データサイエンス系科目群：シミュレーション実習
第 9 回（補講） 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: June 21, 2024

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. 6/12：第 8 回 (期末レポート課題公開)
  9. **6/19**：第 **9** 回　（補講）

1.1. シラバス

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)
   - 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）
6. モンテカルロ法
   - マルコフ連鎖モンテカルロ法
   - メトロポリス判定法
7. 高速化
   - Verlet 帳簿法
   - 帳簿自動更新

目次

2. MD 計算の高速化

目的

- 今回は，MD 計算の中で最も計算コストが高い，力の計算の高速化を行う．
- 特に短距離しか相互作用しない場合，相互作用しない遠くの粒子の距離を計算することは無意味である．
- この様な無意味な計算を省略するために，近接にある粒子の id を記録し（この様な記録：帳簿をリストと呼ぶ），このリストに記録された粒子同士の相互作用のみ計算するアルゴリズムを考える．
- ここでの近接粒子の定義には任意性をもたせてよく，ポテンシャルカットオフより少し余裕をもたせた距離にある粒子を近接粒子と定義する．
- そうすることで，しばらくの間，同じ帳簿を使い回すことができる．
- 時間が経ち，粒子が動くと，近接領域から出てしまう粒子（あるいは外から侵入してくる粒子）が出てくるので，その時帳簿を更新する．
- 以下，帳簿の作り方および帳簿の更新方法について説明する．

2.1. Verlet 帳簿 (list) 法

ここでは最も簡単な帳簿作成法：Verlet 帳簿法を紹介する [**?**, **?**].

- ここではまず，粒子 j の近接粒子 k を図 **1** の様に，距離 $R_{cut} + \Delta$ 以内にある粒子として定義する．ここで R_{cut} はポテンシャルカットオフ長，Δ は **skin(buffer)** 長である。
- これを実現する最も基本的な方法である Verlet 帳簿法では，まず全粒子の距離を計算し，起点となる粒子 j から $R_{cut} + \Delta$ 以内にある近接粒子 k を探すというものである．
- 近傍リストが決まれば、しばらく (n_{list} ステップ分) は同じリストを使って相互作用を計算する．
- その際，Δ と n_{list} の関係に対して最適な組み合わせを見つける必要がある．
- Verlet 帳簿法のサンプルプログラムをリスト 1 に載せた．
- これにより生成される帳簿配列の構造についてリスト 2 に纏めた．

2.1. Verlet 帳簿 (list) 法 (2)

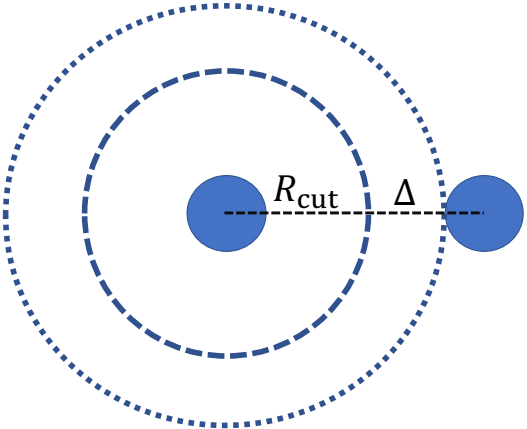


図 1: R_{cut} をカットオフ長，Δ をスキンサイズとして，粒子 j の最も近い粒子 k のリストを作成する．

2.1. Verlet 帳簿 (list) 法 (3)

リスト 1: サンプルプログラム (サブルーチン)

```
1  #define Np 1024
2  #define Nn 100
3  #define L 40.0
4  #define dim 2
5  #define cut 3.0
6  #define skin 1.0
7
8  void list_verlet(int (*list)[Nn],double (*x)[dim]){
9    double dx,dy,dr2;
10   double thresh=cut+skin;
11   for(int i=0;i<Np;i++)
12     for(int j=0;j<Nn;j++)
13       list[i][j]=0;
14
15   for(int i=0;i<Np;i++)
16     for(int j=0;j<Np;j++){
17       if(j>i){
18         dx=x[i][0]-x[j][0];
19         dy=x[i][1]-x[j][1];
20         dx-=L*floor((dx+0.5*L)/L);
21         dy-=L*floor((dy+0.5*L)/L);
22         dr2=dx*dx+dy*dy;
```

2.1. Verlet 帳簿 (list) 法 (4)

```
23          if(dr2<thresh*thresh){
24            list[i][0]++;
25            list[i][(int)list[i][0]]=j;
26          }
27        }
28      }
29  }
```

帳簿配列は以下のようになる．

リスト 2: 帳簿配列の構造

```
1  list[i][0]: The number of the neighboring particles of particle i
2  list[i][1]: Neighboring particle id of particle i (1st)
3  list[i][2]: Neighboring particle id of particle i (2nd)
4  list[i][3]: Neighboring particle id of particle i (3rd)
5  ....
6  list[i][list[i][0]]: Neighboring particle id of particle i (list[i][0]th)
```

以下，力の計算で帳簿配列を用いる場合

2.1. Verlet 帳簿 (list) 法 (5)

リスト 3: 帳簿配列を用いた力の計算

```
#define Np 1024
#define Nn 100
#define L 40.0
#define dim 2
#define cut 3.0
#define skin 1.0

void calc_force(double (*x)[dim],double (*f)[dim],double *a,double *U,int (*list)[Nn]){
  double dx,dy,dr2,dUr,w2,w6,w12,aij;
  double Ucut=1./pow(cut,12);
  ini_array(f);
  *U=0;
  for(int i=0;i<Np;i++)
    for(int j=1;j<=list[i][0];j++){
      dx=x[i][0]-x[list[i][j]][0];
      dy=x[i][1]-x[list[i][j]][1];
      dx-=L*floor((dx+0.5*L)/L);
      dy-=L*floor((dy+0.5*L)/L);
      dr2=dx*dx+dy*dy;
      if(dr2<cut*cut){
        aij=0.5*(a[i]+a[list[i][j]]);
        w2=aij*aij/dr2;
```

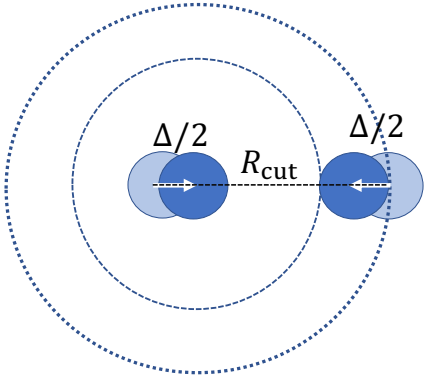
2.1. Verlet 帳簿 (list) 法 (6)

```
23            w6=w2*w2*w2;
24            w12=w6*w6;
25            dUr=-12.*w12/dr2;
26            f[i][0]-=dUr*dx;
27            f[list[i][j]][0]+=dUr*dx;
28            f[i][1]-=dUr*dy;
29            f[list[i][j]][1]+=dUr*dy;
30            *U+=w12-Ucut;
31          }
32        }
33    }
```

2.2 帳簿 (list) 自動更新法

- 近接粒子の帳簿は skin 長を導入することにより，当初帳簿を作成した粒子配置から多少動いても同じ帳簿を使い続けることができる．
- 帳簿の更新のタイミングは，１つでも **skin 長の半分の距離** を動いた粒子が現れた場合である．
- その理由は，ある２粒子が反平行に $skin/2$ の距離動くと，（かなりレアであるが）ポテンシャルカットオフ領域内に領域外から侵入する可能性があり，以前の帳簿が不正確となりうるためである (図 **2** 参照)．
- 従って，粒子の最大変位が $skin/2$ 以内の間は帳簿を使い回し，これを超えた時に，帳簿の更新を自動的に行うアルゴリズムを以下に示す．

2.2 帳簿 (list) 自動更新法 (2)



帳簿の前更新時から粒子が最大 $\Delta/2$動くと外部からカットオフ
領域内への粒子の侵入の可能性が生じる（レアではあるが）

2.2 帳簿 (list) 自動更新法 (3)

図 2: ある 2 粒子が反並行に $\Delta/2(= skin/2)$ の距離動くと，ポテンシャルカットオフ領域内に領域外から侵入する可能性が出てくる．つまり，以前の帳簿が不正確となる．

リスト 4: 帳簿の自動更新のアルゴリズム

```
1  #define Np 1024
2  #define Nn 100
3  #define L 40.0
4  #define dim 2
5  #define cut 3.0
6  #define skin 1.0
7
8  void update(double (*x_update)[dim],double (*x)[dim])
9  {
10   for(int i=0;i<Np;i++)
11     for(int j=0;j<dim;j++)
12       x_update[i][j]=x[i][j];
13 }
14
15 void calc_disp_max(double *disp_max,double (*x)[dim],double (*x_update)[dim])
16 {
17   double dx,dy;
18   double disp;
19   for(int i=0;i<Np;i++){
20     dx=x[i][0]-x_update[i][0];
```

2.2 帳簿 (list) 自動更新法 (4)

```
21      dy=x[i][1]-x_update[i][1];
22      dx-=L*floor((dx+0.5*L)/L);
23      dy-=L*floor((dy+0.5*L)/L);
24      disp = dx*dx+dy*dy;
25      if(disp > *disp_max)
26        *disp_max =disp;
27    }
28  }
29
30  void auto_list_update(double *disp_max,double (*x)[dim],double (*x_update)[dim],int (*list)[Nn]){
31    static int count=0;
32    count++;
33    calc_disp_max(&(*disp_max),x,x_update);
34    if(*disp_max > skin*skin*0.25){
35      list_verlet(list,x);
36      update(x_update,x);
37      //     std::cout<<"update"<<*disp_max<<" "<<count<<std::endl;
38      *disp_max=0.0;
39      count=0;
40    }
41  }
```

- 第 7 回の分子動力学法（md.cpp）をベースにして，帳簿法（自動更新）を施したコード (md_list.cpp) を，GitHub リポジトリより取得可能[リンク].

2.2 帳簿 (list) 自動更新法 (5)

- 第 8 回の分子動力学法（mc.cpp）をベースにして，帳簿法（自動更新）を施したコード (mc_list.cpp) を，GitHub リポジトリより取得可能[リンク].

補足　さらなる高速化として，Verlet 帳簿法をさらに発展させた 領域分割法 **(cell list 法)** と呼ばれる手法がある．システムサイズにも拠るが Verlet 帳簿法の 1.5 倍ほどの速さがでる．とはいえ，Verlet 帳簿法でも十分（従来手法の 10 倍以上）高速化できたので，ここまで出来れば十分．

目次

参考文献・ウェブサイト