# データサイエンス系科目群：シミュレーション実習
# 第 7 回 講義資料

担当：川﨑猛史

名古屋大学大学院理学研究科理学専攻物理科学系・非平衡物理研究室 (R 研)

Last update: May 29, 2023

# 目次

# 目次

# 1. 講義のスケジュール

- 当実習は春 1 期にて実施する．
- 講義資料は各講義予定日当日朝 **11** 時迄にアップロードする．
- スケジュール
  1. 4/17：第 1 回
  2. 4/24：第 2 回
  3. 5/01：第 3 回
  4. 5/08：第 4 回 (週半ばに中間レポート課題公開)
  5. 5/15：第 5 回
  6. 5/22：第 6 回
  7. 5/29：第 7 回 (**5/29**：中間レポート課題提出期限予定)
  8. 6/05：第 8 回 (期末レポート課題公開)
  9. 6/12：第 9 回　（補講）

# 1.1. シラバス

当実習では以下の内容を扱う予定である（進捗に合わせ変更する可能性がある）.

1. 導入
   - C(C++) の使い方 (主に数値計算)
   - Python の使い方 (データ解析と作図)
   - 数値計算の理念
   - 桁落ち
   - 科学計算における無次元化
2. 常微分方程式の数値解法：減衰振動や調和振動子を例に
   - 微分方程式の数値積分
   - オイラー法
3. １粒子系のブラウン運動
   - ランジュバン方程式（確率微分方程式）
   - 正規乱数の生成法
   - オイラー・丸山法
   - 時間平均とアンサンブル平均
   - 拡散係数の計算
4. 多粒子系のブラウン運動
   - 相互作用力の計算方法
   - 非平衡系のシミュレーション：相分離現象を例に
5. 多粒子系の分子動力学シミュレーション
   - 位置ベルレ法と速度ベルレ法
   - 多粒子系における保存則（運動量・エネルギー・角運動量）
6. モンテカルロ法
   - 統計力学の復習
   - マルコフ連鎖モンテカルロ法
   - メトロポリス判定法

# 目次

# 2. 第 6 回自主課題

## 第 6 回自主課題 Langevin 熱浴中の多粒子シミュレーションの実装（相分離現象）

直径 $a$ の 1024 個の円板を周期境界をもつ 1 辺の長さが $L = 40a$ の正方形の平面に分散させる．この時，円盤 $j$ の運動は，Langevin 方程式 $m\frac{\mathrm{d}\mathbf{v}_j(t)}{\mathrm{d}t} = -\zeta\mathbf{v}_j(t) + \mathbf{F}_j^{\mathrm{I}}(t) + \mathbf{F}_j^{\mathrm{B}}(t)$ により駆動されるとする．ここで $\mathbf{F}_j^{\mathrm{B}}(t)$ は，熱揺動力であり，以下の揺動散逸定理 $\langle \mathbf{F}_j^{\mathrm{B}}(t)\mathbf{F}_k^{\mathrm{B}}(t')\rangle = 2k_BT\zeta\delta(t-t')\delta_{jk}\mathbf{1}$ を満たすものとする．$\mathbf{F}_j^{\mathrm{I}}(t)$ は相互作用力であり，以下の Lennard-Jones ポテンシャル

$$U(r_{jk}) = 4\epsilon\left[\left(\frac{a_{jk}}{r_{jk}}\right)^{12} - \left(\frac{a_{jk}}{r_{jk}}\right)^{6}\right] + C_{jk} \quad (r_{jk} < a_{\mathrm{cut}}) \tag{1}$$

により与えられ，カットオフ長は $a_{\mathrm{cut}} = 2.5a$ とする. いま，時間の単位を $t_0 = \sqrt{ma^2/\epsilon}$, 長さの単位を $a$，摩擦係数を $\zeta = \sqrt{m\epsilon/a^2}$ とする．この時，無次元温度 $k_BT/\epsilon$ を様々に変化させた際の，相分離の有無を数値的に観察せよ．

### 解説

自主課題 6 のサンプルプログラム “langevin_many.cpp”をリスト 1 に示したので適宜参照せよ．ここでは, 無次元温度 $T^* = 5.0$ で結晶配置を乱雑に混ぜ，目的の $T^*$(パラメータ $temp$ で指定）に落とす．そして目的温度を $T^* = 0.2, 0.4, 0.6, 1.0$ と変化させた際の様子を図示した（図 **1**）.

# 2. 第 6 回自主課題 (2)

リスト 1: 自主課題 6 のサンプルプログラム “langevin_many.cpp”. 以下の GitHub リポジトリより取得可能[リンク].

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include <iostream>
5  #include <fstream>
6  #include <cfloat>
7  #include "BM.h"
8
9  #define Np 1024
10 #define L 40.0
11 #define tmax 100
12 #define dt 0.01
13 #define temp 0.2
14 #define dim 2
15 #define cut 2.5
16 #define polydispersity 0.0
17
18 void ini_coord_square(double (*x)[dim]){
19   int num_x = (int)sqrt(Np)+1;
20   int num_y = (int)sqrt(Np)+1;
21   int i,j,k=0;
```

# 2. 第 6 回自主課題 (3)

```
22    double shift;
23    for(j=0;j<num_y;j++){
24      for(i=0;i<num_x;i++){
25        x[i+num_x*j][0] = i*L/(double)num_x;
26        x[i+num_x*j][1] = j*L/(double)num_y;
27        k++;
28        if(k==Np)
29          break;
30      }
31         if(k==Np)
32          break;
33    }
34  }
35
36  void set_diameter(double *a){
37    for(int i=0;i<Np;i++)
38      a[i]=1.0+polydispersity*gaussian_rand();
39  }
40
41  void p_boundary(double (*x)[dim]){
42    for(int i=0;i<Np;i++)
43      for(int j=0;j<dim;j++)
44        x[i][j]-=L*floor(x[i][j]/L);
45  }
```

# 2. 第 6 回自主課題 (4)

```
46
47 void ini_array(double (*x)[dim]){
48   for(int i=0;i<Np;i++)
49     for(int j=0;j<dim;j++)
50       x[i][j]=0.0;
51 }
52
53 void calc_force(double (*x)[dim],double (*f)[dim],double *a){
54   double dx,dy,dr2,dUr,w2,w6,w12,aij;
55
56   ini_array(f);
57
58   for(int i=0;i<Np;i++)
59     for(int j=0;j<Np;j++){
60       if(i<j){
61        dx=x[i][0]-x[j][0];
62        dy=x[i][1]-x[j][1];
63        dx-=L*floor((dx+0.5*L)/L);
64        dy-=L*floor((dy+0.5*L)/L);
65        dr2=dx*dx+dy*dy;
66        if(dr2<cut*cut){
67           aij=0.5*(a[i]+a[j]);
68           w2=aij/dr2;
69           w6=w2*w2*w2;
```

# 2. 第 6 回自主課題 (5)

```
70            w12=w6*w6;
71            dUr=-48.*w12/dr2+24.*w6/dr2;
72            f[i][0]-=dUr*dx;
73            f[j][0]+=dUr*dx;
74            f[i][1]-=dUr*dy;
75            f[j][1]+=dUr*dy;
76            }
77          }
78      }
79  }
80
81  void eom(double (*v)[dim],double (*x)[dim],double (*f)[dim],double temp0){
82    double zeta=1.0;
83    double fluc=sqrt(2.*zeta*temp0*dt);
84    for(int i=0;i<Np;i++)
85      for(int j=0;j<dim;j++){
86        v[i][j]+=-zeta*v[i][j]*dt+f[i][j]*dt+fluc*gaussian_rand();
87        x[i][j]+=v[i][j]*dt;
88      }
89  }
90
91  void output(double (*x)[dim],double *a){
92    char filename[128];
93    std::ofstream file;
```

# 2. 第 6 回自主課題 (6)

```
94    static int j=0;
95    sprintf(filename,"coord_T%.3f_%d.dat",temp,j);
96    file.open(filename);
97    for(int i=0;i<Np;i++)
98      file <<x[i][0]<<"\t"<<x[i][1]<<"\t"<<a[i]<<std::endl;
99    file.close();
100   j++;
101 }
102
103 int main(){
104   double x[Np][dim],v[Np][dim],f[Np][dim],a[Np];
105   double tout=0.0;
106   int j=0;
107   set_diameter(a);
108   ini_coord_square(x);
109   ini_array(v);
110
111   while(j*dt < 10.0){
112     j++;
113     calc_force(x,f,a);
114     eom(v,x,f,5.0);
115   }
116   j=0;
117   while(j*dt < tmax){
```

# 2. 第 6 回自主課題 (7)

```
118       j++;
119       calc_force(x,f,a);
120       eom(v,x,f,temp);
121       p_boundary(x);
122       if(j*dt >= tout){
123         output(x,a);
124         tout+=10.;
125       }
126     }
127     return 0;
128   }
```

# 2. 第 6 回自主課題 (8)

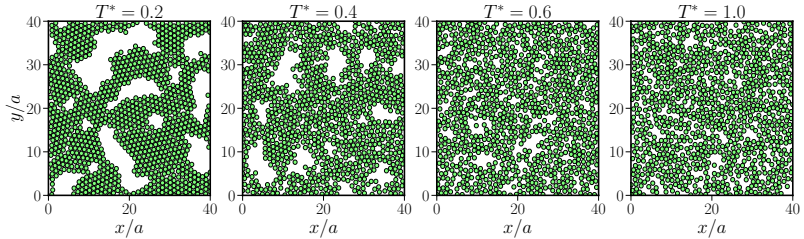


図 1: 周期境界の一辺の長さが $L = 40a$，粒子直径がすべて $a$ の際，無次元温度 $T^*$ を変化させた際の粒子分布．特に低温では相分離が起こっている様子が見て取れる．また，$T^* = 1$ あたりから相分離が起り始めることが分かる．**[補足]** $T^* = 1$ とは $\epsilon = k_{\mathrm{B}}T$ であり，ポテンシャルの谷の深さが熱エネルギーと拮抗している状態．

# 目次

# 3 分子動力学シミュレーション

## 分子動力学シミュレーション（古典 MD）

- 本章では原子・分子が Newton の運動方程式によって駆動される系のシミュレーション：分子動力学 **(Molecular dynamics: MD)** シミュレーションを扱う.
- この様な系は，運動量やエネルギー，角運動量などが保存する，いわゆるハミルトン系と呼ぶ.
- そのため，揺動力や散逸が陽には入らず，温度は一定とならない（定常状態は概ね一定となる）.
- この様な系の統計集団を，$NVE$ アンサンブル，またはミクロカノニカル集団（小正準集団）と呼ぶ

# 3.1 運動方程式とその無次元化

## 運動方程式とその無次元化

- 粒子間ポテンシャル $U$ で他の粒子と相互作用する，質量 $m$ の古典粒子 $j$ の運動方程式は，

$$m\ddot{\mathbf{r}}_j = -\frac{\partial U(\{\mathbf{r}\})}{\partial \mathbf{r}_j} \tag{2}$$

  と表される．
- 運動方程式自体は，Langevin 方程式に比べると，極めてシンプルになり，計算機に乗せる際の無次元化自体は比較的簡単になる．
- 実際，長さの単位を $a$, エネルギーの単位を $\epsilon$, そして時間の単位を $t_0$ とそれぞれおけば, 運動方程式は，˜ をつけた無次元化された変数を用いることで

$$m\frac{a}{t_0^2}\ddot{\tilde{\mathbf{r}}}_j = -\frac{\epsilon}{a}\frac{\partial \tilde{U}(\{\tilde{\mathbf{r}}\})}{\partial \tilde{\mathbf{r}}_j} \tag{3}$$

  となる．

# 3.1 運動方程式とその無次元化 (2)

- そして両辺を $m\frac{a}{t_0^2}$ で割ると

$$\ddot{\tilde{\mathbf{r}}}_j = -\frac{\epsilon t_0^2}{ma^2}\frac{\partial \tilde{U}(\{\tilde{\mathbf{r}}\})}{\partial \tilde{\mathbf{r}}_j} \tag{4}$$

  を得る．

- ここで時間の単位を

$$t_0 = \sqrt{\frac{ma^2}{\epsilon}} \tag{5}$$

  と選ぶと，運動方程式は，

$$\ddot{\tilde{\mathbf{r}}}_j = -\frac{\partial \tilde{U}(\{\tilde{\mathbf{r}}\})}{\partial \tilde{\mathbf{r}}_j} \tag{6}$$

  となり，パラメータフリーの方程式となる．

- 先ほども述べたが，本系では様々な物理量が保存する．そのため，離散化の仕方を工夫しないと，誤差の蓄積により，保存則が破れるなど，誤った計算結果を吐き出すことになる．
- 以下，誤差の蓄積を防止する，シンプレクティックかつ精度の高い離散化（数値積分）の方法を紹介する．

# 3.2 位置ベルレ法

位置ベルレ法

本節では，精度の高い離散化手法であるベルレ法を紹介する．特に，以下に示す方法を位置ベルレ法と呼ぶ (原著論文 [1]，参考文献 [2, 3, 4])．

- ベルレ法は，中心差分法により運動方程式を離散化したものである．
- 中心差分法では，時刻 $t$ に対する 前方差分 および 後方差分 をそれぞれ実行すると

$$\mathbf{r}_j(t+\Delta t) = \mathbf{r}_j(t) + \dot{\mathbf{r}}_j(t)\Delta t + \frac{1}{2!}\ddot{\mathbf{r}}_j(t)(\Delta t)^2 + \frac{1}{3!}\dddot{\mathbf{r}}_j(t)(\Delta t)^3 + O((\Delta t)^4) \tag{7}$$

$$\mathbf{r}_j(t-\Delta t) = \mathbf{r}_j(t) - \dot{\mathbf{r}}_j(t)\Delta t + \frac{1}{2!}\ddot{\mathbf{r}}_j(t)(\Delta t)^2 - \frac{1}{3!}\dddot{\mathbf{r}}_j(t)(\Delta t)^3 + O((\Delta t)^4). \tag{8}$$

となる．

# 3.2 位置ベルレ法 (2)

- 式 (7)+式 (8) から，

$$\begin{aligned} \mathbf{r}_j(t+\Delta t) &= 2\mathbf{r}_j(t) - \mathbf{r}_j(t-\Delta t) + \ddot{\mathbf{r}}_j(t)(\Delta t)^2 + O((\Delta t)^4), \\ &= 2\mathbf{r}_j(t) - \mathbf{r}_j(t-\Delta t) + \underbrace{\frac{\mathbf{F}_j(t)}{m}}_{(*)\text{very small}}(\Delta t)^2 + O((\Delta t)^4) \end{aligned} \tag{9}$$

  を得る．

- 同時に，式 (7)-式 (8) から，速度に関する関係式

$$\dot{\mathbf{r}}_j(t) = \frac{\mathbf{r}_j(t+\Delta t) - \mathbf{r}_j(t-\Delta t)}{2\Delta t} + O((\Delta t)^2) \tag{10}$$

  を得る．

- これを解くことにより，位置に関しては $O((\Delta t)^3)$ の精度，そして速度に関しては $O(\Delta t)$ の精度が得られる．ここで，速度の精度は高くないが，誤差を蓄積する構造になっていないため問題がない．時間発展を担う位置の方程式の精度が高いため，誤差の蓄積が抑えられる．この様な数値積分手法を位置ベルレ法という．

- 一方，位置ベルレ法には数値計算上の問題を有する．

- 式 (9) における (*) で示した項が他の項に比べて極端に小さくなる．

# 3.2 位置ベルレ法 (3)

- これによる桁落ち・情報落ちのリスクが生じるため，この位置ベルレ法を一般的には分子動力学計算では用いられない．
- 数理的な構造は同等であるが，この情報落ちのリスクを回避した手法として速度ベルレ法がある．以下速度ベルレ法について説明する．

# 3.3 速度ベルレ法

速度ベルレ法 [2, 3, 4]

- ここでは数値的な情報落ちを回避したベルレ法（速度ベルレ法）について説明する．

# 3.3 速度ベルレ法 (2)

- まず式 (9) は以下の様に変形可能である.

$$
\begin{aligned}
\mathbf{r}_j(t+\Delta t) &= 2\mathbf{r}_j(t) - \mathbf{r}_j(t-\Delta t) + \frac{\mathbf{F}_j(t)}{m}(\Delta t)^2 + O((\Delta t)^4) \\
&= \mathbf{r}_j(t) + \frac{\mathbf{r}_j(t)}{2} + \frac{\mathbf{r}_j(t)}{2} \\
&\quad -\mathbf{r}_j(t-\Delta t) + \frac{\mathbf{F}_j(t)}{m}(\Delta t)^2 + O((\Delta t)^4) \\
&= \mathbf{r}_j(t) + \frac{1}{2}\left[2\mathbf{r}_j(t-\Delta t) - \mathbf{r}_j(t-2\Delta t) + \frac{\mathbf{F}_j(t-\Delta t)}{m}(\Delta t)^2\right] \\
&+ \quad \frac{1}{2}\mathbf{r}_j(t) - \mathbf{r}_j(t-\Delta t) + \frac{\mathbf{F}_j(t)}{m}(\Delta t)^2 + O((\Delta t)^4) \\
&= \mathbf{r}_j(t) + \mathbf{v}_j(t-\Delta t)\Delta t + \frac{(\Delta t)^2}{2m}[\mathbf{F}_j(t) + \mathbf{F}_j(t-\Delta t)] + \frac{(\Delta t)^2}{2m}\mathbf{F}_j(t) + O((\Delta t)^3) \\
&= \mathbf{r}_j(t) + \mathbf{v}_j(t)\Delta t + \frac{(\Delta t)^2}{2m}\mathbf{F}_j(t) + O((\Delta t)^3)
\end{aligned}
\tag{11}
$$

# 3.3 速度ベルレ法 (3)

- ここでは，

$$\frac{1}{2}\left[\mathbf{r}_j(t) - \mathbf{r}_j(t-2\Delta t)\right] = \mathbf{v}_j(t-\Delta t)\Delta t + O((\Delta t)^3) \tag{12}$$

  を用いた．
- すると，以下の関係式

$$\mathbf{v}_j(t) = \mathbf{v}_j(t-\Delta t) + \frac{\Delta t}{2m}[\mathbf{F}_j(t) + \mathbf{F}_j(t-\Delta t)] + O((\Delta t)^3) \tag{13}$$

  が得られる．
- 時間発展の主計算を速度に関して行うことで情報落ちを回避する．
- また，ここでの速度の精度は $O((\Delta t)^2)$ となり精度が高い．
- これより，速度ベルレ法における重要な式をまとめたものが以下の様になる．

# 3.3 速度ベルレ法 (4)

## （まとめ）速度ベルレ法

$$\mathbf{v}_j(t+\Delta t) = \mathbf{v}_j(t) + \frac{\Delta t}{2m}\{\mathbf{F}_j(t+\Delta t) + \mathbf{F}_j(t)\} \tag{14}$$

$$\mathbf{r}_j(t+\Delta t) = \mathbf{r}_j(t) + \mathbf{v}_j(t)\Delta t + \frac{(\Delta t)^2}{2m}\mathbf{F}_j(t) \tag{15}$$

実際のコーディングの際は以下の通り行えば，極めて効率よく計算できる．

## 速度ベルレ法における計算手順

(1) $\mathbf{r}_j(t+\Delta t) = \mathbf{r}_j(t) + \mathbf{v}_j(t)\Delta t + \frac{(\Delta t)^2}{2m}\mathbf{F}_j(t)$

(2) $\mathbf{v}'_j(t+\Delta t) = \mathbf{v}_j(t) + \frac{\Delta t}{2m}\mathbf{F}_j(t)$

(3) Using $\mathbf{r}_j(t+\Delta t)$, compute $\mathbf{F}_j(t+\Delta t)$.

(4) $\mathbf{v}_j(t+\Delta t) = \mathbf{v}'_j(t+\Delta t) + \frac{\Delta t}{2m}\mathbf{F}_j(t+\Delta t)$

(5) Return to (1) with updating time.

# 3.3 速度ベルレ法 (5)

## 時間反転対称性

離散化した速度ベルレ法の方程式（式 (15)）から出発して，$t + \Delta t \to t$ に同一軌道を通り戻ることができる．まず，速度は項を移行させることでが成立する．

$$\mathbf{v}_j(t) = \mathbf{v}_j(t + \Delta t) + \frac{-\Delta t}{2m}\{\mathbf{F}_j(t + \Delta t) + \mathbf{F}_j(t)\} \tag{16}$$

さらに位置に関しては，

$$\begin{aligned}
\mathbf{r}_j(t) &= \mathbf{r}_j(t + \Delta t) - \mathbf{v}_j(t)\Delta t - \frac{(\Delta t)^2}{2m}\mathbf{F}_j(t) \\
&= \mathbf{r}_j(t + \Delta t) - \mathbf{v}_j(t + \Delta t)\Delta t + \frac{(\Delta t)^2}{2m}[\mathbf{F}_j(t + \Delta t) + \mathbf{F}_j(t)] - \frac{(\Delta t)^2}{2m}\mathbf{F}_j(t) \\
&= \mathbf{r}_j(t + \Delta t) - \mathbf{v}_j(t + \Delta t)\Delta t + \frac{(\Delta t)^2}{2m}\mathbf{F}_j(t + \Delta t)
\end{aligned} \tag{17}$$

となり反転させることができた．

# 3.4 第 7 回自主課題

## 第 7 回自主課題 分子動力学シミュレーション（古典 MD) の実装

粒子数 $N = 1024$，直径 $a$ の同一円盤粒子からなる一辺の長さ $L = 40a$ の周期境界に閉じ込められた 2 次元系を考える． 今回用いる粒子間ポテンシャルは，斥力のみからなる

$$U(r_{jk}) = \epsilon \left( \frac{a_{jk}}{r_{jk}} \right)^{12} + C_{jk} \quad (r_{jk} < a_{\mathrm{cut}})$$

を採用する．ここでカットオフ長は $a_{\mathrm{cut}} = 3.0a$ とする．ここで，長さの単位を $a$, エネルギーの単位を $\epsilon$, そして時間の単位を $t_0 = \sqrt{ma^2/\epsilon}$ とするとき以下の問いに答えよ．

(1) 自主課題 6 で構築した Langevin 熱浴を用いることにより，無次元温度 $T^* = k_B T/\epsilon = 0.9$ を取る熱平衡状態における各粒子の位置座標 $\{\mathbf{r}_j\}$ と速度 $\{\mathbf{v}_j\}$ を取得せよ．

(2) (1) で取得した座標と速度を初期条件として，分子動力学シミュレーションを実行し，この時，全粒子の運動エネルギーを $K$, ポテンシャルエネルギーを $U$ とするとき，力学的エネルギー $U + K$ が時間に対して保存することを示せ．

自主課題 7 のサンプルプログラム（リストを用いた高速化はしていないもの）“md.cpp”は GitHub リポジトリより取得可能である[リンク]. 適宜参照し学習せよ．

# 3.4 第 7 回自主課題 (2)

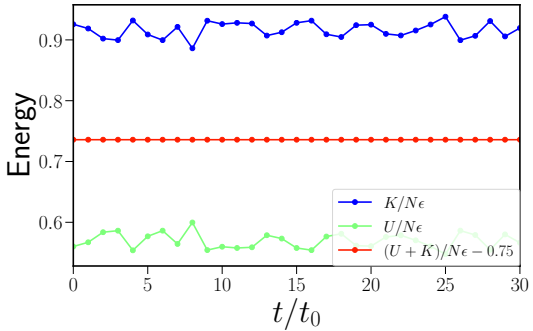


図 2: 自主課題 7(2) の解答例. 力学的エネルギーが保存している様子が見て取れる．なお，ここでは 1 粒子あたりのエネルギーを表示した．

# 目次

# 参考文献・ウェブサイト

[1] Verlet L (1967) Computer ”Experiments” on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules.
Physical Review 159(1):98–103.

[2] Frenkel D, Smit B (2001) Understanding Molecular Simulation: From Algorithms to Applications.
(Elsevier).

[3] Allen MP, Tildesley DJ (2017) Computer Simulation of Liquids: Second Edition.
(Oxford University Press).

[4] Okazaki S, Yoshii N (2011) コンピュータ・シミュレーションの基礎（第 2 版）- 株式会社 化学同人.
(化学同人).