Understanding basic concepts by using Metropolis algorithm

1 Integration of one-variable functions with Metropolis algorithm

Let us start with the integration of a one-variable function with the Metropolis algorithm [1]. In particular, we will consider the simplest example we can imagine: the Gaussian integral, $S(x) = x^2/2$. This very basic example contains essentially all important ingredients; all other cases are, ultimately, just technical improvements of this example.

Of course we can handle the Gaussian integral analytically. Also there is a much better algorithm for generating Gaussian random numbers. We use it just for an educational purpose.

1.1 Metropolis Algorithm

Let us consider the weight $e^{-S(x)}$, where $S(x)$ is a continuous function of $x \in \mathbb{R}$ bounded from below. We further assume that $\int e^{-S(x)}dx$ is finite. The Metropolis algorithm gives us a chain $x^{(0)} \to x^{(1)} \to x^{(2)} \to \cdots$ which satisfies the conditions listed above:

1. Randomly choose $\Delta x \in \mathbb{R}$, and shift $x^{(k)}$ as $x^{(k)} \to x' \equiv x^{(k)} + \Delta x$. (Here Δx and $-\Delta x$ must appear with the same probability, so that the detailed balance condition is satisfied. For example we use the uniform random number between $\pm c$, where $c > 0$ is the 'step size.')

2. Metropolis test: Generate a uniform random number r between 0 and 1. If $r < e^{-\Delta S}$, where $\Delta S = S(x') - S(x^{(k)})$, then $x^{(k+1)} = x'$, i.e. the new value is 'accepted.' Otherwise $x^{(k+1)} = x^{(k)}$, i.e. the new configuration is 'rejected.'

3. Repeat the same for $k+1, k+2, \cdots$.

It is an easy exercise to see that all conditions explained above are satisfied:

- It is a Markov Chain, because the past history is not referred either for the selection of Δx or the Metropolis test.

- It is irreducible; for example, any x and x', by taking n_s large we can make $\frac{x-x'}{n}$ to be in $[-c, c]$, and there is a nonzero probability that $\Delta x = \frac{x-x'}{n}$ appears n times in a row and passes the Metropolis test every time.

- For any x, there is a nonzero probability of $\Delta x = 0$. Hence the period is one for any x.
- If $|x - x'| > c$, $P[x \to x'] = P[x' \to x] = 0$. When $|x - x'| \leq c$, both $\Delta x = x' - x$ and $\Delta x' = x - x'$ are chosen with probability $\frac{1}{2c}$. Let us assume $\Delta S = S[x'] - S[x] > 0$. Then the change $x \to x'$ passes the Metropolis test with probability $e^{-\Delta S}$, while $x' \to x$ is always accepted. Hence $P[x \to x'] = \frac{e^{-\Delta S}}{2c}$ and $P[x' \to x] = \frac{1}{2c}$, and hence $P[x \to x'] = P[x' \to x] = \frac{e^{-S[x']}}{2c}$.

1.1.1 How it works

Let me show a sample code written in C:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

int main(void){
  int iter,niter=100;
  int naccept;
  double step_size=0.5e0;
  double x,backup_x,dx;
  double action_init, action_fin;
  double metropolis;

  srand((unsigned)time(NULL));

  /*********************************/
  /* Set the initial configuration */
  /*********************************/
  x=0e0;
  naccept=0;
  /*************/
  /* Main loop */
  /*************/
  for(iter=1;iter<niter+1;iter++){
    backup_x=x;
    action_init=0.5e0*x*x;

    dx = (double)rand()/RAND_MAX;
    dx=(dx-0.5e0)*step_size*2e0;
```

```
    x=x+dx;

    action_fin=0.5e0*x*x;
    /*******************/
    /* Metropolis test */
    /*******************/
    metropolis = (double)rand()/RAND_MAX;
    if(exp(action_init-action_fin) > metropolis)
      /* accept */
      naccept=naccept+1;
    else
      /* reject */
      x=backup_x;
    /***************/
    /* data output */
    /***************/
    printf("%f\n",x);}
}
```

Let me explain line by line. Firstly, by

```
srand((unsigned)time(NULL));
```

the seed for the random number generator is set. A default random number generator is used, by using the system clock time to set the seed randomly. For more serious simulations, it is better to use a good generator, say the Mersenne twister.

Then we specify an initial configuration; here we took $x^{(0)} = 0$. **naccept** counts how many times the new configurations are accepted.

```
x=0e0;
naccept=0;
```

Then we move on to the main part of the simulation, which is inside the following loop:

```
for(iter=1;iter<niter+1;iter++){ .... }
```

Here, **iter** corresponds to k, and **niter** is the number of configurations we will collect during the simulation.

Inside the loop, the first thing we have to do is to save the value of $\mathbf{x}= x^{(k)}$, because it may or may not be updated:

```
backup_x=x;
```

Then **action_init**= $S(x^{(k)})$ is calculated.

Now we have to generate a random variation **dx**= Δx with an appropriate step size, and shift x to $x' = x^{(k)} + \Delta x$. We can generate a uniform random number in $[0, 1]$ by **rand()/RAND_MAX**. From this we can easily get $-c < \Delta x < c$.

```
dx = (double)rand()/RAND_MAX;
dx=(dx-0.5e0)*step_size*2e0;
x=x+dx;
```

By using x', **action_fin**= $S(x')$ is calculated. Note that **x** and **backup_x** in the code correspond to x' and $x^{(k)}$.

Finally we perform the Metropolis test:

```
metropolis = (double)rand()/RAND_MAX;
if(exp(action_init-action_fin) > metropolis)
  /* accept */
  naccept=naccept+1;
else
  /* reject */
  x=backup_x;
```

metropolis is a uniform random number in $[0, 1]$, which corresponds to r. Depending on the result of the test, we accept or reject x'.

We emphasize again that *all MCMC simulations have exactly the same structure*; there are many fancy algorithms, but essentially, they are all about improving the step $x \to x + \Delta x$.

We take $x^{(0)} = 0$, and Δx to be uniform random number between -0.5 and 0.5. (As we will see later, this parameter choice is not optimal.) In Fig. 1, we show the distribution of $x^{(1)}, x^{(2)}, \cdots, x^{(n)}$, for $n = 10^3, 10^5$ and 10^7. We can see that the distribution converges to $e^{-x^2/2}/\sqrt{2\pi}$. The expectation values $\langle x \rangle = \frac{1}{n}\sum_{k=1}^{n} x^{(k)}$ and $\langle x^2 \rangle = \frac{1}{n}\sum_{k=1}^{n} \left(x^{(k)}\right)^2$ are plotted in Fig. 2. As n becomes large, they converge to the right values, 0 and 1.

Note that the step size c should be chosen so that the acceptance rate is not too high, not too low. If c is too large, the acceptance rate becomes extremely low, then the configurations are rarely updated. If c is too small, the acceptance rate is almost 1, but the change of configuration at each step is extremely small. In both cases, huge amount of configurations are needed in order to approximate the integration measure accurately. The readers can confirm it by changing the step size in the sample code. (We will demonstrate it in Sec. 1.2.)

Typically the acceptance rate 30% – 80% is good. But it can heavily depend on the detail of the system and algorithm.

A bad example

It is instructive to see a wrong example. Let us take Δx from $\left[-\frac{1}{2}, 1\right]$, so that the detailed balance condition is violated; for example $0 \to 1$ has a finite probability but $1 \to 0$

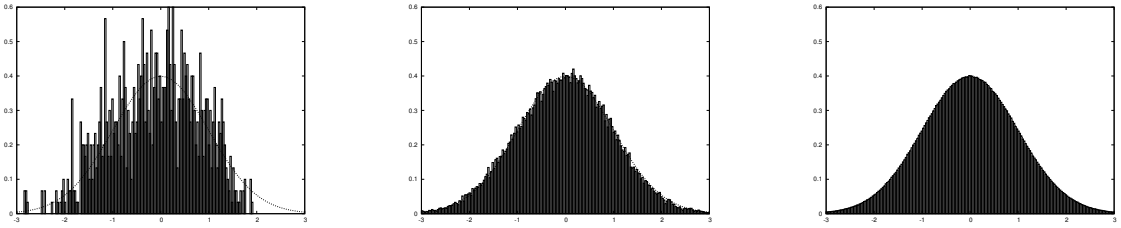

Figure 1: The distribution of $x^{(1)}, x^{(2)}, \cdots, x^{(n)}$, for $n = 10^3, 10^5$ and 10^7, and $\frac{e^{-x^2/2}}{\sqrt{2\pi}}$.

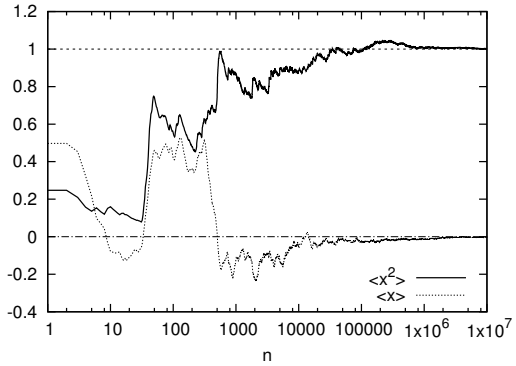

Figure 2: $\langle x \rangle = \frac{1}{n} \sum_{k=1}^{n} x^{(k)}$ and $\langle x^2 \rangle = \frac{1}{n} \sum_{k=1}^{n} \left(x^{(k)} \right)^2$. As n becomes large, they converge to the right values, 0 and 1.

cannot happen. Then, as we can see from Fig 3, the chain does not converge to the right probability distribution.

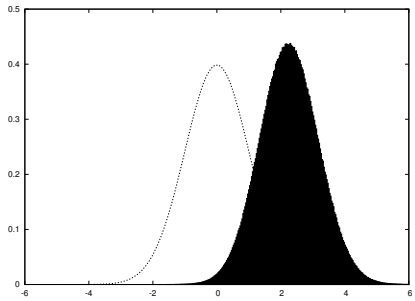

Figure 3: The distribution of $x^{(1)}, x^{(2)}, \cdots, x^{(n)}$ for $n = 10^7$, with *wrong* algorithm with $\Delta x \in \left[-\frac{1}{2}, 1\right]$. The dotted line is the right Gaussian distribution $\frac{e^{-x^2/2}}{\sqrt{2\pi}}$.

1.2 Autocorrelation and Thermalization

In MCMC, $x^{(k+1)}$ is obtained from $x^{(k)}$. In general, they are correlated. This correlation is called *autocorrelation*. The autocorrelation can exist over many steps. The autocorrelation length depends on the detail of the theory, algorithm and the parameter choice. Because of the autocorrelation, some cares are needed.

In the above, we have set $x^{(0)} = 0$, because we knew it is 'the most important configuration'. What happens if we start with an atypical value, say $x^{(0)} = 100$? It takes some time for typical configurations to appear, due to the autocorrelation. The history of the Monte Carlo simulation with this initial condition is shown in Fig. 4. The value of x eventually reaches to 'typical values' $|x| \lesssim 1$ — we often say 'the configurations are thermalized' (note that the same term 'thermalized' has another meaning as well, as we will see shortly) —, but a lot of steps are needed. If we include 'unthermalized' configurations when we estimate the expectation values, we will suffer from huge error unless the number of configurations are extremely large. We should discard unthermalized configurations, say $n \lesssim 1000$.[1]

In generic, more complicated situation, we don't a priori know what the typical configurations look like. Still, whether the configurations are thermalized or not can be seen by looking at several observables. As long as they are changing monotonically, it is plausible that the configuration is moving toward a typical one. When they start to oscillate around certain values (see Fig. 5, we can see a fluctuation around $x = 0$), it is reasonable to think the configuration has been thermalized.

Fig. 5 is a zoom-up of Fig. 4, from $n = 1000$ to $n = 2000$. We can see that the values of x can be strongly correlated unless they are 20 or 30 step separated. (We will give a quantitative method in Sec. 1.2.1.) When we estimate the statistical error, we should not

[1]Number of steps needed for the thermalization is sometimes called 'burn-in time' or 'mixing time'.

treat all configuration to be independent; rather we have only 1 independent configuration every 20 or 30 steps. Note also that we need sufficiently many independent configurations in order to estimate the expectation values reliably. We often say the simulation has been thermalized when we have sufficiently many independent configurations so that the expectation values are stabilized.

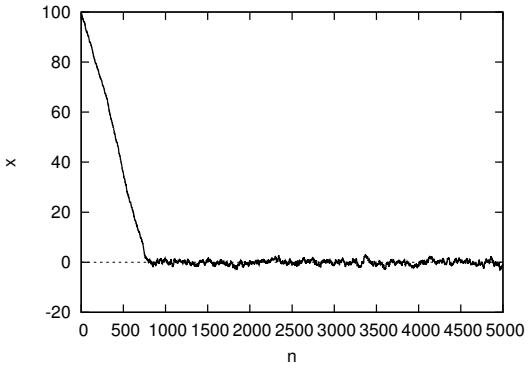

Figure 4: Monte Calro history of the Gaussian integral with the Metropolis algorithm, $\Delta x \in [-0.5, 0.5]$. We took the initial value to be a very atypical value, $x = 100$. It takes a lot of steps to reach typical values $|x| \sim 1$.

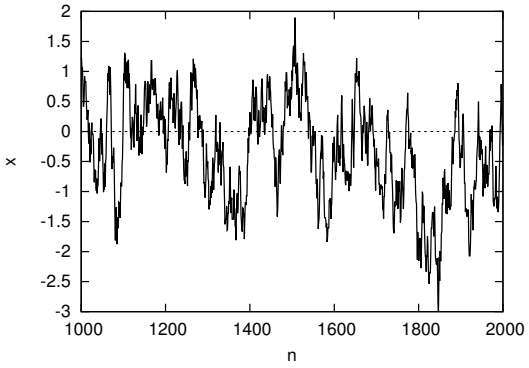

Figure 5: A zoom-up of Fig. 4, from $n = 1000$ to $n = 2000$. The values of x can be strongly correlated unless they are at least 20 or 30 steps separated.

1.2.1 Jackknife method

The Jackknife method provides us with a simple way to estimate the autocorrelation length. Here we assume the quantity of interest can be calculated for each sample.[2] For

[2]The correlation function is in this class. The mass of particle excitation is not; we need to calculate two-point function by using many samples and then extract the mass from its exponential decay.

more generic cases, see Appendix **??**.

In the Jackknife method, we first divide the configurations to bins with width w; the first bin is $\{x^{(1)}\}, \{x^{(2)}\}, \cdots, \{x^{(w)}\}$, the second bin is $\{x^{(w+1)}\}, \{x^{(w+2)}\}, \cdots, \{x^{(2w)}\}$, etc. Suppose we have n bins. Then we define the average of an observable $f(x)$ with k-th bin removed,

$$\overline{f}^{(k,w)} \equiv \frac{1}{(n-1)w} \sum_{j \notin k\text{-th bin}} f(x^{(j)}). \tag{1}$$

The average value

$$\overline{f} \equiv \frac{1}{n} \sum_k \overline{f}^{(k,w)} \tag{2}$$

is the same as the average of all samples, $\frac{1}{nw}\sum_j f(x^{(j)})$, for the class of quantities we are discussing. The *Jackknife error* is defined by

$$\Delta_w \equiv \sqrt{\frac{n-1}{n} \sum_k \left(\overline{f}^{(k,w)} - \overline{f}\right)^2}. \tag{3}$$

By using

$$\tilde{f}^{(k,w)} \equiv \frac{1}{w} \sum_{j \in k\text{-th bin}} f(x^{(j)}), \tag{4}$$

we can easily see

$$\overline{f}^{(k,w)} - \overline{f} = \frac{\overline{f} - \tilde{f}^{(k,w)}}{n-1}. \tag{5}$$

Hence

$$\Delta_w \equiv \sqrt{\frac{1}{n(n-1)} \sum_k \left(\tilde{f}^{(k,w)} - \overline{f}\right)^2}. \tag{6}$$

Namely Δ_w is the standard error obtained by treating $\tilde{f}^{(k,w)}$ to be independent samples.

Typically, as w becomes large, Δ_w increases and then becomes almost constant at certain value of w, which we denote by w_c. This w_c and Δ_{w_c} give good estimates of the autocorrelation length and the error bar.

It can be understood as follows. Let us consider two bin sizes w and $2w$. Then

$$\tilde{f}^{(k,2w)} = \frac{\tilde{f}^{(2k-1,w)} + \tilde{f}^{(2k,w)}}{2}, \tag{7}$$

$$\Delta_{2w} = \sqrt{\frac{1}{\frac{n}{2}\left(\frac{n}{2}-1\right)} \sum_{k=1}^{n/2} \left(\tilde{f}^{(k,2w)} - \overline{f}\right)^2}$$

$$= \sqrt{\frac{4}{n(n-2)} \sum_{k=1}^{n/2} \left(\frac{\left(\tilde{f}^{(2k-1,w)} - \overline{f} \right)}{2} + \frac{\left(\tilde{f}^{(2k,w)} - \overline{f} \right)}{2} \right)^2}. \tag{8}$$

If w is sufficiently large, $\tilde{f}^{(2k-1,w)} - \overline{f}$ and $\tilde{f}^{(2k,w)} - \overline{f}$ should independent, and the cross-term $\left(\tilde{f}^{(2k-1,w)} - \overline{f} \right) \cdot \left(\tilde{f}^{(2k,w)} - \overline{f} \right)$ should average to zero after summing up with respect to sufficiently many k. Then

$$\Delta_{2w} \sim \sqrt{\frac{1}{n^2} \sum_{k=1}^{n} \left(\tilde{f}^{(k,w)} - \overline{f} \right)^2} \sim \Delta_w. \tag{9}$$

In this way, Δ_w becomes approximately constant when w is large enough so that $\tilde{f}^{(k,w)}$ can be treated as independent samples. (Note that n must also be large for the above estimate.) Δ_w is the standard error of these 'independent samples'.

In Fig. 6, $\langle x^2 \rangle$ and Jackknife error Δ_w are shown by using first 50000 samples. We can see that $w_c = 50$ is a reasonably safe choice; $w_c = 20$ is already in the right ballpark. In Fig. 7, bin-averaged values with $w = 50$ are plotted. They do look independent. We obtained $\langle x^2 \rangle = 0.982 \pm 0.012$, which agree reasonably well with the analytic answer, $\langle x^2 \rangle = 1$.

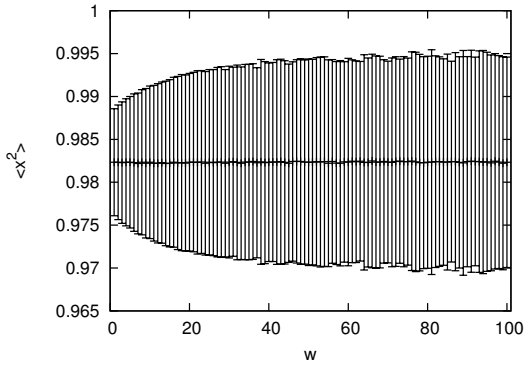

Figure 6: $\langle x^2 \rangle$ and Jackknife error Δ_w with 50000 samples.

1.2.2 Tuning the simulation parameters

In order to run the simulation efficiently, we should tune parameters so that we can obtain *more independent samples with less cost*.[3]

In the current example (Gaussian integral with uniform random number), when the step size c is too large, unless $\Delta x \lesssim 1$ the configuration is rarely updated; the acceptance

[3]In parallelized simulations, the notion of the *cost* is more nontrivial because time is money. Sometimes you may want to invest more electricity and machine resources to obtain the same result with shorter time.

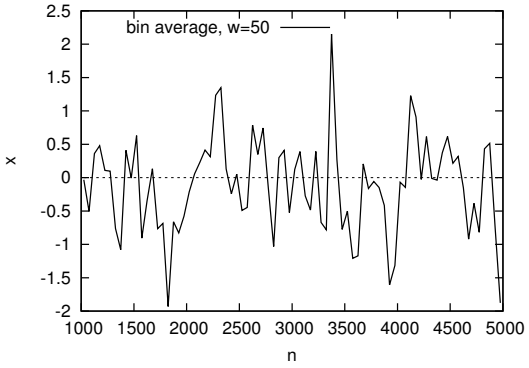

Figure 7: Bin-averaged version of Fig. 5, with bigger window for n, with $w = 50$.

step size c	acceptance	$c \times$ acceptance
0.5	0.9077	0.454
1.0	0.8098	0.810
2.0	0.6281	1.256
3.0	0.4864	1.459
4.0	0.3911	1.564
6.0	0.2643	1.586
8.0	0.1993	1.594

Table 1: Step size vs acceptance rate, total 10000 samples.

rate and the autocorrelation length scale as $1/c$ and c, respectively. On the other hand, when c is too small, the configurations are almost always updated, but only tiny amount. This is just a random walk with step size c, and hence the average change after n steps is $c\sqrt{n}$. Therefore the autocorrelation length should scale as $n \sim 1/c^2$. We expect the autocorrelation is minimized between these two regions. In Table 1, we have listed the acceptance rate for several values of c. We can see that the large-c scaling sets in at around $c = 2 \sim c = 4$. In Fig. 8 we have shown how $\langle x^2 \rangle$ converges to 1 as the number of configurations increases. We can actually see that $c = 2$ and $c = 4$ show faster convergence compared to too small or too large c.

1.3 How to calculate partition function

In MCMC, we cannot directly calculate the partition function Z; we can only see the expectation values. Usually the partition function is merely an normalization parameter, so we do not care. But sometimes it has interesting physical meanings; for example it can be used to test the conjectured dualities between supersymmetric theories.

Suppose you want to calculate $Z = \int dx e^{-S(x)}$, where $S(x)$ is much more complicated

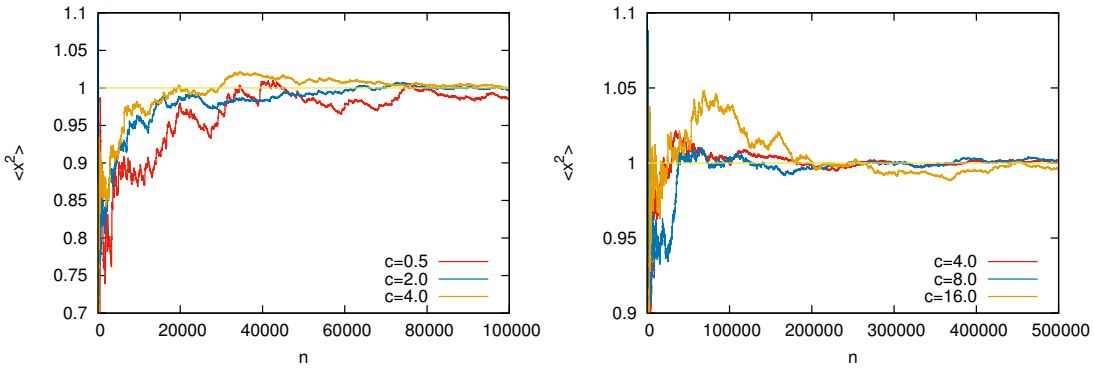

Figure 8: $\langle x^2 \rangle = \frac{1}{n} \sum_{k=1}^{n} \left(x^{(k)} \right)^2$ for several different step sizes c.

than $S_0(x) = x^2/2$. By using MCMC, we can calculate the ratio between Z and $Z_0 = \int dx e^{-S_0(x)} = \sqrt{2\pi}$,

$$\frac{Z}{Z_0} = \frac{1}{Z_0} \int dx e^{-S_0} \cdot e^{S_0 - S} = \left\langle e^{S_0 - S} \right\rangle_0 , \tag{10}$$

where $\langle \cdot \rangle_0$ stands for the expectation value with respect to the action S_0. Because we know Z_0 analytically, we can determine Z.

1.3.1 Overlapping problem and its cure

The method described above can always work *in principle*. In practice, however, it fails when the probability distributions $\rho(x) = \frac{e^{-S(x)}}{Z}$ and $\rho_0(x) = \frac{e^{-S_0(x)}}{Z_0}$ do not have sufficiently large overlap. As a simple example, let us consider $S = (x-c)^2/2$ (though you can analytically handle it!). Then $\rho(x)$ and $\rho_0(x)$ have peaks around $x = c$ and $x = 0$, respectively. When c is very large, say $c = 100$, the value of $e^{S_0 - S}$ appearing in the simulation is almost always an extremely small number $\sim e^{-5000}$, and once every e^{+5000} steps or so we get an extremely large number $\sim e^{+5000}$. And they average to $\frac{Z}{Z_0} = 1$. Clearly, we cannot get an accurate number if we truncate the sum at a realistic number of configurations. It happens because of the absence of the overlap of $\rho(x)$ and $\rho_0(x)$, or equivalently, because important configurations in two different theories are different; hence the 'operator' $e^{S_0 - S}$ behaves badly at the tail of $\rho_0(x)$. This is so-called *overlapping problem*.[4]

In the current situation, the overlapping problem can easily be solved as follows. Let us introduce a series of actions S_0, S_1, S_2, ..., $S_k = S$. We choose them so that S_i and S_{i+1} are sufficiently close and the ratio $\frac{Z_{i+1}}{Z_i}$, where $Z_i = \int dx e^{-S_i(x)}$, can be calculated without the overlapping problem. For example we can take $S_i = \frac{1}{2} \left(x - \frac{i}{k} c \right)^2$ with $\frac{c}{k} \sim 1$. Then we

[4] In SYM, the overlapping problem can appear combined with the *sign problem*; we will revisit this point in Sec. **??**.

can obtain $Z = Z_k$ by calculating $\frac{Z_1}{Z_0}$, $\frac{Z_2}{Z_1}$, $\cdots$, $\frac{Z_k}{Z_{k-1}}$. The same method can be applied to any complicated $S(x)$, as long as $e^{-S(x)}$ is real and positive.

This rather primitive method is actually powerful; for example the partition function of ABJM theory at finite coupling and finite N has been calculated accurately by using this method [2].

1.4 Common mistakes

1.4.1 Don't change step size during the run

Imagine the probability distribution you want to study has a bottleneck like in Fig. 9. For example if $S(x) = -\log\left(e^{-\frac{x^2}{2}} + e^{-\frac{(x-100)^2}{2}}\right)$ then $e^{-S(x)}$ is strongly suppressed between two peaks at $x = 0$ and $x = 100$. By using a small step size $c \sim 1$ you can sample one of the peaks efficiently, but then the other peak cannot be sampled. Then in order to go across the bottleneck you would be tempted to change the step size c when you come close to the bottleneck. You would want to make the step size larger so that you can jump over the bottle neck, or you would want to make the step size smaller so that you can slowly penetrate into the bottle neck. But if you do so, you obtain a wrong result, because the transition probability can depend on the past history. *You must not change the step size during the simulation.*

But it does not mean that you cannot use multiple fixed step sizes; it is allowed to change the step size if the conditions listed in the review are not violated. For example we can take $c = 1$ for even steps and $c = 100$ for odd steps; see Fig. 10.

Or we can throw a dice, namely randomly choose step size $c = 1, 2, 3, 4, 5, 6$ with probability 1/6. As long as the conditions listed in Sec. **??**, in particular the detailed balance, are not violated, you can do whatever you want.

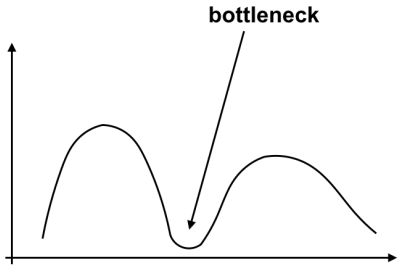

Figure 9: If the probability distribution has a bottleneck, the acceptance rate goes down there.

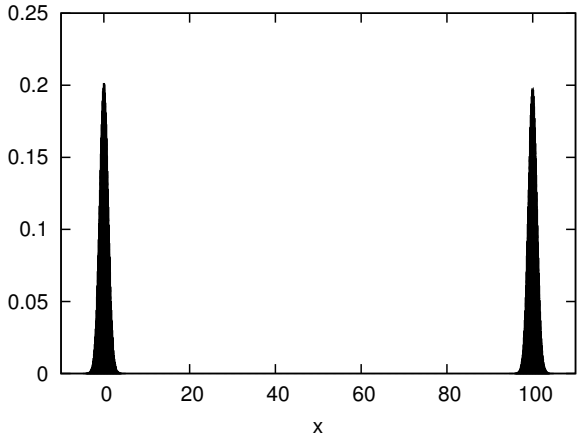

Figure 10: A histogram for $S(x) = -\log\left(e^{-\frac{x^2}{2}} + e^{-\frac{(x-100)^2}{2}}\right)$ with Metropolis, step size $c = 1$ for odd steps and $c = 100$ for even steps, 10^7 samples. The solid curve (which is actually invisible because it agrees with the histogram too precisely...) is the exact answer, $\frac{e^{-\frac{x^2}{2}}+e^{-\frac{(x-100)^2}{2}}}{2\sqrt{2}}$.

Similar temptation of evil is common in muilti-variable case. For example in the lattice gauge theory simulation it often happens that the acceptance is extremely low until the system thermalizes. Then we can use smaller step size just to make the system thermalize, and then start actual data-taking with a larger step size.[5] Or it occasionally happens that the simulation is trapped at a rare configuration so that the acceptance rate becomes almost zero. In such case, it would be useful to use multiple step sizes (the ordinary and very small).

1.4.2 Don't mix independent simulations with different step sizes

This is similar to Sec. 1.4.1: *when you have several independent runs with different step sizes, you must not mix them to evaluate the expectation value*, unless you pay extra cares for the error analysis. Although each stream are guaranteed to converge to the same distribution, if you truncate them at finite number of configurations each stream contains different uncontrollable systematic error.

Note however that, if you can estimate the autocorrelation time of each stream reliably, you can mix different streams with proper weights, with a careful error analysis.

[5] Another common strategy to reach the thermalization is to turn off the Metropolis test.

1.4.3 Make sure that random numbers are really random

In actual simulations, random numbers are not really random, they are just pseudo-random. But we have to make sure that they are sufficiently random. In Fig.11, we used the same sequence of random numbers repeatedly every 1000 steps. The answer is clearly wrong.

This mistake is very common;[6] when one simulates a large system, it will take days or months, so one has to split the simulation to small number of steps. Then when one submits a new job by mistake one would use the same 'random numbers' again, for example by reseting the seed of random numbers to the same number.

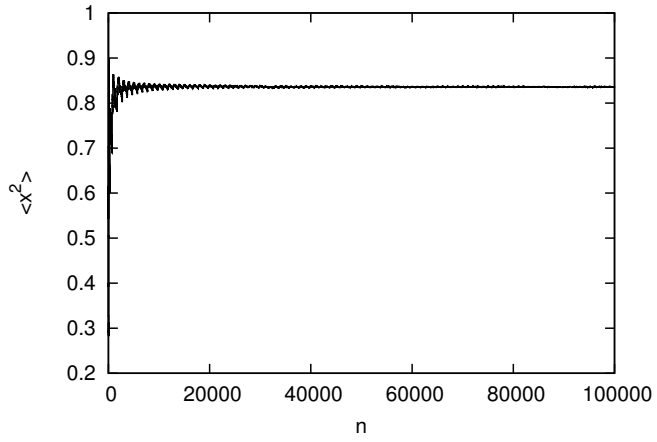

Figure 11: Gaussian integral with Metropolis, $c = 1$, with *non-random* numbers; we chose the same random number sequence every 1000 steps. $\langle x^2 \rangle = \frac{1}{n} \sum_{k=1}^{n} \left(x^{(k)} \right)^2$ converges to a wrong number, which is different from 1.

1.4.4 Don't use Mathematica

You don't need anything more than C or Fortran. It is true that Mathematica can do anything C or Fortran can do, but Mathematica is too slow. Don't use Mathematica even if you are the most patient person in world.

1.5 Sign problem

So far we have assumed $e^{-S(x)} \geq 0$. This was necessary because we interpreted $e^{-S(x)}$ as a 'probability'. But in physics we often encounter $e^{-S(x)} < 0$, or sometimes $e^{-S(x)}$ can be complex. Then a naive MCMC approach does not work. This is infamous *sign problem* (or *phase problem*, when $e^{-S(x)}$ is complex). Although no generic solution of the sign problem is known, there are various theory-specific solutions.

[6] Yes, I did.

References

[1] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of state calculations by fast computing machines. The journal of chemical physics, 21(6), 1087-1092.

[2] M. Hanada, M. Honda, Y. Honma, J. Nishimura, S. Shiba and Y. Yoshida, JHEP **1205**, 121 (2012) doi:10.1007/JHEP05(2012)121 [arXiv:1202.5300 [hep-th]].