Worksheet 5: Monte Carlo

Kartik Jain Rudolf Weeber

January 9, 2019
Institute for Computational Physics, University of Stuttgart

Contents

1 General Remarks

- Deadline for the report is **Monday, January 21, 2019**
- On 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 to the lecture, but does not do the tutorials.
- To hand in your report, send it to your tutor via email
  - Rudolf (`weeber@icp.uni-stuttgart.de`)
  - Kartik (`kjain@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.

On this worksheet, you will employ the Monte-Carlo method to integrate a function and to simulate the Ising model. You will also learn how to do proper error analysis on the results.

2 Simple Sampling – Integration

In general, the Monte-Carlo method can be used to compute expectation values of the type

$$\langle A \rangle = \frac{\int_\Phi A(\phi) P(\phi) \mathrm{d}\phi}{\int P(\phi) \mathrm{d}\phi} \tag{1}$$

where $A(\phi)$ is an arbitrary function and $P(\phi)$ is the a-priori probability distribution of the states ϕ.

In statistical physics, Φ is the set of possible states of a system, P is the Boltzmann distribution $P(\phi) = e^{-\beta H(\phi)}$ and $A(\phi)$ is an observable of the system.

The fundamental idea of the Monte-Carlo method is to approximate equation (1) by replacing the continuous integrals by discrete sums over randomly generated, uniformly distributed states of the system:

$$\langle A \rangle \approx \frac{\sum_N A(\phi_i) P(\phi_i)}{\sum_N P(\phi_i)}, \ \{\phi_i\} \text{ uniformly distributed} \tag{2}$$

This is the so-called *simple sampling*.

This method is employed for the Monte-Carlo integration of functions. In that case, we set $P(\phi) = 1$ and get

$$\int_a^b f(x) \mathrm{d}x \approx \frac{b-a}{N} \sum_N f(x_i), \ \{x_i\} \in [a, b] \text{ uniformly distributed} \tag{3}$$

The first task is to integrate the function $f(x)$ on the interval $[0.1, 10]$:

$$f(x) = \left(-2x^2 \sin x \cos x - 2x \left(\sin x\right)^2\right) * \exp\left(-x^2 \left(\sin x\right)^2\right) \tag{4}$$

Task (2 points)

- Write a Python function `f(x)` that computes the function $f(x)$.
- Plot $f(x)$ on the interval $[0.1, 50]$.
- Write a function that computes the exact solution of I. To get the closed form of $\int f(x)\,dx$, use the sympy Python package for symbolic calculations. See `http://docs.sympy.org/latest/modules/integrals/integrals.html` for examples. You can also use the `lambdify` function from sympy to obtain a function which takes a Numpy array as input.

Task (2 points)

- Write a Python function `simple_sampling(f,a,b,N)` that performs N steps of a simple sampling Monte-Carlo integration of a Python function `f` in the interval $[a, b]$. The function should return the estimate of the integration as well as the statistical error of the estimate (*i.e.* the error computed by error estimation). Keep in mind that you cann pass a Python function such as your implementation of $f(x)$ as an argument to a different Python function.
- Write a Python program that uses the function to compute I for $N = 2^i$ where $2 \leq i \leq 20$.
- Compute the actual error of the integration (*i.e.* the difference between the computed value and the exact solution).
- Plot the actual error and the statistical error against the number of integration steps N.

Hint To compute the statistical error of the estimate, use the fact that the different samples $f(x_i)$ are uncorrelated in the case of simple sampling.

3 Importance Sampling – Metropolis-Hastings Algorithm

Unfortunately, simple sampling is often very ineffective. For example, imagine that ϕ is the state of a system of Lennard-Jones particles with a not-too-low density (*i.e.* ϕ contains the coordinates of all particles). In simple sampling, one would try to randomly generate the positions of all particles. In that case, the probability to generate a state with a very high energy (where particles overlap) is very high, and accordingly $e^{-\beta H(\phi)}$ is

very small. Only when a state of the system is generated where not all particles overlap, this will significantly contribute to the mean value.

To overcome this problem, one can use *importance sampling*. In this method, the idea is to generate the states $\{\phi_i\}$ of the system according to the probability distribution $P(\phi)$. When this can be achieved, equation (2) can be rewritten as

$$\langle A \rangle \approx \frac{1}{N} \sum_N A(\phi_i), \ \{\phi_i\} \text{ P-distributed} \tag{5}$$

One algorithm to generate samples from a given probability distribution $P(\phi)$ is the Metropolis-Hastings algorithm. We do not derive the algorithm here, but only define it. The algorithm generates a sequence of states $\{\phi_i\}$ that is distributed according to $P(\phi)$ in the limit of many steps.

Assume that we have given a state ϕ_i of the system. Then the next state in the sequence can be generated by the following steps:

1. Perform a *trial move*, *i.e.* propose a new state ϕ' according to some prescription (see below).
2. Draw a uniformly distributed random number $r \in [0, 1[$.
3. If $r < \min(1, \frac{P(\phi')}{P(\phi)})$, *accept* the new state, otherwise *reject* it.
4. When the state is accepted, the next state is $\phi_{i+1} = \phi'$.
5. When the state is rejected, the next state is $\phi_{i+1} = \phi_i$.

Examples for a trial move are:

- To generate x-values with a given distribution $P(x)$, a trial move could be to simply add a random number $r \in [-\Delta x, +\Delta x]$ to x.
- In the Ising model, the trial move could be to flip one or more randomly selected spins.
- In a many-particle system, the trial move could be to randomly choose a particle and move it to a random position in a box around the last particle position.

The interesting thing about Monte-Carlo simulations is that you have great freedom in how a trial move can look like. There are only two necessary conditions for the trial move:

1. The trial moves have to be *symmetric*, *i.e.* $P_{\text{trial}}(\phi_i \to \phi_j) = P_{\text{trial}}(\phi_j \to \phi_i)$ where $P_{\text{trial}}(\phi_i \to \phi_j)$ is the probability to propose state ϕ_j when the initial state is ϕ_i.
2. The trial moves have to be "ergodic", *i.e.* it must be possible to reach any possible state of the system via a finite sequence of trial moves.

The goal of the algorithm is to explore the complete phase space as quickly as possible. Therefore, in practice, not all possible trial moves are useful.

One important property of the algorithm to keep in mind is the *acceptance rate*, *i.e.* the probability that a trial move is actually accepted. On the one hand, when a trial move changes the state very much, the acceptance rate may become very low, so that the system state changes only very rarely, and it takes a very long time for the system to get anywhere. An extreme example for this would be to simply generate a new random state in every step (as in simple sampling). On the other hand, when a trial move proposes a state that is very similar to the previous state, the acceptance rate may become close to 1, however, the state still does change only very slowly, as the trial moves are very small. To get the fastest sampling of the phase space, the trial moves should be somewhere in between. In general, keeping the acceptance rate at $\approx 30\%$ is a good idea.

Many trial moves have a parameter that allows to tune the acceptance rate, as for example the parameter Δx in the first example above.

Task (4 points)

- Write a Python function `metropolis(N,P,trial_move,phi0)` that uses the Metropolis-Hastings-algorithm to generate `N` samples from the probability distribution `P`. `phi0` is the initial state, and `trial_move` is a function that proposes a new state given the previous state `phi`. The function shall return the generated states in a Python list as well as the acceptance rate.
- Write a Python program that uses the function to generate N samples of x that are distributed according to the function $g(x) = \exp\left(-x^2\right)/\sqrt{\pi}$. The trial move should be a function that adds a uniformly distributed random number $r \in [-\Delta x, +\Delta x]$ to x.
- Generate 4×100000 samples of the function $g(x)$ with move ranges $\Delta x \in \{0.1, 1.0, 10.0, 100.0\}$.
- Plot a distribution histogram with 100 bins in the range $[-5, 5]$ of the Metropolis samples and compare them to $g(x)$.
- Which move range Δx gives the best results?
- What is the acceptance rate in that case?

Hint To generate the histogram, use the Python function `matplotlib.pyplot.hist`.

4 Simulating the Ising Model

In the following, we will perform simulations of the two-dimensional Ising model on a $(L \times L)$ square lattice. $\sigma_{i,j}$ denotes the spin at lattice position (i, j).

The (total) energy of the system is defined by

$$E = \frac{1}{2} \sum_{i=0}^{L-1} \sum_{j=0}^{L-1} E_{i,j} \tag{6}$$

$$E_{i,j} = -\sigma_{i,j}(\sigma_{i-1,j} + \sigma_{i+1,j} + \sigma_{i,j-1} + \sigma_{i,j+1}) \tag{7}$$

The system uses periodic boundary conditions, *i.e.*

$$\sigma_{-1,j} = \sigma_{L-1,j} \qquad \sigma_{L,j} = \sigma_{0,j}$$
$$\sigma_{i,-1} = \sigma_{i,L-1} \qquad \sigma_{i,L} = \sigma_{i,0}$$

As an observable, we are interested in the (mean) *energy per site* e, which is defined by

$$e = \langle \frac{E}{L^2} \rangle$$

and the (mean) *magnetization per site* m

$$m = \langle |\mu| \rangle \tag{8}$$

$$\mu = \frac{1}{L^2} \sum_{i=0}^{L-1} \sum_{j=0}^{L-1} \sigma_{i,j} \tag{9}$$

4.1 Exact Summation

Task (4 points)

- Implement a program that computes the value of the mean energy per site e and the mean magnetization per site m of the two-dimensional Ising model with (4×4)-spins by exact summation for a given temperature T.
- Plot e and m against the temperature T where $T \in \{1.0, 1.1, \ldots, 4.9, 5.0\}$.

4.2 Monte-Carlo Simulation

The program that you will write in this task will be reused in the next worksheet, so write it cleanly and document it well.

The task is to write and use a program that does a Metropolis MC simulation of the two-dimensional Ising model. The trial move should be to flip a randomly chosen spin.

Task (4 points)

- Implement a Python program that computes the mean energy per site $\langle e \rangle$ and mean magnetization per site $\langle m \rangle$ of the two-dimensional Ising model with $(L \times L)$-spins via the Metropolis Monte-Carlo algorithm.
- Compute and plot the mean magnetization and the mean energy of the (4×4)-Ising model with 10000 Metropolis Monte-Carlo steps at temperature T where $T \in \{1.0, 1.1, \ldots, 4.9, 5.0\}$.
- Compare the results to the results of the exact summation.

Hints Note that it is not necessary to recompute the total energy of the system in every step. Instead, you can compute the change of the energy caused by a single spin flip with help of equation (7).

4.3 Error Analysis

From the lecture's home page, you can download the file `janke02.pdf` of an article by Wolfhard Janke. Section 3 of this article is what the lecture on error analysis was based on, and all that needs to be done in this part of the worksheet can be learned from the article.

Task (3 points)

- Implement a Python function that performs automatic error analysis of a given time series of an observable. The function should return:
  - the observable's mean value $\bar{\mathcal{O}}$
  - the estimated autocorrelation time $\tau_{\mathcal{O},\text{int}}$
  - the effective statistics N_{eff}
  - the error $\epsilon_{\bar{\mathcal{O}}}$ of the observable's mean value

  You can calculate the autocorrelation time using either Binning analysis (sec. 3.5 in the article) or autocorrelation (sec. 3.2 in the article). Once you know the autocorrelation time, you can determine the effective statistics, which in turn is needed for the error of the mean.

- One distribution for which the autocorrelation time is known is the bivariate Gaussian. You can generate it using

```
def bivariate_gaussian(N, rho):
  e = numpy.random.normal(size=N)
  for i in range(1, N):
    e[i] = rho*e[i-1] + numpy.sqrt(1-rho**2)*e[i]
  return e
```

  and it is known that $\tau_{\text{int}} = \frac{1}{2}\frac{1+\rho}{1-\rho}$. Use `N=1e6` and `rho=0.99005`. Apply the error analysis function the data generated like this and make sure you obtain the correct autocorrelation time before you proceed (due to statistical flucuations, you will never get an exact match however).

- If you chose to do Binning analysis, the choice of k, the block size, is a free parameter. The resulting τ_{int} unfortunately depends on the choice of this parameter. The article says that k should be chosen such that $k \gg \tau_{\text{int}}$. At the same time, you want the number of blocks $N_b \gg 1$ for good statistics. A simple way out of this dilemma is to iterate over all $1 \leq k \leq N/10$ and using the largest τ_{int} that you find.

Now you can apply your analysis function to your Ising simulation results.

Task (1 point)

- Determine the statistical errors of your estimates of the mean magnetization per site m and mean energy per site e using the function you just wrote. Make sure to correctly deal with the correlated nature of your data by considering that $N_{\text{eff}} \neq N$.

- Add the corresponding error bars to the plot from section 4.2.