Worksheet 4: Thermostats

Michael Kuron David Sean

December 11, 2017
Institute for Computational Physics, University of Stuttgart

Contents

1 General Remarks

- Deadline for the report is **Monday, January 8, 2018**
- In this worksheet, you can achieve a maximum of 20 points.
- The report should be written as though it would be read by a fellow student who attends the lecture, but doesn't do the tutorials.
- To hand in your report, send it to your tutor via email.
  - David (`david.sean@icp.uni-stuttgart.de`)
  - Michael (`mkuron@icp.uni-stuttgart.de`)

- Please attach the report to the email. For the report itself, please use the PDF format (we will *not* accept MS Word doc/docx files!). Include graphs and images into the report.
- If the task is to write a program, please attach the source code of the program, so that we can test it ourselves.
- The report should be 5–10 pages long. We recommend using LaTeX. A good template for a report is available online.
- The worksheets are to be solved in groups of two or three people.

2 Introduction

All files required for this tutorial can be found in the archive `templates.tar.gz` which can be downloaded from the lecture's homepage.

3 Random numbers

Computers are deterministic machines. There is no magic algorithm to generate true random numbers. In many situations, for example in computer games or simulations, it would be nice to have a way to obtain random numbers. One way around this problem is to use psuedo-random numbers. Although they are not truly random (as the name implies), they can be algorithmitically generated and "seem" sufficiently random for some of these applications. In this section, you will write your own random number generator using the linear congruential generator method (LCG).

In short, this method looks at the remainder of a division between a number and another large number. It uses the remainder both as a pseudo-random number (the outputed result) and as way to modify one of original numbers so that the output will be different if the function is called a second time. If the process is iterated too many times, the numbers will eventually begin to repeat themselves. Since this is detrimental to almost all applications, the parameters from a pseudo-random number generator are chosen such that the period is large. It is important that the generated pseudo-random numbers be as uncorrelated as possible.

The LCG method employs three parameters called the *modulus* m, the *multiplier* a and the *increment* c. Starting from a number X_n, a new number can be obtained following

$$X_{n+1} = (aX_n + c) \mod m \tag{1}$$

where all numbers considered are integers. The value of the generated X_{n+1} can go as high as $m - 1$. In order to produce a floating point random number in $[0, 1)$ one can simply perform a floating point division X_{n+1}/m. Note that for the iteration to be started, one must supply the initial X_0, often called the *seed*.

Task (3 points)

- implement the linear congruential generator in python using the parameters $m = 2^{32}$, $a = 1664525$ and $c = 1013904223$.
- Using your generator, perform a one-dimensional random walk where the $N = 1000$ steps Δx can be any number between (-0.5, 0.5)
- Verify that you obtain the same trajectory every time you execute your code.
- Perform different walks by using different seeds, for example based on `time.time()` or `os.getpid()`.

The Mersenne Twister is a fast, uncorrelated, random number generator with a long period. It a popular method of and is available using `numpy.random.random()`, which uses it's own self-seeding methods. From now on, you can used `numpy.random.random()` to obtain uniform random numbers. In a later task, it will turn out to be useful to know that uniform random numbers from an interval (a, b) have a standard deviation of $\sigma = \frac{b-a}{\sqrt{12}}$.

Often, one wants to obtain a random number from a Gaussian distribution. The *Box–Muller* transform allows one to obtain Gaussian random numbers from uniformly random numbers. The pitfall of this method is that it works in pairs. It needs two uniform random numbers u_1 and u_2 as an input, however, it also yields two random numbers n_1 and n_2 that follows a normal distribution.

The transform is quite straight-forward:

$$n_1 = \sqrt{-2\log(u_1)}\cos(2\pi u_2) \tag{2}$$

$$n_2 = \sqrt{-2\log(u_1)}\sin(2\pi u_2) \tag{3}$$

where n_1 and n_2 are random numbers that follows a zero-mean normal distributed with a standard-deviation of 1.

Task (2 points)

- Implement the Box-Muller transform to generate a histogram of $N = 10000$ random numbers that follows a Gaussian distribution with a mean of $\mu = 2.0$ and a standard-deviation of $\sigma = 5.0$. Normalize your histogram and include the expected analytical curve for this distribution.
- Generate $N = 1000$ random Gaussian velocity vectors $\mathbf{v} = (v_x, v_y, v_z)$ which have elements v_x, v_y and v_z taken from a Gaussian distribution with mean $\mu = 0$ and standard-deviation of $\sigma = \sqrt{\frac{k_\mathrm{B}T}{m}}$ (you can use $m = 1$ and $k_\mathrm{B}T = 1$, but verify your results with different values)
- Plot the distribution of the speeds $|\mathbf{v}|$ obtained from your random vectors and compare with the analytical three-dimensional Maxwell-Boltzmann distribution.

4 Langevin thermostat

As we have seen on the last worksheet, even though the *velocity rescaling thermostat* is able to keep the temperature constant, this is not actually the same as simulating the canonical (N, V, T)-ensemble, as the Maxwell-Boltzmann distribution of the velocities is destroyed.

A thermostat that does allow to simulate the canonical ensemble is the *Langevin thermostat*. In the Langevin thermostat, at each time step every particle is subject to a random (stochastic) force and to a frictional (dissipative) force. There is a precise relation between the stochastic and dissipative terms, which comes from the so-called fluctuation-dissipation theorem, and ensures sampling of the canonical ensemble. The equation of motion of a particle is thus modified to

$$m\mathbf{a}_i = \mathbf{F}_i - m\gamma\mathbf{v}_i + \mathbf{W}_i(t). \tag{4}$$

with the introduction of a friction coefficient γ with the units of an inverse time and a random force $\mathbf{W}_i$ acting on particle i that is uncorrelated in time and between particles, and otherwise characterized by its variance:

$$\langle \mathbf{W}_i(t) \cdot \mathbf{W}_j(t') \rangle = \delta_{ij}\delta\left(t - t'\right) 6k_B mT\gamma. \tag{5}$$

The modified Velocity-Verlet algorithm for a Langevin thermostat is

$$\mathbf{x}_i(t + \Delta t) = \mathbf{x}_i(t) + \Delta t\mathbf{v}_i(t)(1 - \Delta t\frac{\gamma}{2}) + \frac{\Delta t^2}{2m}\mathbf{G}_i(t) \tag{6}$$

$$\mathbf{v}_i(t + \Delta t) = \frac{\mathbf{v}_i(t)(1 - \Delta t\frac{\gamma}{2}) + \frac{\Delta t}{2m}(\mathbf{G}_i(t) + \mathbf{G}_i(t + \Delta t))}{(1 + \Delta t\frac{\gamma}{2})} \tag{7}$$

where $\mathbf{G}_i$ is the total force: $\mathbf{G}_i = \mathbf{F}_i + \mathbf{W}_i$.

Task (3 points)

- For the purposes of this worksheet, the random vector $\mathbf{W}_i(t)$ can follow a uniform distribution. Make sure the mean is zero and that each of the three components have a standard deviation of $\sigma = \sqrt{2mk_\mathrm{B}T\gamma/\Delta t}$ where Δt is the simulation timestep.
- Write a function `step_vv_langevin()` that implements Eq. (6) and (7). Remember to update the force between the two-half steps.
- Extend the template to implement the Langevin thermostat, you can use $\gamma = 0.3$.
- Plot the temperature vs. time.
- Plot the distribution of speeds and compare with the Maxwell-Boltzmann distribution.

5 Andersen thermostat

In the lecture on thermostats you learned about the Andersen thermostat, where an NVT ensemble is generated by replacing the velocities of randomly selected particles by a random velocity drawn from a Maxwell-Boltzmann distribution corresponding to the desired temperature T. This resembles the system's coupling to an infinite heat bath at temperature T.

The probability P for a particle to be selected to undergo this procedure during a time step Δt is $P(\nu, \Delta t) = \nu \Delta t$, where ν is the frequency of (virtual) stochastic collisions with the heat bath.

Task (1 point)
Can a simulation with an Anderson thermostat in general be used to determine dynamical properties of a system, e.g. diffusion coefficients?

In every timestep of the simulation, the same set of commands are sequentally executed. In order to probabilistically perform a task like the velocity assigment step of the Andersen thermostat, one can use random numbers in the following manner:

1. given that `prob` is the probability of performing an action
2. pick a uniform random number `rand` from (0,1)

3. if `rand` $<$ `prob`, perform the action
4. else do not perform the action

Task (3 points)

- Write a function that creates a random velocity vector that follows Maxwell-Boltzmann statistics for a selected particle. You can base this on the code from the first task.
- Write a function `step_vv_anderson()` that attempts to assign a new velocity to the particle during every timestep. The probability of the assignment being performed is given by $\nu \Delta t$ with $\nu = 0.1$
- Extend the template to implement the Andersen thermostat.
- Plot the temperature vs. time.
- Plot the distribution of speeds and compare with the Maxwell-Boltzmann distribution.

Hints

- It is a good idea to verify that the velocity distribution is correct before moving on to assigning the velocities
- You should perform the attempt at the last stage of the velocity verlet update

6 Berendsen thermostat

Another well-known thermostat is the Berendsen thermostat. However, just like the velocity-rescaling thermostat, upon which it generalizes, this method doesn't yield the canonical ensemble exactly. In order to control temperature, velocities are scaled in every time step by a factor λ with

$$\lambda = \sqrt{1 + \frac{\Delta t}{\tau_T} \left(\frac{T_{des}}{T_{act}} - 1 \right)} \,, \tag{8}$$

where Δt is the time step of the simulation, T_{des} and T_{act} the desired and actual system temperature, respectively, and $\tau_T \geq \Delta t$ is a constant.

Task (2 points)

- Implement this in `step_vv_anderson()` using $\tau_T = 3\Delta t$
- Extend the template to implement the Berendsen thermostat.
- Plot the temperature vs. time.
- Plot the distribution of speeds and compare with the Maxwell-Boltzmann distribution.

Hints

- You should perform the velocity replacement attempt at the last stage of the velocity verlet update.

7 Diffusion

In this analysis section you will need simulation trajectories for the three thermostats. E-mail your tutor for sample trajectories if you were unable to perform them.

In this task, you will be asked to calculate the diffusion coefficient of a single simulated particle.

One method consists of measuring the mean-squared-displacement (MSD) and performing a fit using

$$\langle \Delta x^2 \rangle := \langle \Delta \left| \vec{x} \right|^2 \rangle = 2Dd\Delta t, \tag{9}$$

where D is the diffusion coefficient and $d = 3$ the dimensionality of the system. Note that here Δt is not related to the integrator's time step.

The MSD can be obtained from a single trajectory covering T time by subdividing it into N sub-trajectories of duration $\Delta t = \frac{T}{N}$ each. At each time subdivision $\Delta t \in (0, T]$,

$$\langle \Delta x^2 \rangle_i = |\vec{x}(i\Delta t) - \vec{x}((i-1)\Delta t)|^2 \tag{10}$$

is averaged over the N sub-trajectories to give

$$\langle \Delta x^2 \rangle(\Delta t) = \frac{1}{N} \sum_{i=1}^{N} \langle \Delta x^2 \rangle_i. \tag{11}$$

For example, there are $N = 5$ sub-trajectories of length $\Delta t = 10$ that can be extracted from a single trajectory of length $T = 50$ or $N = 7$ at $\Delta t = 7$ (discarding the last few time steps if N does not divide T without remainder).

Your task is to determine the squared displacements $\langle \Delta x^2 \rangle (\Delta t)$ that occur during a time Δt. Find the diffusion coefficient by fitting the data to the diffusion model.

Task (3 points)

- Open the simulation trajectories and plot the MSD resulting from the different thermostats. Include error bars for the average from eq. (11).
- For the case of the Andersen and Langevin thermostats, perform a fit in the linear region to determine the diffusion coefficient.
- Why do the Andersen and Langevin give different diffusion coefficients? Try different values for γ and ν and see how they connect to D.
- Why does the Berendsen thermostat not give a reasonable diffusion coefficient?

An alternate way to obtain a diffusion coefficient from a dynamical simulation is from the velocity autocorrelation function (VACF) via the Green-Kubo relation:

$$D = \int_0^\infty \langle v(t) \cdot v(0) \rangle \, \mathrm{d}t. \tag{12}$$

Numerically, you can determine autocorrelation functions either directly via `numpy.convolve` or `scipy.signal.convolve` or spectrally via `numpy.fft` or `scipy.signal.fftconvolve`.

Task (3 points)

- Open the velocity datafile generated by the three thermostats and plot the VACFs.
- For the Langevin thermostat, numericaly integrate the VACF to determine the diffusion coefficient.
- In which cases can one expect the diffusion coefficients from MSD and VACF to agree?