Monte Carlo Simulation of 1D Harmonic Oscillator

Liuming Liu

Institute of Modern Physics, Chinese Academy of Sciences, Lanzhou 736000, China

CONTENTS

I. INTRODUCTION

The expectation value of a physical observable O in the Euclidean quantum field theory is formally given by

$$< O >= \frac{\int \mathcal{D}x\, O(x)\, e^{-S(x)/\hbar}}{Z}, \tag{1}$$

where $Z = \int \mathcal{D}x\, e^{-S(x)/\hbar}$, $\mathcal{D}x$ represents the integral over all paths. Generally this expression can not be evaluated analytically except for very simple physical problems such as 1D harmonic oscillator. In this lecture we are not aimed at solving 1D harmonic oscillator analytically. Instead, we introduce the Monte Carlo simulation on lattice, which is the basis of the numerical methods used in lattice QCD. As a pedagogical exercise, we implement the Monte Carlo method to solve 1D harmonic oscillator.

The expectation value of an observable in Eq. 1 can be approximated by the average of the observable evaluated on N sample configurations $\{x^{(0)}, x^{(1)}, \cdots, x^{(N-1)}\}$, distributed with probability $\propto e^{-S(x^{(n)})/\hbar}$. The average

$$< O > \approx \frac{1}{N} \sum_{n=0}^{N-1} O(x^{(n)}) \tag{2}$$

should be computed for sufficiently many configurations. In section II , we explain the Monte Carlo method that generates the configurations with desired distribution probability. In section III, we study 1D harmonic oscillator using the Monte Carlo method and give the numerical results of the ground state as well as the first excited state energy.

II. THE MONTE CARLO METHOD

A. Importance sampling

In the path integral, depending on the action $S(x)$ it will give different importance to different configurations. Therefore, when summing over all the configurations, it is more important to consider the configurations with larger weight factor $e^{-S(x^{(n)})/\hbar}$ than those with smaller weight. That is to say, the configurations with smaller action are more important than those with larger action. The idea of importance sampling is to approximate the huge sum by a comparatively small subset of configurations distributed according to the weight factor $e^{-S(x^{(n)})/\hbar}$.

B. Markov chains

How do we find a set of configurations $\{x^{(0)}, x^{(1)}, \cdots, x^{(N-1)}\}$ following the distribution proportional to the weight factor $e^{-S(x^{(n)})/\hbar}$? The idea is to start from an arbitrary configuration and then construct a stochastic sequence of configurations that eventually follows the equilibrium distribution $P(x) \propto e^{-S(x^{(n)})/\hbar}$. This is the so-called Markov chain

$$x^{(0)} \to x^{(1)} \to x^{(2)} \to \cdots\cdots . \tag{3}$$

In this chain, each configuration $x^{(n)}$ is generated subsequently from the previous one $x^{(n-1)}$. A Markov chain is characterized by a transition probability $T(x'|x)$ which is the probability to get x' starting from x. This probability only depends on x' and x. $T(x'|x)$ should obey the relations

$$0 \leq T(x'|x) \leq 1, \qquad \sum_{x'} T(x'|x) = 1. \tag{4}$$

The first relation delimits the range of a probability. The second relation imply that the total probability to jump from a configuration x to any target configuration x' is equal to 1.

When a Markov chain is evolved into equilibrium, the probability to hop into a configuration x' should be equal to the probability to hop out of this configuration. The balance relation reads

$$\sum_{x} T(x'|x)P(x) = \sum_{x} T(x|x')P(x'). \tag{5}$$

The left hand side is the sum of transition probability $T(x'|x)$ leading into x' from all possible starting configuration x, weighted by the probability $P(x)$ that the system is in the configuration x. The right hand side gives the total probability to hop out from x' into all possible final configurations x. The probability $P(x)$ is the equilibrium distribution we want to obtain. The relation in Eq. 5 is called *balance condition*. One can prove that, starting from an arbitrary configuration x_0, the equilibrium distribution $P(x)$ can be eventually achieved by iteratively applying the transition probability $T(x'|x)$ which satisfies the balance condition. In addition, in order to obtain correct results, the Markov chain must be able to access all configurations. If the transition probability $T(x'|x)$ is strictly positive for all pairs of x and x', then every configuration can be eventually reached.

In actual calculations, we start to measure observables after a sufficient number of updates, when we can assume the considered configurations are in equilibrium distribution. To reduce the effect of the autocorrelation between the configurations , we usually skip certain number of configurations before using one to make measurement.

C. Metropolis algorithm

A sufficient condition for a solution of the balance equation Eq. 5 is to require the equality holds term wise:

$$T(x'|x)P(x) = T(x|x')P(x'). \tag{6}$$

This condition is called *detailed balance condition*. The detailed balance condition is a sufficient but not necessary condition for the Markov chain to approach equilibrium. Most Monte Carlo algorithms use the detailed balance condition. The most famous one is the *Metropolis algorithm*, which we will explain in the following.

Starting from an arbitrary configuration x, the Metropolis algorithm advances the Markov chain in the following steps:

1. Choose some candidate configuration x' as a tentative update according to some selection probability $T_0(x'|x)$. $T_0(x'|x)$ is symmetric in x and x', i.e. $T_0(x'|x) = T_0(x|x')$.

2. Evaluate the tentative new action $S(x')$. If the action is lowered comparing to its old value $S(x)$, then x' is accepted as the new configuration. Otherwise, accept it with conditional probability $e^{-(S(x')-S(x))/\hbar}$.

3. Repeat these steps from the beginning.

It is easy to show that the Metropolis algorithm satisfies the detailed balance condition.

III. APPLYING THE MONTE CARLO METHOD TO 1D HARMONIC OSCILLATOR

In this section, we present the detailed processes to solve 1D harmonic oscillator using Monte Carlo simulations on the lattice.

Introducing a time-lattice with lattice spacing $a = \frac{T}{N}$, where T is the total length in time direction, N is a large integer number. See Fig. III for the illustration. Let x_n denotes the position of the oscillator at the n'th time slice, the discretized action of 1D harmonic oscillator becomes

$$S(x) = \sum_{n=0}^{N-1} \left(\frac{m(x_{n+1} - x_n)}{2a} + \frac{am\omega^2(x_{n+1}^2 + x_n^2)}{4}\right). \tag{7}$$

With periodic bound condition, $x_0 = x_N$.

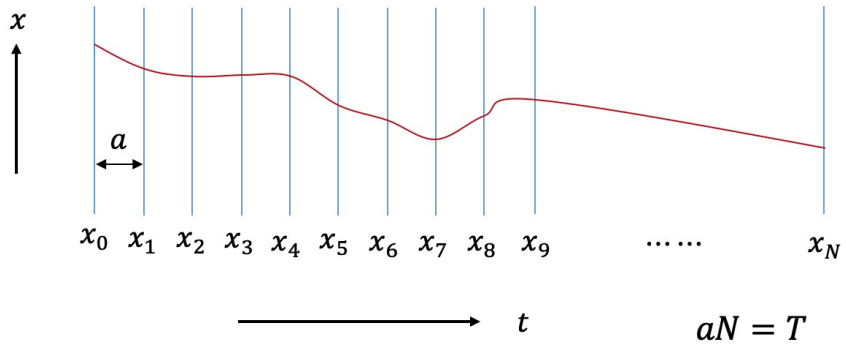

FIG. 1. Illustration of 1D harmonic oscillator on a time lattice.

A. Generating configurations

For the convenience of numerical calculations, we define a dimensionless position variable $u = x/b$, where $b = \sqrt{\hbar/m\omega}$ is the typical length scale of a harmonic oscillator. Rewrite the action with u:

$$\frac{S(u)}{\hbar} = \sum_{n=0}^{N-1} \left(\frac{(u_{n+1} - u_n)^2}{2a\omega} + \frac{a\omega(u_{n+1}^2 + u_n^2)}{4}\right). \tag{8}$$

The classical period of the harmonic oscillator is $2\pi/\omega$. In a practical simulation, the lattice spacing a should be much smaller than the period, while the total length T should be much larger than the period. That is $a \ll 2\pi/\omega \ll T$. For simplicity, we will set $\omega = 1$ in the rest of the lecture. Keeping in mind that the range of a and N should satisfy $a \ll 2\pi \ll aN$, we proceed to generate configurations using the Metropolis algorithm.

1. Choose an arbitrary starting configuration $u = \{u_0, u_1, u_2, \cdots, u_{N-1}\}$. One can choose $u_n = 0$ for all n, which is called a cold start. One can also choose u_n to be some random numbers, which is called a hot start.

2. Choose a candidate configuration $u'_n = u_n + \delta u \times r$, where r is a random number uniformly distributed between [-1,1], δx is a parameter we choose empirically.

3. If the action $S(u')$ is smaller than $S(u)$, u' is accepted as the new configuration. Otherwise it is accepted with conditional probability $e^{-(S(u')-S(u))/\hbar}$. If u' is rejected, u is considered again in the Markov chain.

4. Repeat from step 2.

The input parameters for the simulations are:

- a: lattice spacing. The value of a should be much smaller than 2π.

- N or T: the total length of time. The value of aN or T should be much larger than 2π.

- δu: the size of the step that the algorithm changes the values of the position u. This parameter affects the acceptance rate.

- N_{conf}: the total number of configurations generated in the algorithm.

- N_{dump}: The first N_{dump} configurations will be dumped. This means that the Markov chain reaches equilibrium after N_{dump} configurations.

- N_{skip}: Number of configurations to be skipped before using one to make measurements. This number should be sufficiently large to avoid autocorrelation.

Note: The total number of configurations used for measurements is $(N_{conf} - N_{dump})/N_{skip}$, and only these configurations need to be written to disk.

An example code to generate the configurations is given below.

```
/* compute the weight factor that involves three points x0, x1, x2. When updating x1, the change of
         the action only involves the neighbouring points x0 and x2.  */
double weight(double x0, double x1, double x2, double a){
    double S=0.0;
    S=(x1-x0)*(x1-x0)/(2.0*a) + a*(x0*x0+x1*x1)/4.0 + (x2-x1)*(x2-x1)/(2.0*a) + a*(x2*x2+x1*x1)/4.0;
    return exp(-1.0*S);
}

int main(){
    int Nt,Nconf;
    double a; // lattice spacing
    double delta;  //step size
    char output_path[200]; // output file path

// read in input
    double* x1 = new double[Nt];
    double* x2 = new double[Nt];
    double* x = new double[Nt*Nconf];

//generate random number
        std::default_random_engine seed;
        seed.seed(time(NULL));
        std::uniform_real_distribution<double> ran1(-1.0,1.0); // random number between [-1.0,1.0]
        std::uniform_real_distribution<double> ran2(0,1.0); // random number between [0,1.0]

//initialize the starting configuration to be 0.0
    for(int i=0; i<Nt; i++)
        *(x1+i) = *(x2+i) = 0.0;  // cold start

// update the configuration with metropolis algorithm
    std::vector<int> t_iter;
    for(int i=0; i<Nt; i++) t_iter.push_back(i);
    for(int n=0; n<Nconf; n++){
        std::random_shuffle (t_iter.begin(), t_iter.end()); // shuffle the order of updating
        for(int t=0; t<Nt; t++){
            int i=t_iter[t];
// chose a candidate configuration x2 by updating the ith points of x2
            *(x2+i) = *(x1+i) + delta*ran1(seed);
            if(ran2(seed) < weight(*(x2+(i-1+Nt)%Nt),*(x2+i),*(x2+(i+1)%Nt),aomega)/
                                    weight(*(x1+(i-1+Nt)%Nt),*(x1+i),*(x1+(i+1)%Nt), aomega))
                *(x1+i)=*(x2+i);  // accept the candidate
            else
                *(x2+i)=*(x1+i);  // reject the candidate
        }

        for(int i=0; i<Nt; i++)
            *(x + n*Nt + i) = *(x2 + i);
    }

// write out the configurations x to outfile
}
```

B. Measuring the ground state energy

With the configurations generated in the last section, it is straightforward to compute the expectation value of u^2:

$$< u^2 >= \frac{1}{M} \sum_{m=0}^{M} u^2(m). \tag{9}$$

where M is the number of configurations, $u^2(m)$ is the averaged value of u^2 on all time slices of the configuration $u^{(m)}$. The error of $< u^2 >$ is given by the standard error formula

$$\sigma_{u^2} = \sqrt{\frac{\sum_m (u^2(m) - < u^2 >)^2}{M(M-1)}}. \tag{10}$$

The exact value of $< u^2 >$ can be calculated by the formula [1]

$$< u^2 >_{theory} = \frac{1}{2\sqrt{1+\frac{a^2}{4}}} \frac{1+R^N}{1-R^N}, \qquad R = 1 + \frac{a^2}{4} - a\sqrt{1+\frac{a^2}{4}}. \tag{11}$$

In figure III B, we present the numerical results and the theory values of $< u^2 >$. The blue points with errorbars are the numerical results and the red points are the theory values. The read horizontal line represents the theory value at the continuum and infinite size limit $a \to 0$ and $N \to \infty$, which is 0.5. In the left panel, we compare the results at different lattice spacing, the number of lattice points N is fixed to be 500. One can see that at small values of lattice spacing, the numerical results and the theory values have large discrepancy. The reason is the that the total length T is not large enough. At large lattice spacing, the numerical results also deviate from the continuum theory value 0.5. In the right panel, we comparing the results for different N values, while the lattice spacing is fixed to be 0.01. It is clear that N must be large enough to obtain correct answer.

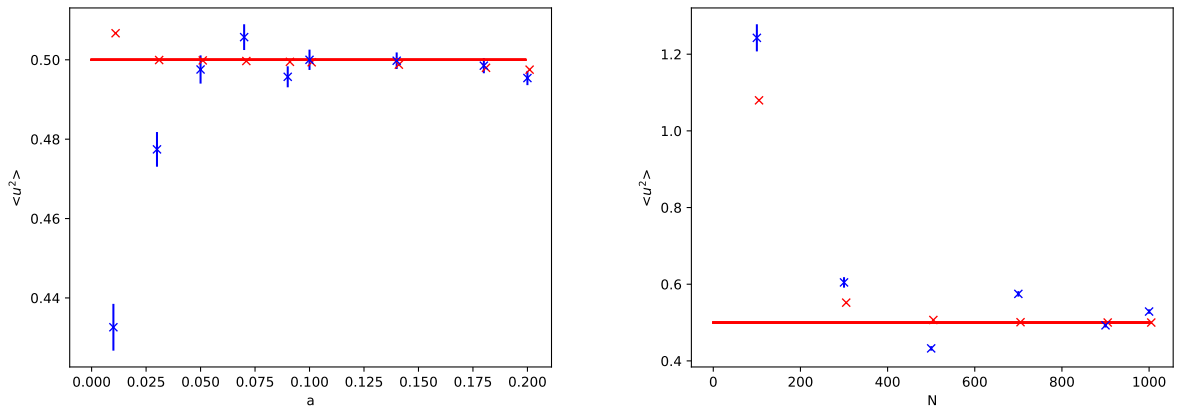

FIG. 2. The expectation value of u^2. The blue points with errorbars are the numerical results and the red points are the theory values. In the left panel, we compare the results at different lattice spacing, while the number of lattice points N is fixed to be 500. In the right panel, we comparing the results for different N values, while the lattice spacing is fixed to be 0.01.

C. Measuring the excited state energy

In order to get the energies of the excited states, we compute the correlation function

$$\begin{aligned} C(t) =< x(t)x(0) > &= \frac{1}{Z}\int \mathcal{D}x x(t)x(0)e^{-S(x)} \\ &= \frac{Tr(e^{-(T-t)\hat{H}}\hat{x}e^{-t\hat{H}}\hat{x})}{Tr(e^{-T\hat{H}})} \\ &= \frac{\sum_m < m|e^{-(T-t)\hat{H}}\hat{x}e^{-t\hat{H}}\hat{x}|m >}{\sum_n < n|e^{-T\hat{H}}|n >} \\ &= \frac{|< m|\hat{x}|n >|^2 e^{-t\Delta E_n}e^{-(T-t)\Delta E_m}}{1+e^{-T\Delta E_1}+e^{-T\Delta E_2}+\cdots} \\ &\xrightarrow{T\to\infty} e^{-\Delta E_1 t}. \end{aligned} \tag{12}$$

Here ΔE_n is the energy difference of n'th excited state with the ground state, this energy difference is what can be measured in experiments. The energy of the first excited energy can be obtained by fit the correlation function to an exponential function at large t range. In Fig. III C we plot the correlation function as a function of t. The correlation function is computed with $a = 0.1$, $N = 500$. When we calculate the correlation function, we do not only average over configurations, but also over time slices. So what we actually calculate is

$$< x(t)x(0) >= \frac{1}{M}\sum_m (\frac{1}{N}\sum_{t_0=0}^{N-1} x(t+t_0)^{(m)}x(t_0)^{(m)}), \tag{13}$$

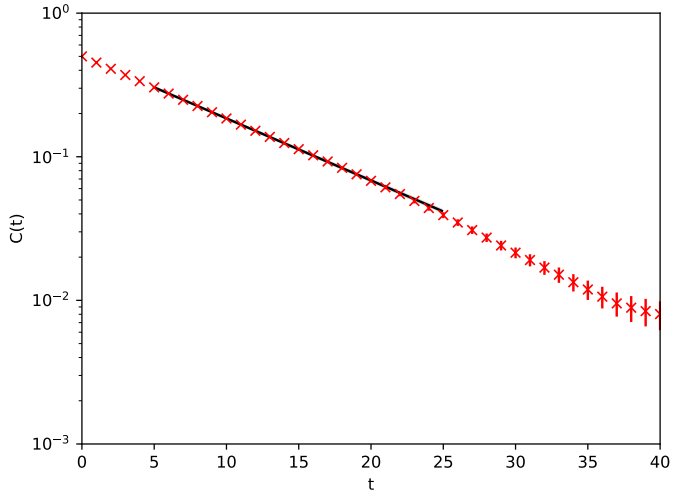

FIG. 3. Correlation function as a function of t. The black line is the exponential fit to the correlation function.

where M is number of configurations.

The fit to the correlation function is performed by minimum χ^2 method. The fit function $f(t;\Delta E) = e^{-t\Delta E}$. χ^2 is defined as

$$\chi^2 = \sum_{i,j}(f(t_i,\Delta E) - C(t_i))C_{ij}^{-1}(f(t_j,\Delta E) - C(t_j)), \tag{14}$$

where C_{ij} is the covariance matrix $C_{ij} = \sum_m \frac{(C(t_i,m)-<C(t_i)>)(C(t_j,m)-<C(t_j)>)}{M(M-1)}$. One minimize the χ^2 to obtain the value of ΔE. The error of ΔE can be estimated by Jackknife or Bootstrap method. See Ref. [2] for more details about Jackknife and Bootstrap. The fitted value of ΔE in Fig. III C is 0.995 ± 0.009, which agrees well with the theory value 1.

[1] M. Creutz and B. Freedman, Annals Phys. **132**, 427 (1981).

[2] C. Gattringer and C. B. Lang, Lect. Notes Phys. **788**, 1 (2010).