Simulation Tutorials
Lecture 5 Lecture Materials

Instructor: Takeshi Kawasaki

Osaka University, D3 center

Last update: May 15, 2025

Contents

Contents

1. Syllabus

The following topics are planned to be covered in this practical lecture (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. Assignment 4, solution

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 $C(t) = \langle\mathbf{v}(t)\cdot\mathbf{v}(0)\rangle = \frac{dk_{B}T}{m}e^{-\zeta t/m}$.

2. Assignment 4, solution (2)

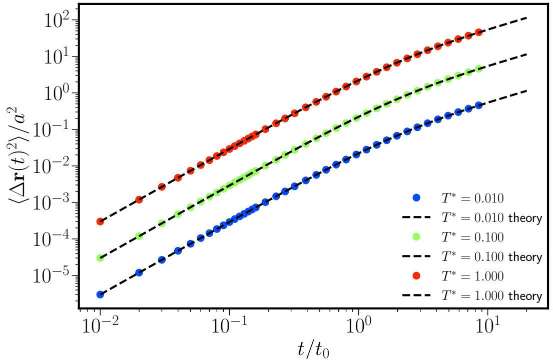

図 **1**: Numerical results of the mean square displacement. Here, the parameter T^* is varied as $0.01, 0.1, 1.0$. The dashed lines represent the theoretical solution [Equation (3)]. The numerical solution agrees well with the theory.

2. Assignment 4, solution (3)

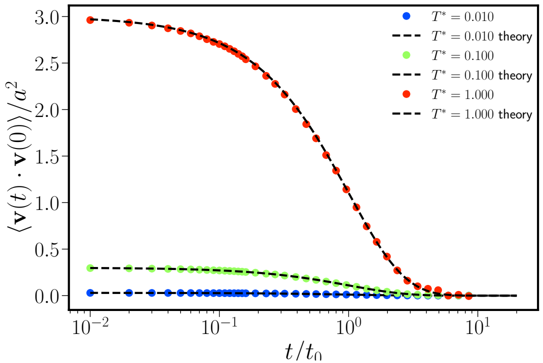

図 **2**: Numerical results of the velocity autocorrelation function. The parameter T^* is varied as $0.01, 0.1, 1.0$. The dashed lines represent the theoretical solution [Equation (9)]. The numerical solution shows fairly good agreement.

2. Assignment 4, solution (4)

Explanation

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

$$\begin{aligned} a^2\left\langle \Delta\tilde{\mathbf{r}}(\tilde{t})^2 \right\rangle &= \frac{2dk_{\mathrm{B}T}}{\zeta}[t_0\tilde{t} + t_0e^{-\tilde{t}} - t_0] \\ &= \frac{2dmk_{\mathrm{B}T}}{\zeta^2}[t_0\tilde{t} + t_0e^{-\tilde{t}} - t_0] \\ &= \frac{2dmk_{\mathrm{B}T}}{\zeta^2}[t_0\tilde{t} + t_0e^{-\tilde{t}} - t_0] \qquad (1) \end{aligned}$$

Thus,

$$\begin{aligned} \left\langle \Delta\tilde{\mathbf{r}}(\tilde{t})^2 \right\rangle &= \frac{2dmk_{\mathrm{B}T}}{\zeta^2 a^2}[t_0\tilde{t} + t_0e^{-\tilde{t}} - t_0] \\ &= 2dT^*[t_0\tilde{t} + e^{-\tilde{t}} - t_0] \qquad (2) \end{aligned}$$

2. Assignment 4, solution (5)

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

$$\boxed{\left\langle \Delta\tilde{\mathbf{r}}(\tilde{t})^2 \right\rangle = 6T^*[t_0\tilde{t} + e^{-\tilde{t}} - t_0]} \tag{3}$$

is obtained.

Supplement

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

$$\boxed{\left\langle \Delta\tilde{\mathbf{r}}(\tilde{t})^2 \right\rangle \approx 3T^*\tilde{t}^2} \tag{4}$$

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

$$\boxed{\left\langle \Delta\tilde{\mathbf{r}}(\tilde{t})^2 \right\rangle \approx 6T^*\tilde{t}} \tag{5}$$

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

2. Assignment 4, solution (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. **1**. The parameter T^* is varied as $0.01, 0.1, 1.0$. The dashed lines represent the theoretical solution [Equation (3)]. 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:

$$C(t) = \langle \mathbf{v}(t) \cdot \mathbf{v}(0) \rangle = \frac{dk_B T}{m} e^{-\zeta t/m} \tag{6}$$

When non-dimensionalized, this becomes:

$$\frac{a^2}{t_0^2} \langle \tilde{\mathbf{v}}(\tilde{t}) \cdot \tilde{\mathbf{v}}(0) \rangle = \frac{dk_B T}{m} e^{-\zeta t/m} \tag{7}$$

2. Assignment 4, solution (7)

Thus,

$$\langle \tilde{\mathbf{v}}(\tilde{t}) \cdot \tilde{\mathbf{v}}(0) \rangle = \frac{dmk_BT}{\zeta^2 a^2} e^{-\zeta t/m} \tag{8}$$

$$= 3T^* e^{-\tilde{t}} \tag{9}$$

where $d = 3$ is assumed. A comparison between this theoretical solution and numerical solution is shown in Fig. **2**. The numerical solution shows fairly good agreement with the theory, though not as precise as for the mean square displacement.

リスト 1: “langevin.cpp” Available from the following GitHub repository **[Link]**

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <iostream>
#include <fstream>
#include <cfloat>
#include "BM.h"

#define tmax 10
#define dt 0.01
```

2. Assignment 4, solution (8)

```
#define temp 0.01 //parameter
#define ensemble 1000
#define dim 3
//using namespace std;

void ini_phase(double *x,double *v){
  int i;
  for(i=0;i<dim;i++){
    x[i]=0.;
    v[i]=0.;
  }
}

void ini_clock(int *j,double *tout){
  *j=0;
  *tout=1.e-2;
}

void eom(double *v,double *x){
  int i;
  for(i=0;i<dim;i++){
    v[i]+=-v[i]*dt+sqrt(2.*temp*dt)*gaussian_rand();
    x[i]+=v[i]*dt;
  }
```

2. Assignment 4, solution (9)

```
}

void output(double *x,double *v,int j){
  char filename[128];
  std::ofstream file;

  sprintf(filename,"coord_dt%.3fT%.3f.dat",dt,temp);
  file.open(filename,std::ios::app); //append
  file <<j*dt<<"\t"<<x[0]<<"\t"<<x[1]<<"\t"<<x[2]<<std::endl;
  //  std::cout<<j*dt<<"\t"<<x[0]<<"\t"<<x[1]<<"\t"<<x[2]<<std::endl;
  file.close();

  sprintf(filename,"vel_dt%.3fT%.3f.dat",dt,temp);
  file.open(filename,std::ios::app); //append
  file <<j*dt<<"\t"<<v[0]<<"\t"<<v[1]<<"\t"<<v[2]<<std::endl;
  file.close();

}

int main(){
  double x[dim],v[dim],t,tout;
  int i,j;
  ini_phase(x,v);
  for(i=0;i<ensemble;i++){
```

2. Assignment 4, solution (10)

```
    ini_clock(&j,&tout);
    output(x,v,j);
    while(j*dt < tmax){
      j++;
      eom(v,x);
      if(j*dt >= tout){
      output(x,v,j);
      tout*=1.2;
      }
    }
  }
  return 0;
}
```

2. Assignment 4, solution (11)

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

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <iostream>
#include <fstream>
#include <cfloat>

#define temp 0.01
#define dt 0.01
#define ensemble 1000
#define window 39
#define dim 3
//using namespace std;

void ini(double *dr2,double *corr){
  for(int i=0;i<window;i++){
    dr2[i]=0.0;
    corr[i]=0.0;
  }
}

void input(double (*x)[dim],double (*v)[dim],double *t){
```

2. Assignment 4, solution (12)

```
  char filename[128];
  std::ifstream file;
  sprintf(filename,"coord_dt%.3fT%.3f.dat",dt,temp);
  file.open(filename);
  int asize=ensemble*window;
  for(int i=0;i<asize;i++){
    file >> t[i] >> x[i][0] >> x[i][1] >> x[i][2];
  }
  file.close();

  sprintf(filename,"vel_dt%.3fT%.3f.dat",dt,temp);
  file.open(filename);
  for(int i=0;i<asize;i++){
    file >> t[i] >> v[i][0] >> v[i][1] >> v[i][2];
    // std::cout << t[i] <<"\t"<<v[0][i]<<"\t"<<v[1][i]<<"\t"<<v[2][i]<<std::endl;
  }
  file.close();
}

void output(double *t,double *dr2,double *corr){
  char filename[128];
  std::ofstream file;
  sprintf(filename,"msd_dt%.3fT%.3f.dat",dt,temp);
  file.open(filename);
```

2. Assignment 4, solution (13)

```
  for(int i=1;i<window;i++)
    file<<t[i]-t[0]<<"\t"<<dr2[i]<<std::endl;
  file.close();

  sprintf(filename,"corr_dt%.3fT%.3f.dat",dt,temp);
  file.open(filename);
  for(int i=1;i<window;i++)
    file<<t[i]-t[0]<<"\t"<<corr[i]<<std::endl;
  file.close();
}

void analyze(double (*x)[dim],double (*v)[dim],double *t,double *dr2,double *corr){
  double dx[dim],corr_x[dim];
  for(int i=0;i<ensemble;i++)
    for(int j=0;j<window;j++){
      for(int k=0;k<dim;k++){
      dx[k]=(x[j+window*i][k]-x[window*i][k]);
      corr_x[k]=v[j+window*i][k]*v[window*i][k];
      dr2[j]+=(dx[k]*dx[k])/ensemble;
      corr[j]+=(corr_x[k])/ensemble;
      }
    }
}

```

2. Assignment 4, solution (14)

```
int main(){
  double t[ensemble*window],dr2[window],corr[window];
  int i,j;
  double  (*x)[dim] = new double[ensemble*window][dim];
  double  (*v)[dim] = new double[ensemble*window][dim];

  ini(dr2,corr);
  input(x,v,t);
  analyze(x,v,t,dr2,corr);
  output(t,dr2,corr);
  delete[] x;
  delete[] v;
  return 0;
}
```

Contents

3. 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

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

リスト 3: 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].

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

リスト 4: 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.

```cpp
double *x = new double;
double *y = new double[10000];
double (*z)[10] = new double[10000][10];
//  Free memory when no longer needed
delete x;
delete [] y;
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).