Data Science Course Group: Simulation Practice
Lecture 5 Lecture Materials

Instructor: Takeshi Kawasaki

Nagoya University, Graduate School of Science, Department of Physics, Non-Equilibrium Physics Laboratory (R Lab)

Last update: August 22, 2024

Contents

Contents

1. Lecture Schedule

- This practical session will be conducted during the first semester of spring.
- Lecture materials will be uploaded by 11:00 AM on the scheduled lecture date.
- Schedule:
  1. 4/15: First Lecture
  2. 4/22: Second Lecture
  3. 4/30 (Tuesday): Third Lecture
  4. 5/13: Fourth Lecture (Midterm report assignment released)
  5. **5/20: Fifth Lecture**
  6. 5/27: Sixth Lecture (**Midterm report submission deadline**)
  7. **6/03: No lecture**
  8. 6/10: Seventh Lecture (**Final report assignment released**)
  9. 6/17: Eighth Lecture
  10. 6/24: Ninth Lecture (Make-up class)

1.1. Syllabus

The following topics are planned to be covered in this practical session (subject to change based on progress).

1. Introduction
   - Using C(C++) (primarily for numerical calculations)
   - Using Python (for data analysis and plotting)
   - Principles of numerical computation
   - Round-off errors
   - Non-dimensionalization in scientific computing
2. Numerical Solutions of Ordinary Differential Equations: Examples of Damped Oscillations and Harmonic Oscillators
   - Numerical integration of differential equations
   - Stability of orbits and conservation laws
3. Brownian Motion of Single Particles
   - Langevin Equation (Stochastic Differential Equation)
   - Generating normal random numbers
   - Euler-Maruyama method
   - Time average and ensemble average
4. Brownian Motion of Multi-Particle Systems
   - Calculation of interaction forces
   - Simulation of non-equilibrium systems: Example of phase separation phenomena
5. Molecular Dynamics Simulation of Multi-Particle Systems
   - Position Verlet method and velocity Verlet method
   - Conservation laws in multi-particle systems
6. Monte Carlo Method
   - Review of statistical mechanics
   - Markov Chain Monte Carlo method
   - Metropolis criterion

Contents

2.1. Generating Normal Random Numbers: Box-Muller Method [1]

In this section, we introduce methods for numerically generating normal random numbers, which were not fully covered in the fourth lecture.

- The Box-Muller method [1] is a well-known technique for generating normal random numbers $\mathbf{R}_G$ with a variance of 1.

Box-Muller Method [1]

Let U_1 and U_2 be uniformly distributed random numbers in the range $[0, 1]$. Then, the following X_1 and X_2 are independent standard (variance 1) normal random numbers.

$$X_1 = \sqrt{-2\log U_1}\cos 2\pi U_2 \quad (1)$$

$$X_2 = \sqrt{-2\log U_1}\sin 2\pi U_2 \quad (2)$$

(Proof)

- Solve for U_1 and U_2 by combining equations (1) and (2).

2.1. Generating Normal Random Numbers: Box-Muller Method [1] (2)

- First, by squaring X_1 and X_2, we obtain the following relation:

$$\log U_1 = -\frac{X_1^2 + X_2^2}{2} \tag{3}$$

$$U_1 = e^{-\frac{X_1^2+X_2^2}{2}} \tag{4}$$

- Dividing equation (1) by equation (2), we get:

$$\frac{X_2}{X_1} = \tan 2\pi U_2$$

$$U_2 = \frac{1}{2\pi} \arctan \frac{X_2}{X1} \tag{5}$$

- Using these, it can be shown that X_1 and X_2 are **independent Gaussian processes**.
- Now, introduce the probability density functions $P(X_1, X_2)$ and $\tilde{P}(U_1, U_2)$.
- The relationship between $P(X_1, X_2)$ and $\tilde{P}(U_1, U_2)$ is given by:

$$P(X_1, X_2)\mathrm{d}X_1\mathrm{d}X_2 = \tilde{P}(U_1, U_2)\mathrm{d}U_1\mathrm{d}U_2 = \tilde{P}(U_1, U_2)\left|\frac{\partial(U_1, U_2)}{\partial(X_1, X_2)}\right|\mathrm{d}X_1\mathrm{d}X_2 \tag{6}$$

2.1. Generating Normal Random Numbers: Box-Muller Method [1] (3)

- Since U_1 and U_2 are independent uniform random numbers in the range $[0, 1]$,

$$\int_0^1 \mathrm{d}U_1 \int_0^1 \mathrm{d}U_2 \tilde{P}(U_1, U_2) = 1 \tag{7}$$

  we obtain $\tilde{P}(U_1, U_2) = 1$. Therefore,

$$P(X_1, X_2) = \left| \frac{\partial(U_1, U_2)}{\partial(X_1, X_2)} \right| \tag{8}$$

- Now, the Jacobian $|\frac{\partial(U_1,U_2)}{\partial(X_1,X_2)}|$ is

$$\left| \frac{\partial(U_1, U_2)}{\partial(X_1, X_2)} \right| = \begin{vmatrix} \frac{\partial U_1}{\partial X_1} & \frac{\partial U_2}{\partial X_1} \\ \frac{\partial U_1}{\partial X_2} & \frac{\partial U_2}{\partial X_2} \end{vmatrix} = \left| \frac{\partial U_1}{\partial X_1} \frac{\partial U_2}{\partial X_2} - \frac{\partial U_2}{\partial X_1} \frac{\partial U_1}{\partial X_2} \right| \tag{9}$$

2.1. Generating Normal Random Numbers: Box-Muller Method [1] (4)

- Therefore, after performing the appropriate differentiation:

$$\left|\frac{\partial(U_1,U_2)}{\partial(X_1,X_2)}\right| = \left|\frac{1}{2\pi}\frac{e^{-\frac{X_1^2+X_2^2}{2}}}{1+(\frac{X_2}{X_1})^2} - \frac{1}{2\pi}\frac{-X_2^2}{X_1^2}\frac{e^{-\frac{X_1^2+X_2^2}{2}}}{1+(\frac{X_2}{X_1})^2}\right| = \frac{1}{2\pi}e^{-\frac{X_1^2+X_2^2}{2}} = \frac{1}{\sqrt{2\pi}}e^{-\frac{X_1^2}{2}}\frac{1}{\sqrt{2\pi}}e^{-\frac{X_2^2}{2}} \tag{10}$$

is obtained. Hence,

$$P(X_1,X_2) = \frac{1}{\sqrt{2\pi}}e^{-\frac{X_1^2}{2}}\frac{1}{\sqrt{2\pi}}e^{-\frac{X_2^2}{2}} \tag{11}$$

Thus, $P(X_1,X_2) = p(X_1)p(X_2)$, showing that X_1 and X_2 are independent Gaussian processes.

$y = \arctan x$ Differentiation

2.1. Generating Normal Random Numbers: Box-Muller Method [1] (5)

$y = \arctan x$ gives, $x = \tan y$. Differentiating this with respect to y,

$$\frac{\mathrm{d}x}{\mathrm{d}y} = \frac{1}{\cos^2 y} \tag{12}$$

This implies,

$$\frac{\mathrm{d}y}{\mathrm{d}x} = \cos^2 y = \frac{1}{1 + \tan^2 y} = \frac{1}{1 + x^2} \tag{13}$$

2.2. Acceleration of the Box-Muller Method (Marsaglia Polar Method)

2.2 Objective

Next, we explain the Marsaglia polar method.

Improved Box-Muller Method (Marsaglia Polar Method) [2]

1. By using uniform random numbers u_1, u_2 within the range $[-1, 1]$, introduce the 2D vector

$$\mathbf{R} = (u_1, u_2) \tag{14}$$

   Here, we randomly generate $\mathbf{R}$ and only extract pairs that fall within a unit circle (discard those outside).

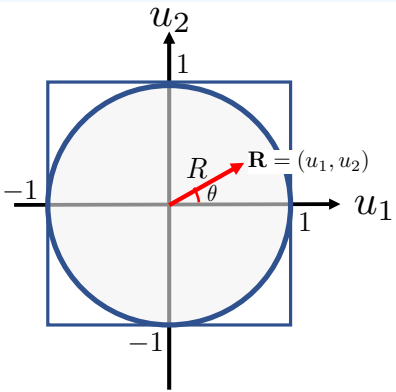
2. In this way, $R^2 = u_1^2 + u_2^2$ becomes a uniform random number in the range $[0, 1]$ (proof provided below). Therefore, set $U_1 = R^2$, and obtain $\sqrt{-2 \log U_1}$.

3. Since the argument of $\mathbf{R}$ is uniformly distributed within the range $[0, 2\pi]$, it is equivalent to $2\pi U_2$. Thus, trigonometric functions with $2\pi U_2$ as arguments can be obtained indirectly as:

$$\frac{u_1}{\square} = \cos 2\pi U_2 \tag{15}$$

- In the Box-Muller method, **the calculation of trigonometric functions is relatively expensive**, and it becomes a significant cost when generating a large number of random numbers.
- Here, we introduce a method that achieves the same results as the Box-Muller method without directly calculating trigonometric functions (Marsaglia polar method) [2].

2.2. Acceleration of the Box-Muller Method (Marsaglia Polar Method) (2)



$$U_1 = R^2$$

$$\cos 2\pi U_2 = \frac{u_1}{R}$$

$$\sin 2\pi U_2 = \frac{u_2}{R}$$

図 1: Uniform random number generation in the Marsaglia polar method.

2.2. Acceleration of the Box-Muller Method (Marsaglia Polar Method) (3)

(2 Proof)
Let $X = R^2$. The relationship between the probability density function $f(X)$ and the probability density function $g(R)$ with respect to R is given by:

$$f(X)\mathrm{d}X = g(R)\mathrm{d}R = g(R)\frac{\partial R}{\partial X}\mathrm{d}X \tag{17}$$

Now, since $g(R)$ is proportional to R^1, $g(R) = CR$ (where C is a constant). Therefore, by performing the following integration:

$$\int_0^1 g(R)\mathrm{d}R = \left[\frac{C}{2}R^2\right]_0^1 = \frac{C}{2} = 1 \tag{18}$$

we obtain $C = 2$. On the other hand, from the relationship $X = R^2$, $\frac{\partial R}{\partial X} = \frac{1}{2R}$. Therefore,

$$f(X)\mathrm{d}X = \frac{2R}{2R}\mathrm{d}X = 1\mathrm{d}X \tag{19}$$

is obtained. Hence, $f(X) = 1$, showing that $X = R^2$ is uniformly distributed over $[0, 1]$.

2.2. Acceleration of the Box-Muller Method (Marsaglia Polar Method) (4)

リスト 1: Normal Random Number Generation Algorithm (Box-Muller Method) The following program ”BM.h” can be obtained from the GitHub repository **[Link]**. It can be used as a subroutine or included as a header.

```
1  double unif_rand(double left, double right)
2  {
3    return left + (right - left)*rand()/RAND_MAX;
4  }
5  double gaussian_rand(void)
6  {
7    static double iset = 0;
8    static double gset;
9    double fac, rsq, v1, v2;
10
11   if (iset == 0) {
12     do {
13       v1 = unif_rand(-1, 1);
14       v2 = unif_rand(-1, 1);
15       rsq = v1*v1 + v2*v2;
16     } while (rsq >= 1.0 || rsq == 0.0);
17     fac = sqrt(-2.0*log(rsq)/rsq);
18
19     gset = v1*fac;
20     iset = 0.50;
21     return v2*fac;
```

2.2. Acceleration of the Box-Muller Method (Marsaglia Polar Method) (5)

```
22    } else {
23      iset = 0;
24      return gset;
25    }
26  }
```

[1] The area of the annulus, when divided into small segments, is proportional to R.

2.3. Various Averages

2.3 Objective

Since numerical results include statistical errors, it is necessary to perform various averaging operations to extract physically important elements during analysis. Two commonly used averaging methods are introduced.

2.3.1. Time Averaging

Time Averaging

Time averaging refers to the operation of averaging the values at each time point for physical quantities in a steady state.

- This operation is performed on a physical quantity $X(t)$ in a steady state, and is actually computed as:

$$\langle X \rangle_{t_0} = \lim_{T \to \infty} \frac{1}{T} \int_0^T \mathrm{d}t_0 X(t_0) \tag{20}$$

- Similarly, when this operation is applied to a two-time correlation function $C(t, t_0) = X(t + t_0)X(t_0)$, it becomes:

$$C(t) = \lim_{T \to \infty} \frac{1}{T} \int_0^T \mathrm{d}t_0 C(t, t_0) \tag{21}$$

  At this time, $C(t) = \langle X(t + t_0)X(t_0) \rangle_{t_0}$, or more simply $C(t) = \langle X(t)X(0) \rangle$.

2.3.1. Time Averaging (2)

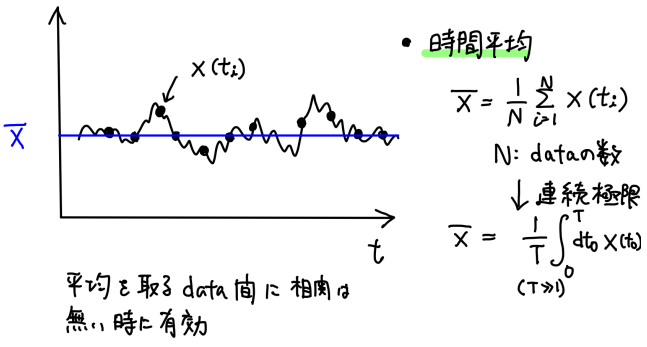


図 2: Concept of time averaging.

2.3.2. Ensemble Averaging

Ensemble Averaging

Ensemble averaging refers to the process of averaging the results of many independent and identical trials (experiments).

- Let A_i be the physical quantity of the αth sample in a thermal equilibrium state. Then the ensemble average is given by:

$$\langle A \rangle_{\text{ens}} = \frac{1}{N_{\text{ens}}} \sum_{\alpha=1}^{N_{\text{ens}}} A_\alpha \tag{22}$$

- This is equivalent to the following statistical mechanical average:

$$\langle A \rangle_{\text{ens}} = \frac{\text{Tr} A(\mathbf{q}, \mathbf{p}) e^{-\beta \hat{H}(\mathbf{q},\mathbf{p})}}{Z} \tag{23}$$

where Z is the partition function: $Z = \text{Tr} e^{-\beta \hat{H}(\mathbf{q},\mathbf{p})}$.

Note In a thermal equilibrium state, **time averaging and ensemble averaging become equal.**

→ **Ergodic Hypothesis**

2.3.2. Ensemble Averaging (2)

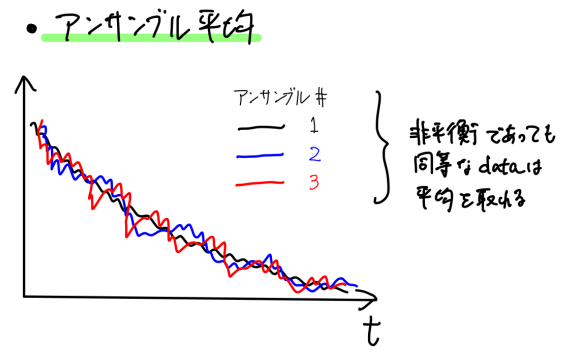


図 3: Concept of ensemble averaging.

Contents

3. Explanation of the Fourth Assignment

Fourth Assignment Implementation of Single-Particle Brownian Motion

Consider the motion of a single particle driven by thermal fluctuations in a three-dimensional solvent with temperature T and friction coefficient ζ. This particle's motion is widely known to be modeled by the Langevin equation $m\dot{\mathbf{v}}(t) = -\zeta\mathbf{v}(t) + \mathbf{F}_{\mathrm{B}}(t)$. When this Langevin equation is non-dimensionalized using length a and time unit $\frac{m}{\zeta}$, the parameter $T^* = \frac{mk_{\mathrm{B}}T}{a^2\zeta^2}$ becomes the main parameter. Answer the following questions about the motion of this particle. Note that **the notation $\langle\cdots\rangle$ below refers to quantities averaged over time or ensemble.**

(1) Non-dimensionalize the analytical solution for the mean square displacement $\left\langle \Delta\mathbf{r}(t)^2 \right\rangle = \frac{2dk_{\mathrm{B}}T}{\zeta}\left\{t + \frac{m}{\zeta}e^{-\zeta t/m} - \frac{m}{\zeta}\right\}$ (see the fourth lecture notes) and express it using the parameter T^*.

(2) Confirm that the theoretical solution and numerical solution match for any T^*. The numerical solution should be obtained using the semi-implicit Euler-Maruyama method.

(3) Compute the velocity autocorrelation function $C(t) = \langle \mathbf{v}(t)\cdot\mathbf{v}(0)\rangle$. Also, compare the numerical result with the theoretical solution $C(t) = \langle \mathbf{v}(t)\cdot\mathbf{v}(0)\rangle = \frac{dk_{\mathrm{B}}T}{m}e^{-\zeta t/m}$.

3. Explanation of the Fourth Assignment (2)

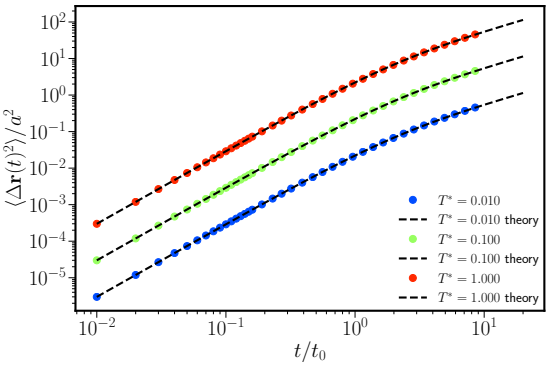


図 4: Numerical results of the mean square displacement. Here, the parameter T^* is varied as $0.01, 0.1, 1.0$. The dashed lines represent the theoretical solution [Equation (26)]. The numerical solution agrees well with the theory.

3. Explanation of the Fourth Assignment (3)

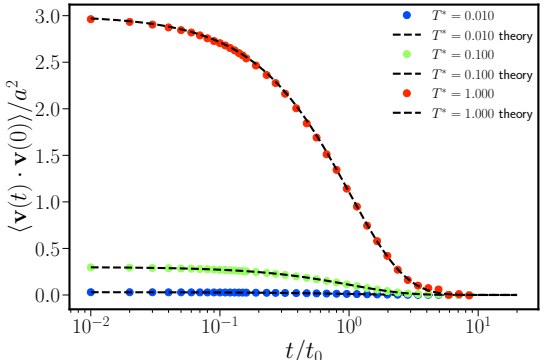


図 5: Numerical results of the velocity autocorrelation function. The parameter T^* is varied as $0.01, 0.1, 1.0$. The dashed lines represent the theoretical solution [Equation (32)]. The numerical solution shows fairly good agreement.

3. Explanation of the Fourth Assignment (4)

Explanation

(1) The analytical solution for the mean square displacement $\left\langle \Delta \mathbf{r}(t)^2 \right\rangle = \frac{2dk_{\mathrm{BT}}}{\zeta}\left\{ t + \frac{m}{\zeta}e^{-\zeta t/m} - \frac{m}{\zeta} \right\}$ is non-dimensionalized. Considering the units of length a and time $t_0 = m/\zeta$, we obtain:

$$\begin{aligned} a^2 \left\langle \Delta \tilde{\mathbf{r}}(\tilde{t})^2 \right\rangle &= \frac{2dk_{\mathrm{BT}}}{\zeta}[t_0\tilde{t} + t_0 e^{-\tilde{t}} - t_0] \\ &= \frac{2dmk_{\mathrm{BT}}}{\zeta^2}[t_0\tilde{t} + t_0 e^{-\tilde{t}} - t_0] \\ &= \frac{2dmk_{\mathrm{BT}}}{\zeta^2}[t_0\tilde{t} + t_0 e^{-\tilde{t}} - t_0] \end{aligned} \tag{24}$$

Thus,

$$\begin{aligned} \left\langle \Delta \tilde{\mathbf{r}}(\tilde{t})^2 \right\rangle &= \frac{2dmk_{\mathrm{BT}}}{\zeta^2 a^2}[t_0\tilde{t} + t_0 e^{-\tilde{t}} - t_0] \\ &= 2dT^*[t_0\tilde{t} + e^{-\tilde{t}} - t_0] \end{aligned} \tag{25}$$

3. Explanation of the Fourth Assignment (5)

Therefore, in three dimensions ($d = 3$):

$$\boxed{\left\langle \Delta\tilde{\mathbf{r}}(\tilde{t})^2 \right\rangle = 6T^*[t_0\tilde{t} + e^{-\tilde{t}} - t_0]} \tag{26}$$

is obtained.

Supplement

In the short-time limit, by expanding $e^{-\tilde{t}} \approx 1 - \tilde{t} + \frac{1}{2}\tilde{t}^2$:

$$\boxed{\left\langle \Delta\tilde{\mathbf{r}}(\tilde{t})^2 \right\rangle \approx 3T^*\tilde{t}^2} \tag{27}$$

a ballistic trajectory is obtained. In the long-time limit, the term $e^{-\tilde{t}} - 1$ drops out, so:

$$\boxed{\left\langle \Delta\tilde{\mathbf{r}}(\tilde{t})^2 \right\rangle \approx 6T^*\tilde{t}} \tag{28}$$

a diffusive trajectory is obtained. Thus, the non-dimensionalized diffusion coefficient is equal to the value of the parameter T^* itself.

3. Explanation of the Fourth Assignment (6)

(2) Examples of numerical computation in C language are provided in Listings 2 and 3. The main computation program that solves the equation of motion is “langevin.cpp,” and the analysis program for the mean square displacement and velocity autocorrelation function, using the data output from “langevin.cpp,” is “analyze.cpp.” Numerical results for the mean square displacement are shown in Fig. **4**. The parameter T^* is varied as $0.01, 0.1, 1.0$. The dashed lines represent the theoretical solution [Equation (26)]. Good agreement with the theory is observed.

(3) As shown in the supplement of the fourth lecture, the velocity autocorrelation function of a Brownian particle for $t \geq 0$ is given by:

$$C(t) = \langle \mathbf{v}(t) \cdot \mathbf{v}(0) \rangle = \frac{dk_B T}{m} e^{-\zeta t/m} \tag{29}$$

When non-dimensionalized, this becomes:

$$\frac{a^2}{t_0^2} \langle \tilde{\mathbf{v}}(\tilde{t}) \cdot \tilde{\mathbf{v}}(0) \rangle = \frac{dk_B T}{m} e^{-\zeta t/m} \tag{30}$$

3. Explanation of the Fourth Assignment (7)

Thus,

$$\langle \tilde{\mathbf{v}}(\tilde{t}) \cdot \tilde{\mathbf{v}}(0) \rangle = \frac{dmk_BT}{\zeta^2 a^2} e^{-\zeta t/m} \tag{31}$$

$$= 3T^* e^{-\tilde{t}} \tag{32}$$

where $d = 3$ is assumed. A comparison between this theoretical solution and numerical solution is shown in Fig. **5**. The numerical solution shows fairly good agreement with the theory, though not as precise as for the mean square displacement.

リスト 2: “langevin.cpp” Available from the following GitHub repository **[Link]**

```
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 tmax 10
10 #define dt 0.01
```

3. Explanation of the Fourth Assignment (8)

```
11 #define temp 0.01 //parameter
12 #define ensemble 1000
13 #define dim 3
14 //using namespace std;
15
16 void ini_phase(double *x,double *v){
17   int i;
18   for(i=0;i<dim;i++){
19     x[i]=0.;
20     v[i]=0.;
21   }
22 }
23
24 void ini_clock(int *j,double *tout){
25   *j=0;
26   *tout=1.e-2;
27 }
28
29 void eom(double *v,double *x){
30   int i;
31   for(i=0;i<dim;i++){
32     v[i]+=-v[i]*dt+sqrt(2.*temp*dt)*gaussian_rand();
33     x[i]+=v[i]*dt;
34   }
```

3. Explanation of the Fourth Assignment (9)

```
35 }
36
37 void output(double *x,double *v,int j){
38   char filename[128];
39   std::ofstream file;
40
41   sprintf(filename,"coord_dt%.3fT%.3f.dat",dt,temp);
42   file.open(filename,std::ios::app); //append
43   file <<j*dt<<"\t"<<x[0]<<"\t"<<x[1]<<"\t"<<x[2]<<std::endl;
44   //  std::cout<<j*dt<<"\t"<<x[0]<<"\t"<<x[1]<<"\t"<<x[2]<<std::endl;
45   file.close();
46
47   sprintf(filename,"vel_dt%.3fT%.3f.dat",dt,temp);
48   file.open(filename,std::ios::app); //append
49   file <<j*dt<<"\t"<<v[0]<<"\t"<<v[1]<<"\t"<<v[2]<<std::endl;
50   file.close();
51
52 }
53
54 int main(){
55   double x[dim],v[dim],t,tout;
56   int i,j;
57   ini_phase(x,v);
58   for(i=0;i<ensemble;i++){
```

3. Explanation of the Fourth Assignment (10)

```
59      ini_clock(&j,&tout);
60      output(x,v,j);
61      while(j*dt < tmax){
62        j++;
63        eom(v,x);
64        if(j*dt >= tout){
65        output(x,v,j);
66        tout*=1.2;
67        }
68      }
69    }
70    return 0;
71  }
```

3. Explanation of the Fourth Assignment (11)

リスト 3: “analyze.cpp” Available from the following GitHub repository **[Link]**

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include <iostream>
5  #include <fstream>
6  #include <cfloat>
7
8  #define temp 0.01
9  #define dt 0.01
10 #define ensemble 1000
11 #define window 39
12 #define dim 3
13 //using namespace std;
14
15 void ini(double *dr2,double *corr){
16   for(int i=0;i<window;i++){
17     dr2[i]=0.0;
18     corr[i]=0.0;
19   }
20 }
21
22 void input(double (*x)[dim],double (*v)[dim],double *t){
```

3. Explanation of the Fourth Assignment (12)

```
23    char filename[128];
24    std::ifstream file;
25    sprintf(filename,"coord_dt%.3fT%.3f.dat",dt,temp);
26    file.open(filename);
27    int asize=ensemble*window;
28    for(int i=0;i<asize;i++){
29      file >> t[i] >> x[i][0] >> x[i][1] >> x[i][2];
30    }
31    file.close();
32
33    sprintf(filename,"vel_dt%.3fT%.3f.dat",dt,temp);
34    file.open(filename);
35    for(int i=0;i<asize;i++){
36      file >> t[i] >> v[i][0] >> v[i][1] >> v[i][2];
37      // std::cout << t[i] <<"\t"<<v[0][i]<<"\t"<<v[1][i]<<"\t"<<v[2][i]<<std::endl;
38    }
39    file.close();
40  }
41
42  void output(double *t,double *dr2,double *corr){
43    char filename[128];
44    std::ofstream file;
45    sprintf(filename,"msd_dt%.3fT%.3f.dat",dt,temp);
46    file.open(filename);
```

3. Explanation of the Fourth Assignment (13)

```
47    for(int i=1;i<window;i++)
48      file<<t[i]-t[0]<<"\t"<<dr2[i]<<std::endl;
49    file.close();
50
51    sprintf(filename,"corr_dt%.3fT%.3f.dat",dt,temp);
52    file.open(filename);
53    for(int i=1;i<window;i++)
54      file<<t[i]-t[0]<<"\t"<<corr[i]<<std::endl;
55    file.close();
56  }
57
58  void analyze(double (*x)[dim],double (*v)[dim],double *t,double *dr2,double *corr){
59    double dx[dim],corr_x[dim];
60    for(int i=0;i<ensemble;i++)
61      for(int j=0;j<window;j++){
62        for(int k=0;k<dim;k++){
63        dx[k]=(x[j+window*i][k]-x[window*i][k]);
64        corr_x[k]=v[j+window*i][k]*v[window*i][k];
65        dr2[j]+=(dx[k]*dx[k])/ensemble;
66        corr[j]+=(corr_x[k])/ensemble;
67        }
68      }
69  }
70
```

3. Explanation of the Fourth Assignment (14)

```
71 int main(){
72   double t[ensemble*window],dr2[window],corr[window];
73   int i,j;
74   double  (*x)[dim] = new double[ensemble*window][dim];
75   double  (*v)[dim] = new double[ensemble*window][dim];
76
77   ini(dr2,corr);
78   input(x,v,t);
79   analyze(x,v,t,dr2,corr);
80   output(t,dr2,corr);
81   delete[] x;
82   delete[] v;
83   return 0;
84 }
```

Contents

4. Fifth Assignment

Fifth Assignment Preparation for Multi-Particle Calculations

In a two-dimensional plane, distribute 512 discs with a diameter of 1 in a square space with a side length of $L = 40$.

(1) Arrange the particles in a square lattice and visualize the result.

(2) Arrange the particles in a hexagonal lattice and visualize the result.

(3) (Advanced) Consider an algorithm for measuring the distance between particle i and other particles j when the square boundary is periodic (necessary for force calculations, etc.).

(4) (Advanced) Consider an algorithm for storing the ”particle numbers” within a distance (e.g., 5) calculated in (3) into an array (Verlet list).

Contents

5. Appendix 1: Comparison of Stack Memory and Dynamic (Heap) Memory Allocation (C Language)

リスト 4: Variable and Array Allocation Using Stack Memory (Example in C). Stack memory (in the stack region) is generally limited to around 10 MB on a laptop PC (quite small) [3].

```
1 double x,y[10000],z[10000][10];
```

リスト 5: Dynamic Memory Allocation (Example in C++). The method for releasing memory differs between variables and arrays. Memory (in the heap region) can generally be allocated up to the GB order on a laptop PC [3]. The same can be done in C language using the malloc function.

```
1 double *x = new double;
2 double *y = new double[10000];
3 double (*z)[10] = new double[10000][10];
4 //  Free memory when no longer needed
5 delete x;
6 delete [] y;
7 delete [] z;
```

Contents

References and Websites

[1] Box GEP, Muller ME (1958) A Note on the Generation of Random Normal Deviates. The Annals of Mathematical Statistics 29(2):610–611.

[2] Marsaglia G, Bray TA (1964) A Convenient Method for Generating Normal Variables. SIAM Review 6(3):260–264.

[3] Lemniscater N (year?) C++のスタックメモリと動的メモリの上限値調査 (https://qiita.com/LemniscaterN/items/a3abfa143612cb928bde).