Data Science Curriculum (Doctoral Course): Simulation Tutorials
Lecture 2 Lecture Materials

Instructor: Takeshi Kawasaki

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

Last update: August 23, 2024

Contents

Contents

1. Lecture Schedule

- This practice will be conducted in the first half of the spring semester.
- Lecture materials will be uploaded by 11:00 AM on the scheduled day of each lecture.
- Schedule
  1. 4/15: Lecture 1
  2. **4/22: Lecture 2**
  3. 4/30 (Tue): Lecture 3
  4. 5/13: Lecture 4 (**Midterm Report Assignment Released**)
  5. 5/20: Lecture 5
  6. 5/27: Lecture 6 (**Midterm Report Submission Deadline**)
  7. 5/29: Lecture 7
  8. **6/03: No Class**
  9. 6/10: Lecture 8 (**Final Report Assignment Released**)
  10. 6/17: Lecture 9 (Make-up Lecture)

1.1. Syllabus

The following topics are planned to be covered in this practice (subject to change based on progress):

1. Introduction
   - Usage of C (C++) (mainly for numerical computation)
   - Usage of Python (for data analysis and plotting)
   - Philosophy of numerical computation
   - Numerical error and loss of significance
   - Nondimensionalization in scientific computation
2. Numerical solutions to ordinary differential equations: Examples of damped oscillations and harmonic oscillators
   - Numerical integration of differential equations
   - Stability and conservation laws of orbits
3. Brownian motion of single particles
   - Langevin equation (stochastic differential equations)
   - Generation of normal random numbers
   - Euler-Maruyama method
   - Time averaging and ensemble averaging
4. Brownian motion in multi-particle systems
   - Calculation methods for interaction forces
   - Simulation of nonequilibrium 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. Explanation of Assignment 1

Convergence of π (Monte Carlo Simulation)

(1) Regarding the sample program in List 2, consider the convergence of the calculated value of π as the number of random number generations n is varied. Plot the relationship between n on the horizontal axis and $\pi(n)$ on the vertical axis by taking various values of the numerical calculation of $\pi(n)$ with respect to n. Also, evaluate and plot how the error $\delta(n) = |\pi(n) - \pi|$ changes with respect to n.

(2) (Advanced Problem) In large-scale numerical calculations, the pseudo-random number generator rand() may cause problems due to its short random number cycle. On the other hand, the Mersenne Twister, developed by Makoto Matsumoto, is famous as a high-quality pseudo-random number generator [**Reference Link**][1]. Now, rewrite the sample program in List 2 to use the Mersenne Twister [**Reference Link**][2]. You may use the header file for the Mersenne Twister placed in the specified GitHub repository [**Link**][3].

2. Explanation of Assignment 1 (2)

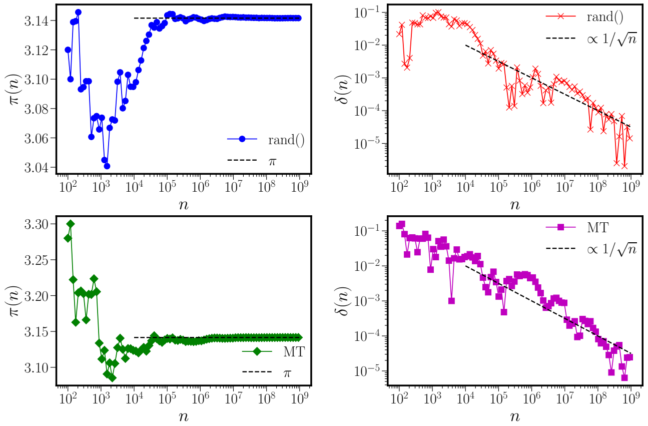

図 1: Calculated $\pi(n)$ and $\delta(n)$ using the standard random number generator rand() (upper) and Mersenne Twister (lower).

- Below is the calculation program using C(C++).

リスト 1: Assignment 1 Sample Calculation Program “pi_error.cpp”. Available in the following GitHub repository: **[Link]**

```

#include <stdio.h> // for printf, etc
#include <stdlib.h> // for rand(), etc
#include <math.h> // for sin(),cos(), etc
#include <iostream>// for std::cout, etc
#include <iomanip>//  for std::setprecision()
#include <fstream> // for ifstream/ofstream
#include <time.h>// for time(NULL), etc
#include "MT.h"// for MT

int main(void){
  int i, count = 0, max = 1e+8;
  double x,y,z,pi, out=1.e+2;
  char fname[128];
  std::ofstream file;
  srand(time(NULL));
  sprintf(fname,"pi-error.dat");
  file.open(fname);
  for(i=0;i<max;i++){
    x = (double)rand()/RAND_MAX;
    y = (double)rand()/RAND_MAX;
    z = x*x + y*y;
    if(z<=1.0)
      count++;
    if(i>=(int)out){     // (int)out: cast double to int
      pi=(double)count /(double)i*4.0; // (double)count: cast int to double
      file<<std::setprecision(16)<<i<<"\t"<<pi<<"\t"<<abs(pi-M_PI)<<std::endl;;
      std::cout<<std::setprecision(16)<<i<<"\t"<<pi<<"\t"<<abs(pi-M_PI)<<std::endl;
      out*=1.2;
    }
  }
  file.close();
  out=1e+2;
  count=0;

  sprintf(fname,"pi-error-MT.dat");
  file.open(fname);
  init_genrand(time(NULL));
  for(i=0;i<max;i++){
    x = genrand_res53();
    y = genrand_res53();
    z = x*x + y*y;
    if(z<=1.0)
      count++;
    if(i>=(int)out){
      pi=(double)count /(double)i*4.0;
      file<<std::setprecision(16)<<i<<"\t"<<pi<<"\t"<<abs(pi-M_PI)<<std::endl;;
      std::cout<<std::setprecision(16)<<i<<"\t"<<pi<<"\t"<<abs(pi-M_PI)<<std::endl;
      out*=1.2;
    }
  }
  file.close();
  return 0;
}
}
```

- Below is a sample program for plotting.

リスト 2: Python Sample Program “pi_error.py” Available in the GitHub repository: **[Link]**. The equivalent program is also included in the jupyter notebook file “jupyter.ipynb”.

```
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
# Increases the resolution of the figures
%config InlineBackend.figure_format = 'retina'
import numpy as np

# Tex font (optional): uncomment the following if you want to use it
#plt.rcParams["text.usetex"] =True
plt.rcParams['font.family'] = 'Arial' # Font name to use
plt.rcParams["font.size"] = 25

# Change the overall size and aspect ratio of the figure
fig = plt.figure(figsize=(18,12))
# Adjust this when placing multiple plots

 ###########################
ax = fig.add_subplot(221)
# Change the file paths as needed
i, pi,error  = np.loadtxt("./Lecture2/pi-error-MT.dat", comments='#', unpack=True)
plt.plot(i, pi, "o-",markersize=10,color="b",label=r"rand()")
plt.xscale('log')
###Drawing a line ######
x= np.linspace(1e4, 1e8, 100)
```

```
y= np.pi+0*x
plt.plot(x, y, "--",markersize=3,linewidth = 2.0, color="k",label=r"$\pi$")
#########
# Format the plot
plt.tick_params(which='major',width = 1, length = 10)
plt.tick_params(which='minor',width = 1, length = 5)
ax.spines['top'].set_linewidth(3)
ax.spines['bottom'].set_linewidth(3)
ax.spines['left'].set_linewidth(3)
ax.spines['right'].set_linewidth(3)
plt.xlabel(r"$n$",color='k', size=30)
plt.ylabel(r"$\pi(n)$",color='k', size=30)

# Adjust the legend position, size, and whether it appears
plt.legend(ncol=1, loc=4, borderaxespad=0, fontsize=25,frameon=False)
# Set the aspect ratio of each plot to 1:1
#################################

 ###########################
ax = fig.add_subplot(222)
# Change the file paths as needed
i, pi,error  = np.loadtxt("./Lecture2/pi-error.dat", comments='#', unpack=True)
plt.plot(i, error, "x-",markersize=10,color="r",label=r"rand()")
plt.xscale('log')
plt.yscale('log')

###Drawing a line ######
x= np.linspace(1e4, 1e8, 100)
```

```
y=1/x**0.5
plt.plot(x, y, "--",markersize=3,linewidth = 2.0, color="k",label=r"$\propto 1/\sqrt{n}$")
#########


# Format the plot
plt.tick_params(which='major',width = 1, length = 10)
plt.tick_params(which='minor',width = 1, length = 5)
ax.spines['top'].set_linewidth(3)
ax.spines['bottom'].set_linewidth(3)
ax.spines['left'].set_linewidth(3)
ax.spines['right'].set_linewidth(3)
plt.xlabel(r"$n$",color='k', size=30)
plt.ylabel(r"$\delta(n)$",color='k', size=30)

# Adjust the legend position, size, and whether it appears
plt.legend(ncol=1, loc=1, borderaxespad=0, fontsize=25,frameon=False)
#################################

 ###########################

ax = fig.add_subplot(223)
# Change the file paths as needed
i, pi,error  = np.loadtxt("./Lecture2/pi-error-MT.dat", comments='#', unpack=True)
plt.plot(i, pi, "D-",markersize=10,color="g",label=r"MT")
plt.xscale('log')
###Drawing a line ######
x= np.linspace(1e4, 1e8, 100)
```

```
y= np.pi+0*x
plt.plot(x, y, "--",markersize=3,linewidth = 2.0, color="k",label=r"$\pi$")
#########

# Format the plot
plt.tick_params(which='major',width = 1, length = 10)
plt.tick_params(which='minor',width = 1, length = 5)
ax.spines['top'].set_linewidth(3)
ax.spines['bottom'].set_linewidth(3)
ax.spines['left'].set_linewidth(3)
ax.spines['right'].set_linewidth(3)
plt.xlabel(r"$n$",color='k', size=30)
plt.ylabel(r"$\pi(n)$",color='k', size=30)

# Adjust the legend position, size, and whether it appears
plt.legend(ncol=1, loc=4, borderaxespad=0, fontsize=25,frameon=False)
################################

###########################

ax = fig.add_subplot(224)
# Change the file paths as needed
i, pi,error  = np.loadtxt("./Lecture2/pi-error-MT.dat", comments='#', unpack=True)
plt.plot(i, error, "s-",markersize=10,color="m",label=r"MT")

###Drawing a line ######
x= np.linspace(1e4, 1e8, 100)
y=1/x**0.5
```

```
plt.plot(x, y, "--",markersize=3,linewidth = 2.0, color="k",label=r"$\propto 1/\sqrt{n}$")
#########
plt.xscale('log')
plt.yscale('log')
# Format the plot
plt.tick_params(which='major',width = 1, length = 10)
plt.tick_params(which='minor',width = 1, length = 5)
ax.spines['top'].set_linewidth(3)
ax.spines['bottom'].set_linewidth(3)
ax.spines['left'].set_linewidth(3)
ax.spines['right'].set_linewidth(3)
plt.xlabel(r"$n$",color='k', size=30)
plt.ylabel(r"$\delta(n)$",color='k', size=30)

# Adjust the legend position, size, and whether it appears
plt.legend(ncol=1, loc=1, borderaxespad=0, fontsize=25,frameon=False)
#################################

# Adjust the margins of the plot
plt.subplots_adjust(wspace=0.3, hspace=0.3)

# Change the file paths as needed.
plt.savefig('./Lecture2/pi-error.png')
plt.savefig('./Lecture2/pi-error.pdf')
```

Contents

3.Basics of Numerical Computation

Below, we delve into the main topic of this Lecture. Here, we summarize the basic points that should be kept in mind when performing numerical computations. Specifically, we will explain:

- **Floating Point Numbers and Rounding Errors (Information Loss and Cancellation)**
- **Discretization**
- **Non-dimensionalization**

3.1. Floating Point Numbers and Rounding Errors (Information Loss and Cancellation)

In this section, we introduce floating point numbers, which often cause problems in numerical computation, leading to rounding errors, information loss, and cancellation. We will also discuss the reasons these issues occur.

Information Loss

When adding a large value and a small value, the information carried by the smaller value is lost. For example, since the precision of a float type is about 6-7 digits, when adding a float value of 77777.7 and 1.23456, the result is also limited to 6-7 digits, resulting in 77778.9. This means that the lower 4 digits of the information carried by 1.23456 are lost.

$$77777.7 + 1.23456 = 77778.9 \tag{1}$$

Cancellation

When subtracting two nearly equal numbers, significant digits are lost, leading to an error.

$$77777.7 - 77777.6 = 0.1 \tag{2}$$

- Next, we consider the nature of floating point numbers, which are the cause of these issues.

3.1. Floating Point Numbers and Rounding Errors (Information Loss and Cancellation) (2)

- Floating point numbers are a way to represent numerical values on a computer, by dividing a number into a mantissa, which represents the digits, and an exponent, which represents the position of the decimal point.
- This is the most widely used representation method for numbers containing decimal points [4].

Floating Point Numbers and Machine Epsilon

Single-precision floating point number (float type): 32-bit, Double-precision floating point number (double type): 64-bit.
Floating point number structure – for binary numbers (base 2):

Sign bit	0 if positive, 1 if negative (1-bit).
Mantissa	(1 digit before the decimal point + fraction part) (float: 23-bit, double: 52-bit).
Exponent	Signed integer (float: 8-bit, double: 11-bit), represented as an offset binary value (to handle negative values, a bias is added to the exponent's origin): float uses $2^7 - 1 = 127$, double uses $2^{10} - 1 = 1023$.

Machine Epsilon (the smallest unit of the mantissa): for double, it is approximately $2^{-52} \approx 2.22 \times 10^{-16}$.
IEEE 754 Floating Point Standard: The mantissa is represented as 1.xxx, with the 1 omitted (known as a hidden bit).

3.2. Discretization

Next, this section explains the method of discretization (differencing) necessary for numerical calculation of derivatives and integrals.

- Here, as an example, we will solve a simple differential equation numerically.
- Suppose a sphere with diameter a [m] and mass m [kg] moves in a one-dimensional fluid with a drag coefficient ζ [kg/s] and an initial velocity v_0 [m/s]. The equation of motion for the sphere is:

$$m\frac{\mathrm{d}v(t)}{\mathrm{d}t} = -\zeta v(t) \tag{3}$$

- To solve this equation numerically, we need to discretize it and perform numerical integration.
- Although increasing the accuracy of numerical integration is crucial depending on the problem, first, let's introduce the Euler method, which is low in accuracy but very simple.
- The exact first derivative of a function $f(t)$ with respect to time is:

$$\frac{\mathrm{d}f(t)}{\mathrm{d}t} = \lim_{\Delta t\to 0}\frac{f(t+\Delta t)-f(t)}{\Delta t} \tag{4}$$

3.2. Discretization (2)

- However, in numerical calculations, it is impossible to take the limit $\Delta t \to 0$ precisely, so a small value is assigned to Δt at each time step, and the calculation is approximated as follows. This method is specifically called the $\boxed{\text{Euler method}}$.

$$\boxed{\frac{\mathrm{d}f(t)}{\mathrm{d}t} \approx \frac{f(t+\Delta t) - f(t)}{\Delta t} + \mathcal{O}((\Delta t))} \tag{5}$$

- The Euler method accumulates errors on the order of Δt compared to the exact derivative, so it may not work for some problems.
- On the other hand, for the problem at hand, which is not a conservative system, a reasonable level of accuracy can be achieved (it doesn't work for harmonic oscillators).
- Using the Euler method, equation 3 can be discretized as:

$$m\frac{v(t+\Delta t) - v(t)}{\Delta t} = -\zeta v(t) \tag{6}$$

resulting in the equation of motion:

$$v(t+\Delta t) = v(t) - \frac{\zeta v(t)\Delta t}{m} \tag{7}$$

3.2. Discretization (3)

- For dissipative differential equations like the one we're handling, the Euler method can provide sufficient accuracy. However, for systems that do not involve dissipation (like harmonic oscillators), where conservation laws such as energy and momentum need to be strictly preserved, higher-order numerical integration methods like the Velocity Verlet method or the Runge-Kutta method are required.

3.3. Non-dimensionalization

Here, we will discuss the non-dimensionalization of physical variables (constants).

- **The numbers that a computer can handle are fundamentally dimensionless.**
- Therefore, to compute an equation with physical dimensions on a computer, it is necessary to separate the physical units and dimensionless numbers.
- To solve the differential equation discussed in the previous section on a computer, we will non-dimensionalize it.
- Let's first assume the time transformation $t \to t_0\tilde{t}$ and the velocity $v = v_0\tilde{v} = \frac{a}{t_0}\tilde{v}$.
- Then, equation (7) becomes:

$$\begin{aligned} \frac{a}{t_0}\tilde{v}(\tilde{t} + \tilde{\Delta t}) &= \frac{a}{t_0}\tilde{v}(\tilde{t}) - \frac{\zeta a \Delta\tilde{t}}{m}\tilde{v}(\tilde{t}) \qquad (8) \\ \tilde{v}(\tilde{t} + \tilde{\Delta t}) &= \tilde{v}(\tilde{t}) - \frac{\zeta t_0}{m}\tilde{v}(\tilde{t})\Delta\tilde{t} \qquad (9) \end{aligned}$$

- Here, $\frac{\zeta t_0}{m}$ is a dimensionless parameter, and in principle, any value can be assigned to it, but for simplicity, we set it to 1. To achieve this, we select the time unit as:

$$\boxed{t_0 = \frac{m}{\zeta}} \qquad (10)$$

This time scale corresponds to the relaxation time of velocity damping!

3.3. Non-dimensionalization (2)

- The choice of t_0 is arbitrary, but if not chosen carefully, the value of the parameter $\frac{\zeta t_0}{m}$ may become too large or too small, leading to **cancellation errors**, so be cautious.
- Now, with $\frac{\zeta t_0}{m} = 1$, the resulting difference equation becomes:

$$\boxed{\tilde{v}(\tilde{t} + \Delta\tilde{t}) = \tilde{v}(\tilde{t}) - \tilde{v}(\tilde{t})\Delta\tilde{t}} \tag{11}$$

  which is extremely simple.
- Solving this numerically involves solving the recurrence relation for $\tilde{v}(\tilde{t})$.

Contents

4. Assignment 2

Numerical Computation and Analytical Comparison of Damped Motion

Consider the motion of a particle in a solvent in one dimension. The equation of motion for this particle is:

$$m\frac{\mathrm{d}v(t)}{\mathrm{d}t} = -\zeta v(t),$$

and at time $t = 0$, it is assumed to be moving with a speed of $10a\zeta/m$. Answer the following questions based on this setup. When non-dimensionalizing, take the unit of length as a and the unit of time as $t_0 = m/\zeta$.

(1) Find the analytical solution of this differential equation. Further, non-dimensionalize it using the units specified in the problem statement.

(2) Numerically compute the time evolution of the particle's velocity over the time interval $[0, 10^4 t_0]$ and compare it with the analytical solution. For simplicity, use the Euler method for discretizing the equation of motion and compare results for different time resolutions: $\Delta t/t_0 = 0.1, 0.01, 0.001$.

Contents

5.1. Common Commands Used on Unix-based Terminals

Here are some commonly used commands on Unix-based terminals, as requested.

- Common Unix Commands [5]

リスト 3: Common Unix Commands:

```
pwd    Display the full path of the current directory
cd     Change directory
cd /   Move to the home directory
mkdir  Create a new directory
rm -r  Delete a specified directory along with its contents
cp -r (dirA) (dirB)   Copy a directory
mv (file) (dir)   Move a file to a directory
ls     Display the list of files in the current directory
cat (file)   Display the contents of a file
rm (file)   Delete a file
cp (fileA) (fileB)   Copy a file
mv (fileA) (fileB)   Rename a file
touch (file)   Create an empty file
find (dir)   List files under a specified directory
more (file)   Display the contents of a file (pauses after each page)
```

Contents

References

[1] Makoto Matsumoto.
Mersenne Twister: A random number generator (since 1997/10).
http://www.math.sci.hiroshima-u.ac.jp/m-mat/MT/mt.html.

[2] Takahiro Ohmi.
C 言語による乱数生成.
https://omitakahiro.github.io/random/random variables generation.html.

[3] Takeshi Kawasaki.
2024-simulation-tutorial (GitHub), April 2022.

[4] 浮動小数点数とは - IT 用語辞典.
https://e-words.jp/w/%E6%B5%AE%E5%8B%95%E5%B0%8F%E6%95%B0%E7%82%B9%E6%95%B0.html.

[5] Takayuki Omori.
UNIX コマンド集.
https://g-omr.github.io/unix.html, 2021.