

1本の高分子鎖の、Kremer-Grest 模型による統計物理計算

慶大理工（高野研 D2）

岩岡伸之

高分子物理学研究のための計算ツール研究会

第1回 LAMMPS 初心者講習会

2013/12/23 (月) @お茶の水女子大学

目次

(1) Kremer–Grest (KG) 模型とは

(2) 1本鎖の線状 KG 模型のシミュレーションと簡単な解析

- ▷ KG 模型のための入力スクリプトの復習
- ▷ 【練習1】 LAMMPS の dump から、基本的な物理量の計算
- ▷ 【練習2】 coil-globule 転移を観察

(3) 応用例；散逸粒子動力学（DPD）法を用いた相分離

- ▷ DPD について
- ▷ 【練習3】 小さい系で高分子のラメラ相を観察

(4) 高分子以外の例；1成分ガラス系

- ▷ 自作ポテンシャルのコンパイルと使い方
- ▷ 【練習4】 動径分布関数の計算

(1) Kremer-Grest模型とは

Kremer-Grest (KG) 模型

- G. S. Grest & K. Kremer: PRA 33 (1986) 3628

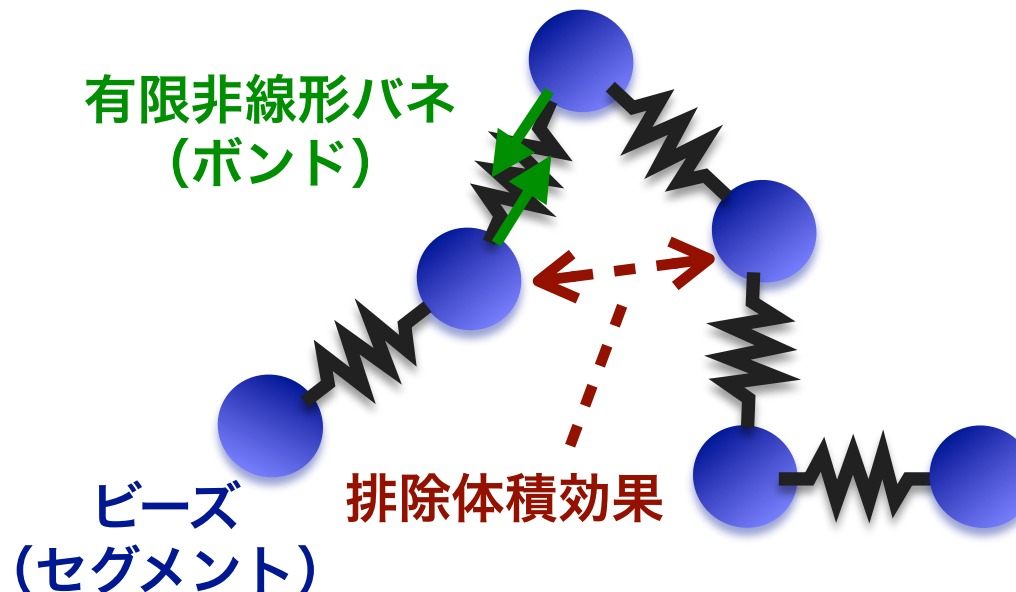
▷ 高分子セグメント ⇒ **ビーズ** (排除体積有り)

▷ ボンド ⇒ **有限非線形バネ**

} 現実鎖

▷ 異なるボンドの交差は起きない。

▷ 高分子系で普遍的な性質を良く再現。



KG 模型のポテンシャル

- 斥力 Lennard-Jones ポテンシャル

$$U_{ij}^0 = \begin{cases} 4\epsilon \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^6 + \frac{1}{4} \right], & r_{ij} \leq \sigma 2^{1/6} , \\ 0, & r_{ij} > \sigma 2^{1/6} , \end{cases}$$

▷ 任意のセグメント対の間に働く排除体積効果。

- 有限伸張非線形弾性 (FENE) ポテンシャル

$$U_{ij}^{\text{ch}} = \begin{cases} -0.5kR_0^2 \ln[1 - (r_{ij}/R_0)^2], & r_{ij} \leq R_0 , \\ 0, & r_{ij} > R_0 , \end{cases}$$

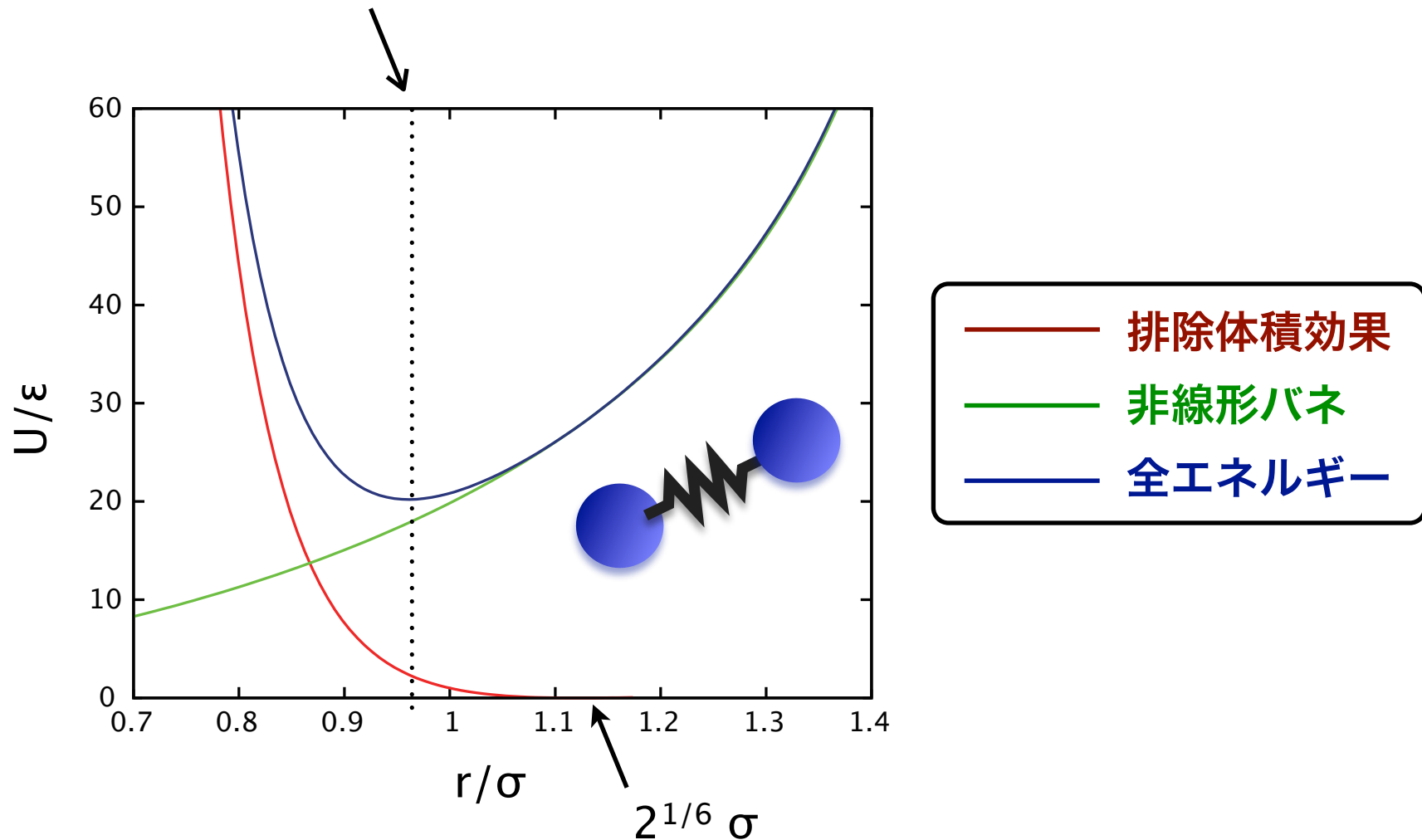
▷ 高分子構造に沿って隣接したセグメント対の間に働くバネの力。

▷ $k = 30.0 \epsilon/\sigma^2$, $R_0 = 1.5 \sigma$ が標準的。

KG 模型のポテンシャル

- 斥力 LJ + FENE ($k = 30.0 \epsilon/\sigma^2$, $R_0 = 1.5 \sigma$)

⊇ 平衡距離 : 0.97σ



(2) 1本鎖の線状 KG 模型の シミュレーションと簡単な解析

入カスクリプトの復習

- LAMMPSの入カスクリプトの基本構成（マニュアルより）

(A) 初期条件

▷ 単位系、次元、境界条件、粒子の型、など。

(B) 粒子の定義

▷ 粒子の位置・速度、分子のトポロジー情報など。

➡ 基本的に、テキストファイルから読み込む方針

(C) 系のパラメター設定

▷ 力場の係数、シミュレーション変数、出力設定、など

(D) シミュレーション・ラン

▷ (A) ~ (C) の設定に従い、シミュレーションを実行

入カスクリプトの復習

###

Section (A)

units lj

dimension 3

boundary p p p

atom_style bond

###

Section (B)

read_data data.polymer05 ----->

pair_style lj/cut 1.122462048309373

pair_coeff 1 1 1.0 1.0 1.122462048309373

ϵ , σ , r_c

bond_style fene

special_bonds fene

bond_coeff 1 30.0 1.5 1.0 1.0

K_{sp} , R_0 , ϵ , σ

velocity all create 1.0 12345 dist gaussian

LAMMPS G-K model of a single polymer

5 atoms

4 bonds

1 atom types

1 bond types

-10.00000 10.00000 xlo xhi

-10.00000 10.00000 ylo yhi

-10.00000 10.00000 zlo zhi

Atoms

1	1	1	-2.5	0.0	0.0
2	1	1	-1.5	0.0	0.0
3	1	1	-0.5	0.0	0.0
4	1	1	0.5	0.0	0.0
5	1	1	1.5	0.0	0.0

Masses

1	1.00000
---	---------

Bonds

1	1	1	2
2	1	2	3
3	1	3	4
4	1	4	5

入カスクリプトの復習

###

Section (C)

timestep 0.01

run_style verlet

neighbor 0.4 bin

neigh_modify every 1 delay 1

fix 1 all nve

fix 2 all langevin 1.0 1.0 2.0 13579

(左から順に) T_{start} , T_{end} , Γ^{-1} , seed

$$\} \ddot{\mathbf{r}}_i = -\nabla U_i - \Gamma \dot{\mathbf{r}}_i + \mathbf{W}_i(t)$$

###

Section (D)

run 50000 **dumpの間隔**

dump 1 all custom 50 dump.polymer id xu yu zu

dump_modify 1 sort id

run 50000

【練習1】

1) $N = 5 \sim 40$ までのデータファイル (data.polymer##) を読み、シミュレーションを行う。dump ファイルから鎖の慣性半径 or 末端間距離を計算し、 N 依存性を調べる。

▷ 余裕のある方は、自分でデータファイルを作成。

▷ dumpファイルの処理の仕方の例 : ana.f90 →

```
open(20, file='dump.polymer')
loop = 0
do
  read(20, *, end=10000)
  loop = loop + 1
  read(20, *) time
  read(20, *)
  read(20, *) natoms
  read(20, *)
  read(20, *) xlo, xhi
  read(20, *) ylo, yhi
  read(20, *) zlo, zhi
  read(20, *)
  do i=1, natoms
    read(20, *) itmp, xtmp, ytmp, ztmp
    x(itmp) = xtmp
    y(itmp) = ytmp
    z(itmp) = ztmp
  enddo
enddo
10000 CONTINUE
close(20)
```

2) $N = 200$ のデータファイルを使って、シミュレーションを実行し、リスタートファイルを作成する。

▷ リスタートファイルの作り方の例 :

➡ restart 1000 restart.polymer

【練習2】

1) 作成したリスタートファイル (N = 200) を読み込み、シミュレーションが再開できることを確認。

▷ リスタートファイルの読み方の例：

➡ `read_restart RESTART_FILE.*` # (A), (B) の情報が読まれる

2) 再開の際、LJ ポテンシャルのカットオフ長を 3.0 (引力あり) に変更し、シミュレーションを行い、dump を作成。

▷ `restart_read` の後に `pair_coeff` コマンドのパラメターを書き換える。

▷ Dump ファイルから各時間での R_e or R_g を計算し、時間変化をみる。

▷ Dump ファイルを xyz 形式で出力してみる：

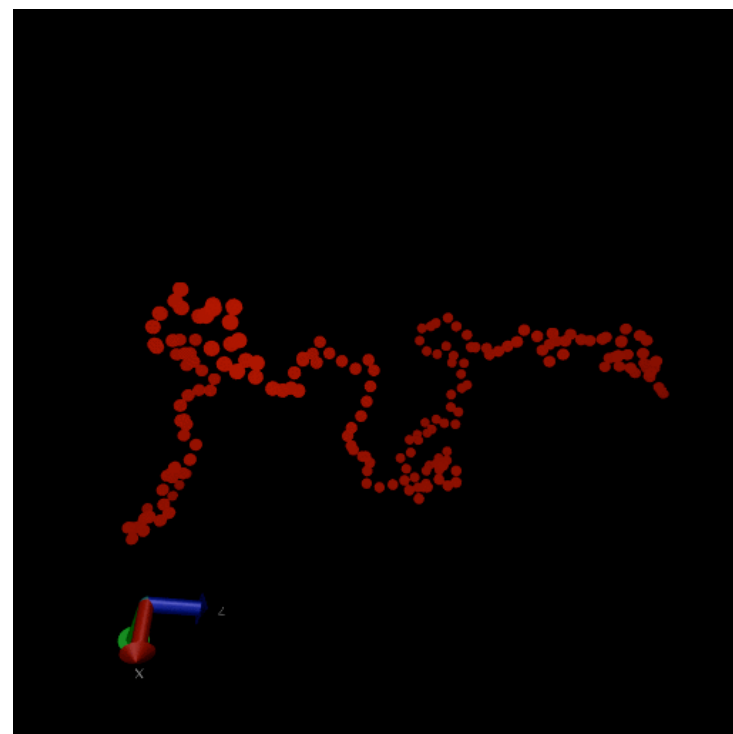
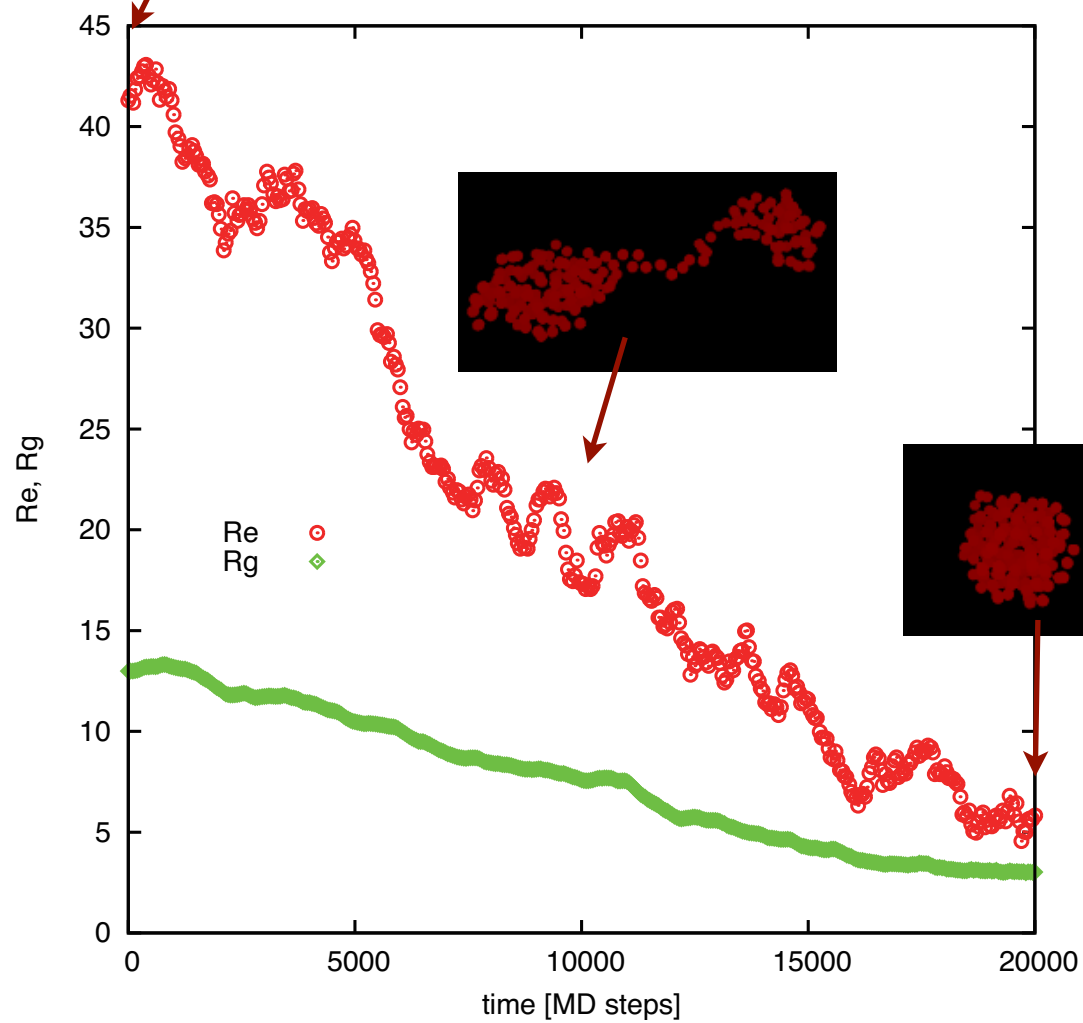
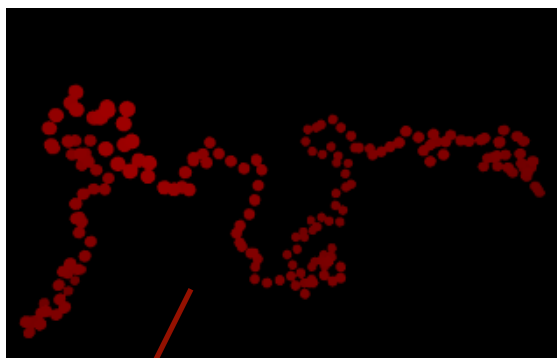
➡ `dump 2 all xyz 50 dump.polymer.xyz` # 位置情報のみ出力

3) VMD で動きを観る：Coil状態 → globule状態への転移

【練習2】の入力スクリプト

- in00.N200 : $r_c = 2^{1/6}$ の通常の KG 模型。
- in01.N200 : pair_coeff コマンドで $r_c = 3.0$ へ書き換え。
 - ▷ {pair, bond}_coeff での値は、適宜書き換えることが可能。
 - ➡ pair_coeff 1 1 1.0 1.0 1.12
run 100
pair_coeff 1 1 1.0 1.0 2.24
run 100
 - ▷ read_restart で前回のシミュレーションで用いた条件 (Section (A), (B)) は自動的に読み込まれる。よって、bond_coeff に関しては、書かなくても良いが、書いた方がわかりやすい。
 - ➡ 但し、中には restart ファイルに書き込まれないものもあるので、各コマンドのマニュアル (特に、Restriction項) を確認。

【練習2】の結果



(3) 応用例；
散逸粒子動力学（DPD）法

散逸粒子動力学 (DPD) 法

- R. D. Groot & T. J. Warren: J. Chem. Phys. 107 (1997)

$$m \frac{d\mathbf{r}_i}{dt} = \mathbf{v}_i, \frac{d\mathbf{v}_i}{dt} = \mathbf{f}_i$$

$$\mathbf{f}_i = \sum_{j \neq i} (\mathbf{F}_{ij}^C + \mathbf{F}_{ij}^D + \mathbf{F}_{ij}^R)$$

$$\mathbf{F}_{ij}^C = \begin{cases} a_{ij}(1-r_{ij})\hat{\mathbf{r}}_{ij} & (r_{ij} < 1) \\ 0 & (r_{ij} \geq 1) \end{cases}, \quad \mathbf{F}_{ij}^D = -\gamma w^D(r_{ij})(\hat{\mathbf{r}}_{ij} \cdot \mathbf{v}_{ij})\hat{\mathbf{r}}_{ij}, \quad \mathbf{F}_{ij}^R = \sigma w^R(r_{ij})\theta_{ij}\hat{\mathbf{r}}_{ij}$$

+ バネの力 (任意性あり)

$$w^D(r) = [w^R(r)]^2 = \begin{cases} (1-r)^2 & (r < 1) \\ 0 & (r \geq 1) \end{cases}$$

$$\sigma^2 = 2\gamma k_B T$$

$$\langle \theta_{ij}(t) \theta_{kl}(t') \rangle = (\delta_{ik} \delta_{jl} + \delta_{il} \delta_{jk}) \delta(t - t')$$

- 運動量保存 ⇒ 流体力学相互作用あり。

⊇ 高分子系のメソスケール・シミュレーション；相分離ダイナミクス、
モルフォロジー形成などなど

★ ボンドの交差が禁止されていないことに注意。

DPD の使い方

pair_style dpd 1.0 1.0 13459 (左から順に) T_{start} , T_{end} , seed
communicate single vel yes

pair_style dpd とセット

(DPD では力の計算に速度が必要になるため)

pair_coeff 1 1 a_{11} γ r_c
 pair_coeff 1 2 a_{12} γ r_c
 pair_coeff 2 2 a_{22} γ r_c

...
 fix 1 all nve
 ...

a_{ij} の取り方

$a_{ii}\rho = 75 k_B T$, $a_{ij} = a_{ii} + \underline{c(\rho)} X_{ij}$

密度に依る

pair_style dpd で計算される相互作用

$$F^C = Aw(r)$$

$$F^D = -\gamma w^2(r)(\hat{r}_{ij} \bullet \vec{v}_{ij})$$

$$F^R = \sigma w(r)\alpha(\Delta t)^{-1/2}$$

$$w(r) = 1 - r/r_c$$

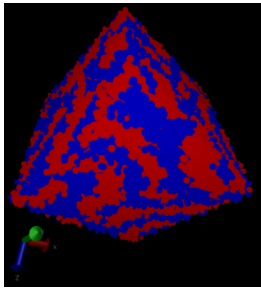
$$\sigma^2 = 2 \gamma k_B T$$

DPD の使い方

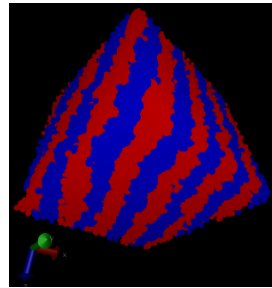
- まとめると、`pair_style` を `lj` → `dpd` へと変更するだけであり、他はこれまでと全く同様の入力スクリプトで良い。
 - ▷ 通常の DPD では、 $m = r_c = k_B T = 1$ とする。
 - ▷ `pair_coeff` では、 a_{ij} , γ , r_c を決める。
 - ➡ a_{ij} は簡単ではない！
 - ➡ 文献では γ ではなく、 σ を与えているものが多いので混同しないようにする。 γ が決まれば、 σ は自動的に決まる (FDT) 。
 - ▷ `pair_style dpd` の時間発展は、`fix nve` で。熱浴の役割をもつ項 (F^D 、 F^R) があるから、`fix langevin` 等の熱浴は要らない。
 - ➡ `pair_style dpd/tstat` コマンドにより、純粋な熱浴として使うこともできる ($F^C = 0$ 、 F^D と F^R のみ計算) 。

高分子系での例

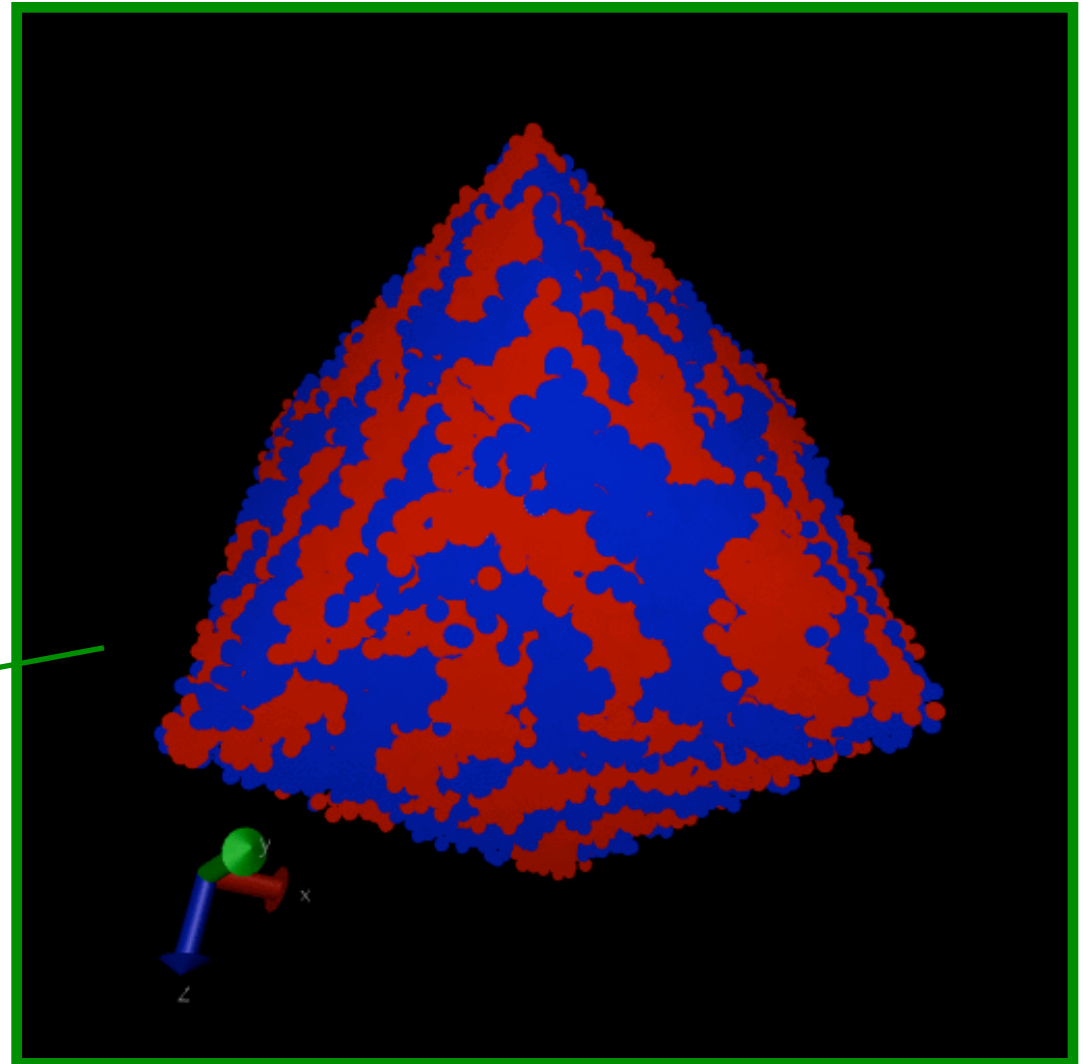
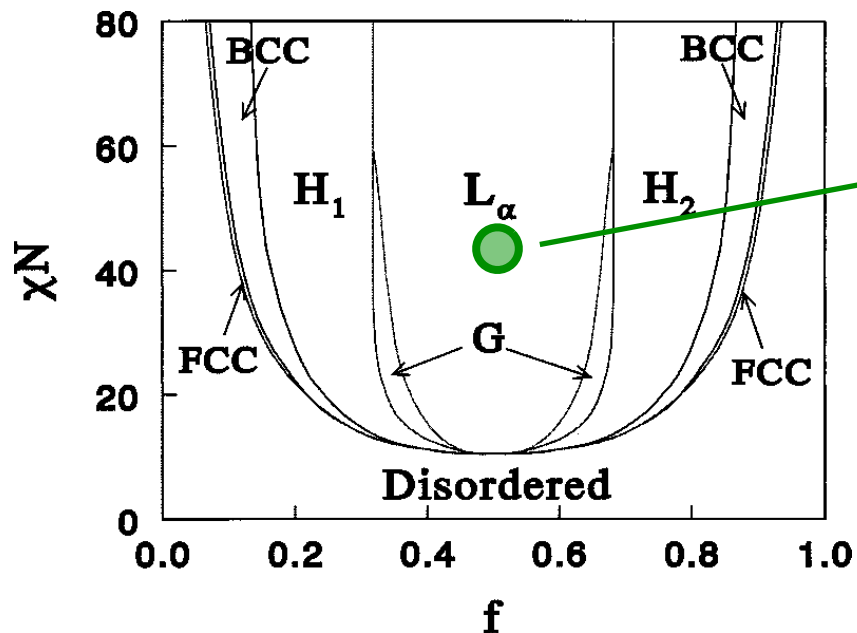
- 対称ブロックコポリマー A_5B_5 ($f = 0.5$, $N_{ch} = 4,000$)



Initial state
@ $50 \times T_{DPD}$



Equilibrium state
@ $2,000 \times T_{DPD}$



【練習3】

- **data.A4B4.Nch50** ($\rho = 3.0$) を読み込み、DPD シミュレーションを行う。A セグメント (`atom-type = 1`) と B セグメント (`atom-type = 2`) をそれぞれ分けて **dump** する。

▷ $T = 1.0$, $a_{11} = a_{22} = 25$, $a_{12} = 45$, $\gamma = 6$, $r_c = 1.0$ と設定。

▷ A4B4 ポリマーのバネは、`bond_style harmonic` を設定：

➡ `bond_coeff 1 80 0.86` # 数値は、bond-ID, K_{sp} , r_{eq} を表す。

▷ 同じ atom-ID の粒子をグループ化、それを **dump** する例：

➡ `group Ax type 1` # `atom-type = 1` に Ax という group-ID を設定

➡ `dump 1 Ax xyz dump.Ax.xyz` # group-ID = Ax の粒子のみ xyz 形式出力

- 先ほどと同様にして、**VMD** で相分離の様子を観る。

▷ 例えば、`dump.{A4, B4}.xyz` を分けて読み込ませれば色分け可能。

【練習3】の入力スクリプト

```
###  
# Section (A)  
units lj  
dimension 3  
boundary p p p  
atom_style bond
```

```
###  
# Section (B)  
read_data data.A4B4.Nch50  
group Ax type 1  
group By type 2
```

```
pair_style dpd 1.0 1.0 13459  
communicate single vel yes  
bond_style harmonic
```

```
###  
# Section (C)  
pair_coeff 1 1 25 6.0 1.0  
pair_coeff 1 2 45 6.0 1.0  
pair_coeff 2 2 25 6.0 1.0  
bond_coeff 1 80.0 0.86
```

Input Data for DPD

400 atoms
350 bonds
2 atom types
1 bond types

... (略) ...

Masses

1 1.00000
2 1.00000

Atoms

1 1 1 -0.063 -1.89 1.49 0 0 0
2 1 1 0.058 1.87 0.664 0 -1 0
... (略) ...
5 1 2 1.25 -1.66 -0.125 0 0 0
6 1 2 1.63 -0.76 -0.314 0 0 0
... (略) ...

Bonds

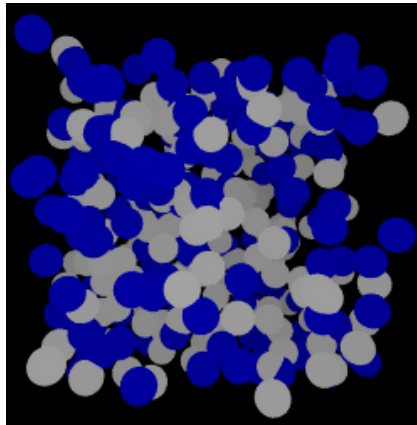
1 1 2 1
2 1 3 2
... (略) ...
5 1 6 5
... (略) ...

```
neighbor 0.5 bin  
neigh_modify delay 1 every 1  
velocity all create 1.0 29308 dist gaussian  
timestep 0.02  
run_style verlet  
fix 1 all nve
```

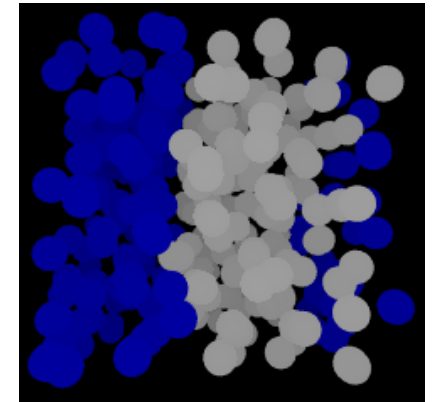
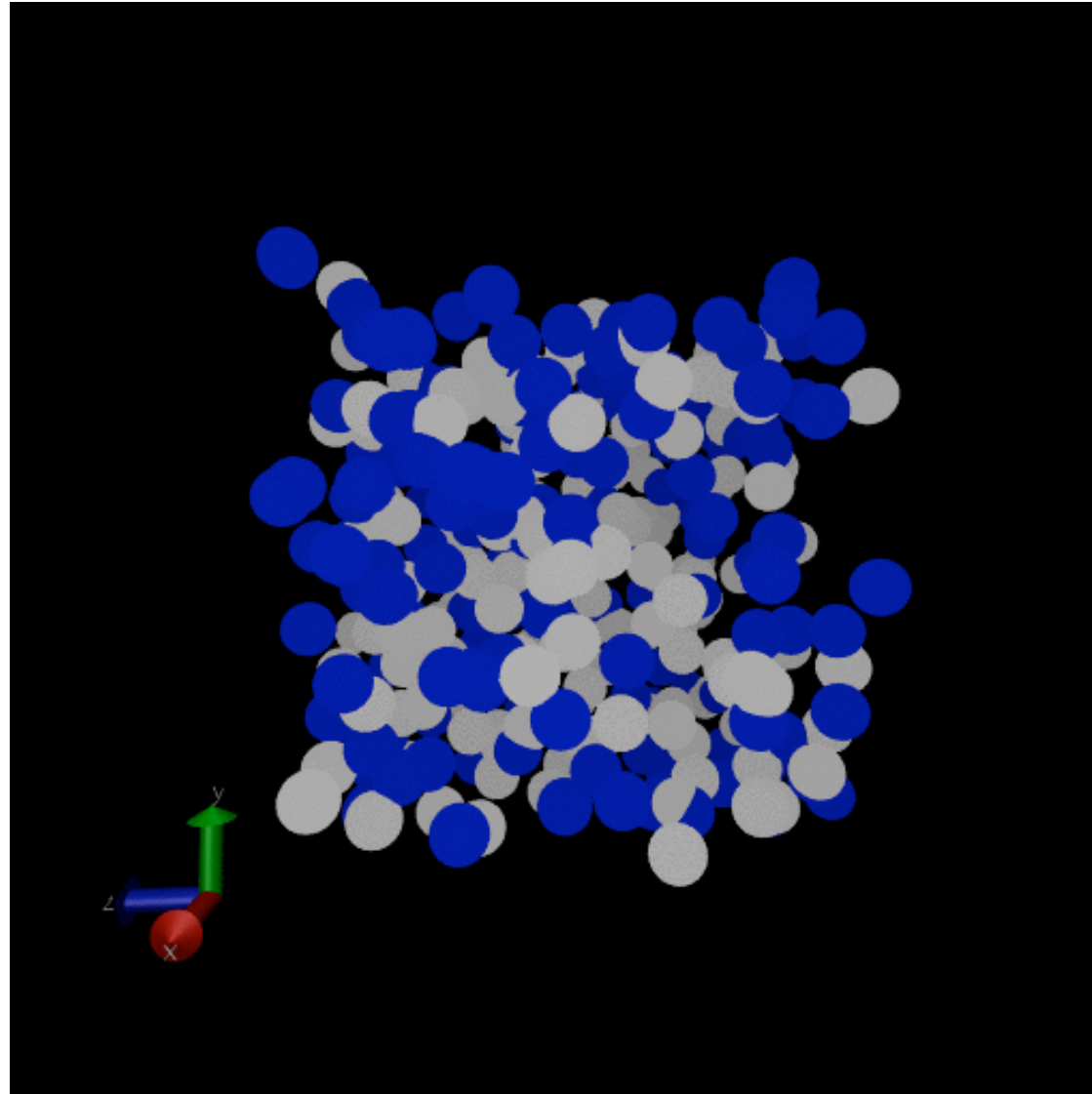
```
###  
# Section (D)  
dump 1 Ax xyz 10 dump.A4.xyz  
dump_modify 1 sort id  
dump 2 By xyz 10 dump.B4.xyz  
dump_modify 2 sort id
```

```
thermo 1000  
thermo_style custom step cpu temp  
restart 5000 restart.A4B4.Nch50  
run 5000
```

【練習3】の結果



$t = 0$

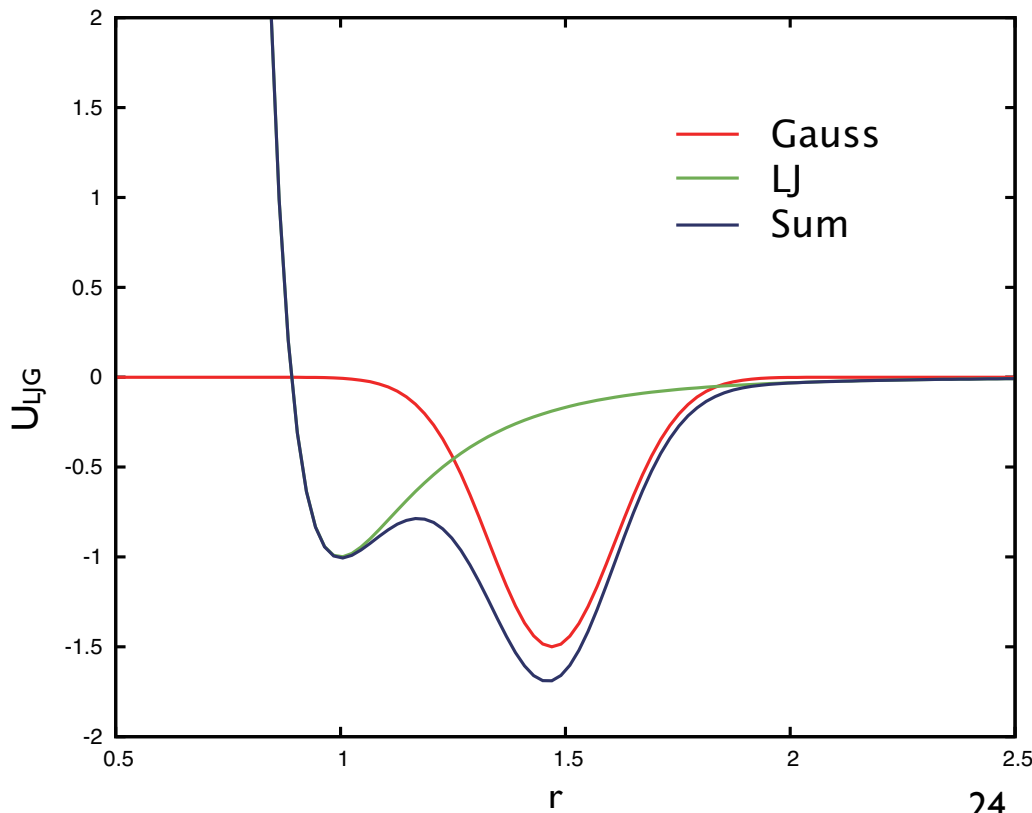


$t = 100 \times T_{\text{DPD}}$

**(4) 高分子から離れて；
1成分ガラス系**

1成分ガラス系

- V. V. Hoang & T. Odagaki: Physica B 403 (2008) 3910.
 - ▷ 1成分ガラス（金属ガラス）のモデル系として提案。
 - ▷ ポテンシャルは、**係数が若干異なる LJ ポテンシャル**と **Gauss 型ポテンシャル**から成る2重底。



デフォルトの LJ と少し違う

$$U_{\text{LJG}} = \frac{\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - 2 \left(\frac{\sigma}{r} \right)^6 \right]}{-1.5\epsilon \exp \left[-\frac{(r - 1.47\sigma)^2}{0.04\sigma^2} \right]}$$

LJG ポテンシャルを使うには...①

- LJG ポテンシャル = LJ + Gauss 型

▷ LAMMPS の LJ : `pair_lj_cut.{cpp, h}` $\Rightarrow E = 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^6 \right]$

▷ Gauss 型 : `pair_gauss_cut.{cpp, h}` $\Rightarrow E = \frac{H}{\sigma_h \sqrt{2\pi}} \exp \left[-\frac{(r-r_{mh})^2}{2\sigma_h^2} \right]$

➡ `pair_coeff` の入力パラメーターを適切に設定し、両者を足せば、LJG ポテンシャルにできる。

▷ 2つの `pair_style` を組み合わせる方法 : `pair_style hybrid/overlay`

```
pair_style hybrid/overlay ljg/cut 2.5 gauss/cut 2.5
pair_coeff 1 1 lj/cut 1.0 0.890898718 2.5
pair_coeff 1 1 gauss/cut -0.53173615527166 1.47 0.14142135623731
```

H, **r_{mh},** **σ_h**

★ 但し、hybrid/overlay の情報は、restart ファイルに書き込まれないので、毎回入力スクリプトに書く必要がある。

LJG ポテンシャルを使うには...②

★方針； pair_coeff 1 1 ϵ σ H r_{\min} σ_h r_c と1つで記述したい：

```
pair_style ljg/cut r_c  
pair_coeff 1 1  $\epsilon$   $\sigma$  H  $r_{\min}$   $\sigma_h$   $r_c$ 
```

1. pair_lj_cut.{h, cpp} を雛形として、コピー pair_ljg_cut.{h, cpp} を用意。
2. 元のポテンシャルの名前がついている箇所を新しい名前に修正。
3. ソースプログラムの global 内部変数を増やす。
 - ➡ ϵ , σ , H, r_{\min} , σ_h , r_c に対応する変数をソースプログラム中に（無ければ）加える。
4. pair_coeff コマンドの引数 arg の数を増やす（この例では、8 個必要）。
 - ➡ cpp ファイル中で、読み込める arg の数を増やし、変数と arg を対応させる。
5. 力の計算部分を自分用に書き換え、読み込み・書き込み部分も編集。

LJG ポテンシャルを使うには...②

2. 元のポテンシャルの名前がついている箇所を新しい名前に修正

▷ pair_ljg_cut.h :

- ➡ LJCut --> LJGCut
- ➡ LMP_PAIR_LJ_CUT_H --> LMP_PAIR_LJG_CUT_H
- ➡ PairStyle(lj/cut, ...) --> PairStyle(ljg/cut, ...)

ljg/cut が pair_style 名になる。

▷ pair_ljg_cut.cpp での書き換え :

- ➡ LJCut --> LJGCut
- ➡ pair_lj_cut.h --> pair_ljg_cut.h
- ➡ respa_enable = 1 --> respa_enable = 0

RESPA 法は使わないとする。

LJG ポテンシャルを使うには...②

3. ソースプログラムの global 内部変数を増やす。

▷ pair_ljg_cut.h

```
protected:  
double cut_global;  
double **cut;  
double **epsilon,**sigma;  
double **lj1,**lj2,**lj3,**lj4,**offset;  
double *cut_respa;
```



```
protected:  
double cut_global;  
double **cut;  
double **epsilon,**sigma;  
double **lj1,**lj2,**lj3,**lj4,**offset;  
double *cut_respa;  
double **hgauss,**sigmah,**rmh;  
double **pgauss;
```

▷ pair_ljg_cut.cpp

```
memory->destroy(cut);  
memory->destroy(epsilon);  
memory->destroy(sigma);  
memory->destroy(lj1);  
memory->destroy(lj2);  
memory->destroy(lj3);  
memory->destroy(lj4);  
memory->destroy(offset);
```



```
memory->destroy(cut);  
memory->destroy(epsilon);  
memory->destroy(sigma);  
memory->destroy(lj1);  
memory->destroy(lj2);  
memory->destroy(lj3);  
memory->destroy(lj4);  
memory->destroy(hgauss);  
memory->destroy(sigmah);  
memory->destroy(rmh);  
memory->destroy(pgauss);  
memory->destroy(offset);
```

```
memory->create(cut,n+1,n+1,"pair:cut");  
memory->create(epsilon,n+1,n+1,"pair:epsilon");  
memory->create(sigma,n+1,n+1,"pair:sigma");  
memory->create(lj1,n+1,n+1,"pair:lj1");  
memory->create(lj2,n+1,n+1,"pair:lj2");  
memory->create(lj3,n+1,n+1,"pair:lj3");  
memory->create(lj4,n+1,n+1,"pair:lj4");  
memory->create(offset,n+1,n+1,"pair:offset");
```



```
memory->create(cut,n+1,n+1,"pair:cut");  
memory->create(epsilon,n+1,n+1,"pair:epsilon");  
memory->create(sigma,n+1,n+1,"pair:sigma");  
memory->create(lj1,n+1,n+1,"pair:lj1");  
memory->create(lj2,n+1,n+1,"pair:lj2");  
memory->create(lj3,n+1,n+1,"pair:lj3");  
memory->create(lj4,n+1,n+1,"pair:lj4");  
memory->create(hgauss,n+1,n+1,"pair:hgauss");  
memory->create(sigmah,n+1,n+1,"pair:sigmah");  
memory->create(rmh,n+1,n+1,"pair:rmh");  
memory->create(pgauss,n+1,n+1,"pair:pgauss");  
memory->create(offset,n+1,n+1,"pair:offset");
```

LJG ポテンシャルを使うには...②

4. pair_coeff コマンドの引数 arg の数を増やす。

▷ pair_ljg_cut.cpp

```
void PairLJGCut::coeff(int narg, char **arg)
{
    if (narg < 4 || narg > 5)
        error->all(FLError, "Incorrect args for pair coefficients");
    if (!allocated) allocate();
    int ilo, ihi, jlo, jhi;
    force->bounds(arg[0], atom->ntypes, ilo, ihi);
    force->bounds(arg[1], atom->ntypes, jlo, jhi);
    double epsilon_one = force->numeric(FLError, arg[2]);
    double sigma_one = force->numeric(FLError, arg[3]);
    double cut_one = cut_global;
    if (narg == 5) cut_one = force->numeric(FLError, arg[4]);
    int count = 0;
    for (int i = ilo; i <= ihi; i++) {
        for (int j = MAX(jlo, i); j <= jhi; j++) {
            epsilon[i][j] = epsilon_one;
            sigma[i][j] = sigma_one;
            cut[i][j] = cut_one;
            setflag[i][j] = 1;
            count++;
        }
    }
    ...
}
```



```
void PairLJGCut::coeff(int narg, char **arg)
{
    if (narg < 7 || narg > 8)
        error->all(FLError, "Incorrect args for pair coefficients");
    if (!allocated) allocate();
    int ilo, ihi, jlo, jhi;
    force->bounds(arg[0], atom->ntypes, ilo, ihi);
    force->bounds(arg[1], atom->ntypes, jlo, jhi);
    double epsilon_one = force->numeric(FLError, arg[2]);
    double sigma_one = force->numeric(FLError, arg[3]);
    double hgauss_one = force->numeric(FLError, arg[4]);
    double rmh_one = force->numeric(FLError, arg[5]);
    double sigmah_one = force->numeric(FLError, arg[6]);
    double cut_one = cut_global;
    if (narg == 8) cut_one = force->numeric(FLError, arg[7]);
    int count = 0;
    for (int i = ilo; i <= ihi; i++) {
        for (int j = MAX(jlo, i); j <= jhi; j++) {
            epsilon[i][j] = epsilon_one;
            sigma[i][j] = sigma_one;
            hgauss[i][j] = hgauss_one;
            sigmah[i][j] = sigmah_one;
            rmh[i][j] = rmh_one;
            cut[i][j] = cut_one;
            setflag[i][j] = 1;
            count++;
        }
    }
    ...
}
```

LJG ポテンシャルを使うには...②

5. 力の計算部分の書き換え (pair_ljg_cut.cpp)

```
void PairLJGCut::compute(int eflag, int vflag)
...
if (rsq < cutsq[iatype][jatype]) {
    /* LJ term */
    r2inv = 1.0/rsq;
    r6inv = r2inv*r2inv*r2inv;
    forcelj = r6inv * (lj1[iatype][jatype]*r6inv - lj2[iatype][jatype]);
    fpair = factor_lj*forcelj*r2inv;
    f[i][0] += delx*fpair;
    f[i][1] += dely*fpair;
    f[i][2] += delz*fpair;
    if (newton_pair || j < nlocal) {
        f[j][0] -= delx*fpair;
        f[j][1] -= dely*fpair;
        f[j][2] -= delz*fpair;
    }
    ...
    if (eflag) {
        evdwl = r6inv*(lj3[iatype][jatype]*r6inv-lj4[iatype][jatype]) -
            offset[iatype][jatype];
        evdwl *= factor_lj;
    }
    ...
}
```



```
void PairLJGCut::compute(int eflag, int vflag)
...
if (rsq < cutsq[iatype][jatype]) {
    /* LJ term */
    r2inv = 1.0/rsq;
    r6inv = r2inv*r2inv*r2inv;
    forcelj = r6inv * (lj1[iatype][jatype]*r6inv - lj2[iatype][jatype]);
    fpair = factor_lj*forcelj*r2inv;
    /* Add Gauss term */
    r = sqrt(rsq);
    rexp = (r-rmh[iatype][jatype])/sigmah[iatype][jatype];
    ugauss = pgauss[iatype][jatype]*exp(-0.5*rexp*rexp);
    fpair += factor_lj*rexp/r*ugauss/sigmah[iatype][jatype];
    f[i][0] += delx*fpair;
    f[i][1] += dely*fpair;
    f[i][2] += delz*fpair;
    if (newton_pair || j < nlocal) {
        f[j][0] -= delx*fpair;
        f[j][1] -= dely*fpair;
        f[j][2] -= delz*fpair;
    }
    ...
    if (eflag) {
        evdwl = r6inv*(lj3[iatype][jatype]*r6inv-lj4[iatype][jatype]) + ugauss -
            offset[iatype][jatype];
        evdwl *= factor_lj;
    }
    ...
}
```

* void PairLJG::compute(int eflag, int vflag) も同様なので略。

```

double PairLJGCut::init_one(int i, int j)
{
    ...
    lj1[i][j] = 48.0 * epsilon[i][j] * pow(sigma[i][j],12.0);
    lj2[i][j] = 24.0 * epsilon[i][j] * pow(sigma[i][j],6.0);
    lj3[i][j] = 4.0 * epsilon[i][j] * pow(sigma[i][j],12.0);
    lj4[i][j] = 4.0 * epsilon[i][j] * pow(sigma[i][j],6.0);
    ...
    lj1[j][i] = lj1[i][j];
    lj2[j][i] = lj2[i][j];
    lj3[j][i] = lj3[i][j];
    lj4[j][i] = lj4[i][j];
    offset[j][i] = offset[i][j];
    cut[j][i] = cut[i][j];
    ...
    etail_ij = 8.0*MY_PI*all[0]*all[1]*epsilon[i][j] *
        sig6 * (sig6 - 3.0*rc6) / (9.0*rc9);
    ptail_ij = 16.0*MY_PI*all[0]*all[1]*epsilon[i][j] *
        sig6 * (2.0*sig6 - 3.0*rc6) / (9.0*rc9);
    ...
}

```



```

double PairLJGCut::init_one(int i, int j)
{
    ...
    pgauss[i][j] = hgauss[i][j] / sqrt(MY_2PI) / sigmah[i][j];
    /*----- Default LJ prefactors -----
    lj1[i][j] = 48.0 * epsilon[i][j] * pow(sigma[i][j],12.0);
    lj2[i][j] = 24.0 * epsilon[i][j] * pow(sigma[i][j],6.0);
    lj3[i][j] = 4.0 * epsilon[i][j] * pow(sigma[i][j],12.0);
    lj4[i][j] = 4.0 * epsilon[i][j] * pow(sigma[i][j],6.0);
    */
    /*@@@@ Modified LJ prefactors for 1-component glass @@@@@*/
    lj1[i][j] = 12.0 * epsilon[i][j] * pow(sigma[i][j],12.0);
    lj2[i][j] = 12.0 * epsilon[i][j] * pow(sigma[i][j],6.0);
    lj3[i][j] = epsilon[i][j] * pow(sigma[i][j],12.0);
    lj4[i][j] = 2.0 * epsilon[i][j] * pow(sigma[i][j],6.0);
    ...
    lj1[j][i] = lj1[i][j];
    lj2[j][i] = lj2[i][j];
    lj3[j][i] = lj3[i][j];
    lj4[j][i] = lj4[i][j];
    hgauss[j][i] = hgauss[i][j];
    sigmah[j][i] = sigmah[i][j];
    rmh[j][i] = rmh[i][j];
    pgauss[j][i] = pgauss[i][j];
    offset[j][i] = offset[i][j];
    cut[j][i] = cut[i][j];
    ...
    /*----- Default LJ -----
    etail_ij = 8.0*MY_PI*all[0]*all[1]*epsilon[i][j] *
        sig6 * (sig6 - 3.0*rc6) / (9.0*rc9);
    ptail_ij = 16.0*MY_PI*all[0]*all[1]*epsilon[i][j] *
        sig6 * (2.0*sig6 - 3.0*rc6) / (9.0*rc9);
    */
    /*@@@@ Modified LJ @@@@@*/
    etail_ij = 2.0*MY_PI*all[0]*all[1]*epsilon[i][j] *
        sig6 * (sig6 - 6.0*rc6) / (9.0*rc9);
    ptail_ij = 8.0*MY_PI*all[0]*all[1]*epsilon[i][j] *
        sig6 * (sig6 - 3.0*rc6) / (9.0*rc9);
    ...
}

```

LJG ポテンシャルを使うには...②

5. 読み込み・書き込み部分 (pair_ljg_cut.cpp)

```
void PairLJGCut::write_restart(FILE *fp)
{
    ...
    if (setflag[i][j]) {
        fwrite(&epsilon[i][j], sizeof(double), 1, fp);
        fwrite(&sigma[i][j], sizeof(double), 1, fp);
        fwrite(&cut[i][j], sizeof(double), 1, fp);
    }
    ...
}

void PairLJGCut::read_restart(FILE *fp)
{
    ...
    if (me == 0) {
        fread(&epsilon[i][j], sizeof(double), 1, fp);
        fread(&sigma[i][j], sizeof(double), 1, fp);
        fread(&cut[i][j], sizeof(double), 1, fp);
    }
    MPI_Bcast(&epsilon[i][j], 1, MPI_DOUBLE, 0, world);
    MPI_Bcast(&sigma[i][j], 1, MPI_DOUBLE, 0, world);
    MPI_Bcast(&cut[i][j], 1, MPI_DOUBLE, 0, world);
    ...
}
```



```
void PairLJGCut::write_restart(FILE *fp)
{
    ...
    if (setflag[i][j]) {
        fwrite(&epsilon[i][j], sizeof(double), 1, fp);
        fwrite(&sigma[i][j], sizeof(double), 1, fp);
        fwrite(&hgauss[i][j], sizeof(double), 1, fp);
        fwrite(&rmh[i][j], sizeof(double), 1, fp);
        fwrite(&sigmah[i][j], sizeof(double), 1, fp);
        fwrite(&cut[i][j], sizeof(double), 1, fp);
    }
    ...
}

void PairLJGCut::read_restart(FILE *fp)
{
    ...
    if (me == 0) {
        fread(&epsilon[i][j], sizeof(double), 1, fp);
        fread(&sigma[i][j], sizeof(double), 1, fp);
        fread(&hgauss[i][j], sizeof(double), 1, fp);
        fread(&rmh[i][j], sizeof(double), 1, fp);
        fread(&sigmah[i][j], sizeof(double), 1, fp);
        fread(&cut[i][j], sizeof(double), 1, fp);
    }
    MPI_Bcast(&epsilon[i][j], 1, MPI_DOUBLE, 0, world);
    MPI_Bcast(&sigma[i][j], 1, MPI_DOUBLE, 0, world);
    MPI_Bcast(&hgauss[i][j], 1, MPI_DOUBLE, 0, world);
    MPI_Bcast(&rmh[i][j], 1, MPI_DOUBLE, 0, world);
    MPI_Bcast(&sigmah[i][j], 1, MPI_DOUBLE, 0, world);
    MPI_Bcast(&cut[i][j], 1, MPI_DOUBLE, 0, world);
    ...
}
```

LJG ポテンシャルを使うには...②

- make の際、pair_* は自動的にコンパイルされるので、用意した pair_ljg_cut.{cpp, h} を LAMMPS の /src にもっていき、再コンパイル (make) する。
- 入力スクリプトでのコマンド：

```
pair_style ljg/cut 2.5
pair_coeff 1 1 1.0 1.0 -0.5317... 1.47 0.1414... 2.5
           ε   σ      H      rmin   σh      rc
```

LJG のポテンシャル：
$$U_{\text{LJG}} = \epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - 2 \left(\frac{\sigma}{r} \right)^6 \right] - 1.5\epsilon \exp \left[- \frac{(r - 1.47\sigma)^2}{0.04\sigma^2} \right]$$

用意した pair_style：
$$U_{\text{LJG}} = \epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - 2 \left(\frac{\sigma}{r} \right)^6 \right] + \frac{H}{\sigma_h \sqrt{2\pi}} \exp \left[- \frac{(r - r_{\text{mh}})^2}{2\sigma_h^2} \right]$$

【練習4】

- 用意した restart.LJG.N256 (数密度 : 0.80) を読み、NVT シミュレーション ($T = 2.5$ の Langevin 熱浴) を行う。
pair_style ljg/cut を使う。また、動径分布関数を計算する。

▷ restart.LJG.N256 : $T = 2.5$ で平衡化させてあります。

▷ 動径分布関数を計算するコマンド例 :

➡ compute **myRDF** all rdf 100
fix 3 all ave/time 100 1 200 c_**myRDF** file rdf.dat &
mode vector ave running

- Langevin 熱浴を使って、温度を $T = 2.5$ から 1.5, 1.0 まで下げ、各温度で動径分布関数を計算してその温度変化をみる。

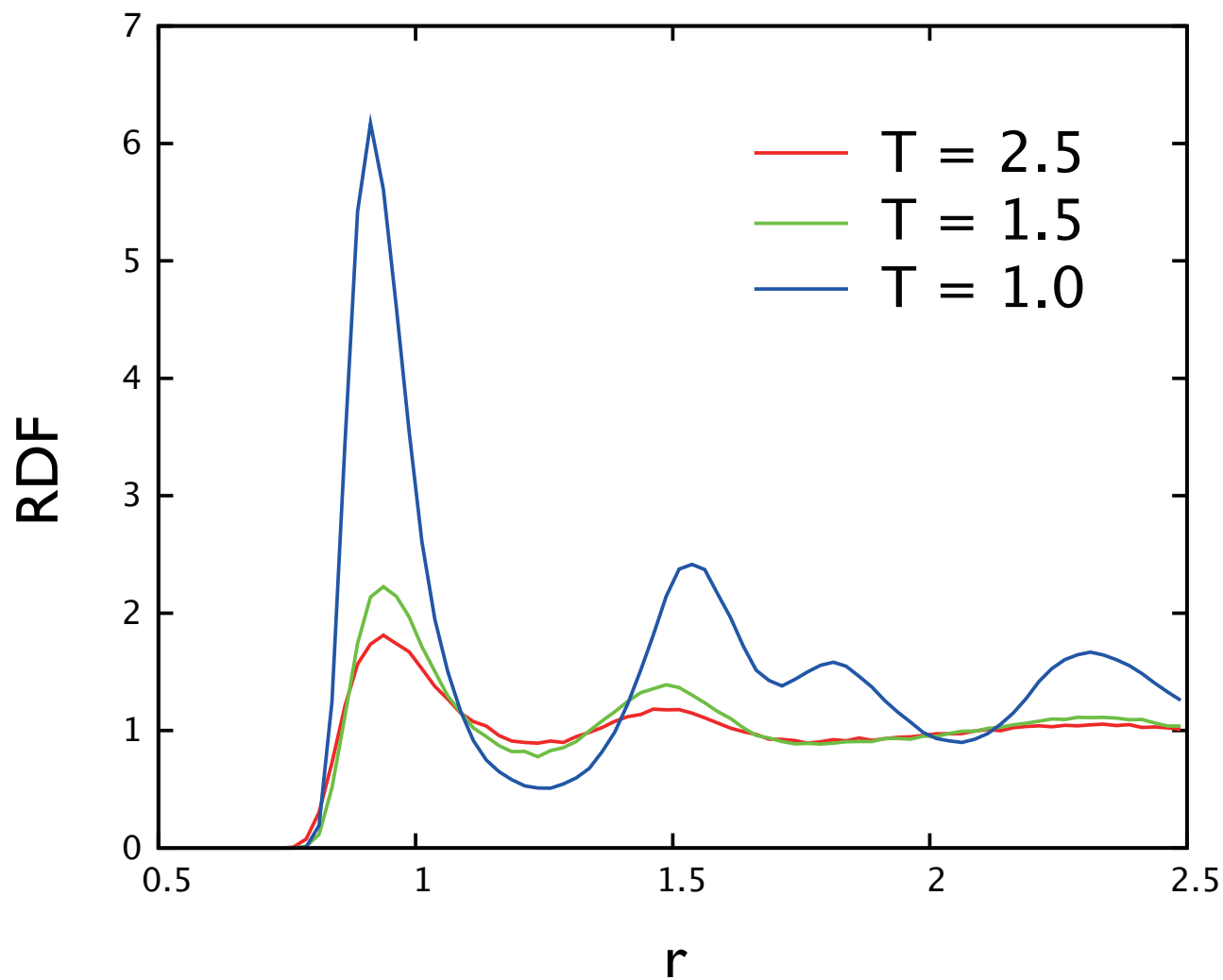
▷ 温度は fix langevin コマンドのパラメーター T_{start} , T_{end} により調整可能。

【練習4】の入力スクリプト

```
###  
# Section (C)  
# Use "ljg/cut"  
pair_style ljg/cut 2.5  
pair_coeff 1 1 1.0 1.0 -0.53173615527166 1.47 0.14142135623731 2.5  
# Use "hybrid"  
#pair_style hybrid/overlay lj/cut 2.5 gauss/cut 2.5  
#pair_coeff 1 1 lj/cut 1.0 0.890898718 2.5  
#pair_coeff 1 1 gauss/cut -0.53173615527166 1.47 0.14142135623731  
  
...  
fix 1 all nve  
fix 2 all langevin 2.5 1.5 2.0 2943  
....  
run 10000 10000 step かけて、冷却 ; T = 2.5 → 1.5  
  
###  
# Section (C)-(D)  
unfix 2  
fix 3 all langevin 1.5 1.5 2.0 30810  
run 10000  
  
compute myRDF all rdf 100  
fix 4 all ave/time 100 1 200 c_myRDF file rdf.dat mode vector ave running  
run 10000
```

unfix (対応するfix コマンドを消去) してから、
冷却後の T を固定して Langevin 熱浴を。

【練習4】の結果



dump の解析プログラムの例

```
program ana
  implicit none
  integer, parameter :: nmax=200
  real*8 x(nmax), y(nmax), z(nmax)
  real*8 xlo, xhi, ylo, yhi, zlo, zhi, time
  real*8 xtmp, ytmp, ztmp
  integer natoms, itmp, loop, i
  real*8 Re2, Re2tmp, Rg2, Rg2tmp, comx, comy, comz
  Re2 = 0.d0
  Rg2 = 0.d0
  !/////
  ! Read dump file
!! open(20, file='dump.polymer')
  open(20, file='dump.N200')
  open(21, file='time_Re2_Rg2.dat')
  loop = 0
  do
```

```
    read(20, *, end=10000)
    loop = loop + 1
    read(20, *) time
    read(20, *)
    read(20, *) natoms
    read(20, *)
    read(20, *) xlo, xhi
    read(20, *) ylo, yhi
    read(20, *) zlo, zhi
    read(20, *)
    do i=1, natoms
      read(20, *) itmp, xtmp, ytmp, ztmp
      x(itmp) = xtmp
      y(itmp) = ytmp
      z(itmp) = ztmp
    enddo
```

読み込みループ ⇒ 各読み込みで計算

```
!----
! Calc. of Re*Re
Re2tmp = 0.d0
Re2tmp = Re2tmp + ( x(1) - x(natoms) )*( x(1) - x(natoms) )
Re2tmp = Re2tmp + ( y(1) - y(natoms) )*( y(1) - y(natoms) )
Re2tmp = Re2tmp + ( z(1) - z(natoms) )*( z(1) - z(natoms) )
Re2 = Re2 + Re2tmp
!----
! Calc. of Rg*Rg
comx = sum( x(1:natoms) )/dble(natoms)
comy = sum( y(1:natoms) )/dble(natoms)
comz = sum( z(1:natoms) )/dble(natoms)
Rg2tmp = 0.d0
do i=1, natoms
  Rg2tmp = Rg2tmp + ( x(i) - comx )*( x(i) - comx )
  Rg2tmp = Rg2tmp + ( y(i) - comy )*( y(i) - comy )
  Rg2tmp = Rg2tmp + ( z(i) - comz )*( z(i) - comz )
enddo
Rg2 = Rg2 + Rg2tmp/dble(natoms)
!----
! Output instantaneous data
write(21, '(3e12.5)') time, Re2tmp, Rg2tmp/dble(natoms)
enddo
10000 CONTINUE
close(20)
close(21)
!/////
! Average
Re2 = Re2/dble(loop)
Rg2 = Rg2/dble(loop)
!/////
! Output average value
write(*,*) " # of samples, # of atoms, Re2, Rg2"
write(*, '(2i8,2e12.4)') loop, natoms, Re2, Rg2
end program ana
```