与えられた課題の計算を行う．

付録

9.A　乱数

MC 法の実際の手続きでは乱数が重要な役目を果たす．計算機で使用する乱数は，なるべく相関のないように工夫された方法で生成される．しかし，完全に相関のない乱数を生成することはできないので，正確には疑似乱数である．ここでは簡単のためにこれを単に乱数と呼ぶことにする．乱数の生成法は数学的に詳しく研究されているが，本実験では簡単で実用的な生成法である乗算合同法を用いている．n 番目の乱数 ξ_n は $n-1$ 番目の乱数 ξ_{n-1} から

$$\xi_n = a\xi_{n-1} (\mathrm{mod} M)$$

と生成される．ここで，右辺は $a\xi_{n-1}$ の値を M で割った余りを意味し，a と M は正の整数である．生成された乱数が比較的良い統計的性質示す a と M の値が調べられているが，ここでは $a = 16807$，$M = 2^{32} - 1$ を用いている．

9.B　GNUPLOT の使い方

実習室の Linux ではグラフを作成するのに GNUPLOT を使うことができる．起動は，ターミナルから

```
gnuplot
```

とタイプすると，GNUPLOT のプロンプト

```
gnuplot>
```

が現れる．終了は

```
gnuplot> quit
```

とする．

データが `data_name.dat` というファイルに

```
1.1 2.3
2.5 5.6
3.8 7.2
```

という形で保存されている場合には，

```
gnuplot> plot "data_name.dat" with points
```

とすれば，点がプロットされる．データを線で結びたければ `with lines` とし，点をプロットし，それを線で結びたければ `with linespoints` とする．複数のデータがある場合は，

```
gnuplot> plot "data_name1.dat" with points, "data_name2.dat" with lines
```

などとする．
　x 軸を対数スケールにするには

```
gnuplot> set logscale x
```

とした後で `plot` を実行する．両対数プロットなら

```
gnuplot> set logscale xy
```

とする．通常のスケールに戻すには

```
gnuplot> set nologscale xy
```

とする．
　軸にラベルを付けるには

```
gnuplot> set xlabel "N"
gnuplot> set ylabel "R"
```

などとした後で，`plot` する．
　グラフを PostScript 形式のファイルに保存するには

```
gnuplot> set terminal postscript
gnuplot> set output "graph_name.ps"
```

とした後で `plot` を実行する．この時，描画されたグラフは指定したファイル (上の例では gnuplot を起動したディレクトリの `graph_name.ps`) に出力される．ディスプレイにはグラフは描画されないので注意する．作成した PostScript 形式のファイルは次のようにしてプリンタに出力することができる．

```
lpr graph_name.ps
```

上記の操作の後で再びディスプレイにグラフを表示するためには

```
gnuplot> set terminal x11
```

とする必要があるので注意する．

9.C　分布を求めるプログラム

　第 9.5.1 節で取り上げたプログラム random.c をもとに分布を計算するプログラムを作ってみよう．乱数は $[0,1)$ 区間で生成されるので，区間を 10 等分し，その区間に小さいほうから $0, 1, 2 \cdots, 9$ と番号を付けて，$0 \leq x < 0.1$ なら区間 0 に，$0.1 \leq x < 0.2$ なら区間 1 に，というように分類する．プログラムを以下に示す．平均値，分散を計算するプログラムに追加した部分は，分布を求める関数だけである．

プログラム：random_dis.c

```
 1  #include <stdio.h>
 2  #include <math.h>
 3  #include "lce.h"
 4
 5  #define M_MAX 10000                                   /* マクロの定義 */
 6  #define N_DIS_MAX 100
 7
 8  int    M,N_dis;
 9  double average,variance,bin;
10  double x[M_MAX+2];
11  double distribution[N_DIS_MAX+2];
12
13  void Calc_average(void);
14  void Calc_distribution(void);
15
16  int main(){
17    int    i;
```

```
18
19    M     = 10000;
20    N_dis = 10;
21    bin   = 1.0/N_dis;
22
23    init_nlce_random(4711);
24
25    for(i=0; i<M; i++){
26      x[i] = nlce_random();
27    }
28
29    Calc_average();
30    printf("average = %f, variance = %f \n",average,variance);
31
32    Calc_distribution();
33    for(i=0; i<N_dis; i++){
34      printf("%f %f \n",bin*(i+0.5),distribution[i]);
35    }
36    return 0;
37  }
38
39  void Calc_average(void){                                  /* 平均値を計算する関数 */
40    int    i,sum;
41
42    sum = 0;
43    average = 0.0;
44    variance = 0.0;
45    for(i=0; i<M; i++){
46      average  += x[i];
47      variance += x[i]*x[i];
48      sum++;
49    }
50    average /= sum;
51    variance = variance/sum - average*average;
52  }
53
54  void Calc_distribution(void){                             /* 分布を計算する関数 */
55    int    i,j;
56    double sum;
57
58    for(i=0; i<N_dis; i++){
59      distribution[i] = 0.0;
60    }
61
62    for(i=0; i<M; i++){
63      j = floor(x[i]*N_dis);
64      distribution[j] += 1.0;
65    }
66
67    sum = 0.0;
68    for(i=0; i<N_dis; i++){
69      sum += distribution[i]*bin;
70    }
71
72    for(i=0; i<N_dis; i++){
73      distribution[i] /= sum;
74    }
75  }
```

- 5，6 行目では，M_MAX，N_DIS_MAX という名前のマクロを定義している．これは，プログラム中で例えば M_MAX という名前が出てきたら，それを 10000 と置き換えるということを意味する．
- 63 行目では，floor を使って x の値に応じて区間を指定している．例えば x が 0.52 のときは，まず 10 倍して 5.2 とし，floor で 5.0 となる．

参考文献

[1] 統計アンサンブルの詳細については「統計熱力学入門」(創成化学コース) の講義内容を参照

[2] 「高分子化学基礎 I，II」(創成化学コース) の講義内容とテキストを参照

[3] 神山新一，佐藤明「分子シミュレーション講座 1，モンテカルロ・シミュレーション」(朝倉書店，1997)