Physics 514 – Monte Carlo Integration

Emanuel Gull

October 6, 2016

Chapter 1

Integration

In thermodynamics, quantum mechanics, statistical mechanics, as well as in many other fields of physics, often very high dimensional integrals have to be evaluated. Even in a classical N-body simulation the phase space has $6N$ dimensions, as there are coordinates for position and momentum of each particle. In a quantum problem of N particles the phase space dimension is even exponential in the number of particles. We therefore need an efficient technique for evaluating very high dimensional integrals.

1.1 Standard quadrature methods

A Riemann integral of a function f in the interval $[a, b]$ can be approximated by a finite sum

$$\int_a^b f(x)dx = \sum_{i=1}^{N} \left(f(a + i\Delta x)\Delta x + O(\Delta x^2) \right) \qquad (1.1)$$

where $\Delta x = (b-a)/N$. The discretization error decreases as $1/N$ for this 'rectangular' rule. A better approxima-

Also worth noting: Rhomberg quadrature, used for smooth functions
Trapezoidal rule multiple times, converges quickly

tion is the trapezoidal rule:

$$\int_a^b f(x)dx = \Delta x \left[\frac{1}{2}f(a) + \sum_{i=1}^{N} f(a+i\Delta x) + \frac{1}{2}f(b)\right] + O(\Delta x^2) \tag{1.2}$$

or the Simpson rule

$$\int_a^b f(x)dx \tag{1.3}$$
$$= \frac{\Delta x}{3}\left[f(a) + \sum_{i=1}^{N/2} 4f(a+(2i-1)\Delta x) + \sum_{i=1}^{N/2} 2f(a+2i\Delta x) + f(b)\right] +$$

(for even N). In practice, more elaborate schemes are used depending on the properties of the functions: *Romberg* quadrature, *Gaussian* quadrature, or one of many *adaptive quadrature* schemes. Numerical mathematics textbooks, e.g. *Introduction to numerical analysis* by J. Stoer and R. Bulirsch, have more information on these methods[1].

While standard integration methods work well for functions of few variables, the convergence in high-dimensional spaces is very slow: with N points in d dimensions, the linear distance L, $L^d = N$, between two points scales only as $N^{-1/d}$. Thus the Simpson rule in d dimensions converges only as $N^{-4/d}$, which is too slow to be practical for large d.

[1] J. Stoer and R. Bulirsch, Introduction to numerical analysis, Springer, New York (1980), ISBN 0-387-90420-4

This is Gull's go-to reference book for numerical methods

1.2 Monte Carlo integrators

The solution to this problem are Monte Carlo integrators, for which the convergence is independent on the number of dimensions. With randomly chosen points the convergence does not depend on dimensionality. Using N randomly chosen poinst $\mathbf{x}_i$ the integral can be approximated by

$$\frac{1}{\Omega}\int f(\mathbf{x})d\mathbf{x} \approx \overline{f} := \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_i) \tag{1.4}$$

where $\Omega = \int dx$ is the integration volume. The error of a Monte Carlo estimate is

$$\Delta = \sqrt{\frac{\mathrm{Var} f}{N}} \sim \sqrt{\frac{\overline{f^2} - \overline{f}^2}{N-1}} \tag{1.5}$$

where the variance $\mathrm{Var}(f)$ is defined as

$$\mathrm{Var}(f) = \frac{1}{\Omega}\int f(\mathbf{x})^2 d\mathbf{x} - \left[\frac{1}{\Omega}\int f(\mathbf{x})d\mathbf{x}\right]^2 \sim \frac{N}{N-1}(\overline{f^2} - \overline{f}^2) \tag{1.6}$$

Thus the error scales as $N^{-1/2}$, with a prefactor given by the variance. In $d \geq 9$ dimensions Monte Carlo methods are preferable to the Simpson rule.

1.2.1 Importance Sampling

Simple Monte Carlo integration is is often not a suitable integration method. The reason is that integrals of interest, e.g. phase space integrals as they appear in statistical mechanics, are often strongly peaked in a small

region of phase space and therefore have a large variance, slowing down the convergence with to a large prefactor in front of the $\sqrt{N}$. A solution to this problem will need to find a way to reduce the variance of the sampling procedure. One possible way is "importance sampling": the points $\mathbf{x}_i$ are not chosen from a uniform distribution, but according to a distribution $p(\mathbf{x})$ with

$$\int p(\mathbf{x}) d\mathbf{x} = 1 \tag{1.7}$$

Using these p-distributed random points the sampling is done according to:

$$\langle f \rangle = \frac{1}{\Omega} \int f(\mathbf{x}) d\mathbf{x} = \frac{1}{\Omega} \int \frac{f(\mathbf{x})}{p(\mathbf{x})} p(\mathbf{x}) d\mathbf{x} \approx \frac{1}{N} \sum_{i=1}^{N} \frac{f(\mathbf{x}_i)}{p(\mathbf{x}_i)} \tag{1.8}$$

and the error is

$$\Delta = \sqrt{\frac{\mathrm{Var}(f/p)}{N}} \tag{1.9}$$

It is ideal to choose the distribution function p as closely to f as possible. Then the ratio f/p is nearly constant and the variance small.

As an example, it is much more efficient to compute $\int_0^1 f(x) = \int_0^1 \exp(-x^2)$ using exponentially distributed random numbers with $p(x) = \exp(-\lambda x)$ instead of uniformly distributed random numbers (try it!).

1.3 Pseudo random numbers

The most important ingredient for a Monte Carlo calculation is a source of random numbers. The problem is:

thermal noise on your chip, Fermi's hat, quantum noise, listening to photons with telescope… lots of interesting ways to get "random" numbers

how can a deterministic computer calculate true random numbers?

Since going to a casino is not useful in practice, and truly random data available from chip noise or atmospheric noise is too slow, pseudo random number generators have to be used. Despite being deterministic these pseudo random number generators can produce sequences of numbers that look random if one does not know the underlying algorithm. As long as they are sufficiently random ('sufficiently' being defined by the problem being investigated), the pseudo random numbers can be used instead of true random numbers.

1.3.1 Uniformly distributed random numbers

A popular type of random number generator produsing uniformly distributed numbers is the linear congruential generator, or LCG:

$$x_n = (ax_{n-1} + c) \mod m, \tag{1.10}$$

with poisitive integer numbers a, x, and m. The quality of the pseudo random numbers depends sensitively on the choice of these parameters. A common and good choice is $a = 16807$, $c = 0$, and $m = 2^{31} - 1$ with $x_0 = 667790$. The main problem of LCG generators is that, because the next number x_{n+1} depends only on one previous number x_n, the sequence of numbers produced is at most m. Current computers will exhaust this sequence in seconds. LCG generators should thus no longer be used.

The popular unix system random number generator, `drand48`, is a linear congruential generator.

Many modern generators are based on lagged Fibonacci methods, such as the generator

$$x_n = x_{n-607} + x_{n-253} \mod m \tag{1.11}$$

The first 607 numbers need to be produced by another generator, e.g. an LCG generator. Instead of the shifts $(607, 253)$ other good choices are $(2281, 1251)$, $(9689, 5502)$, or $(44497, 23463)$. By calculating the next number from more than one previous number these generators can have extremely long periods.

One of the more recent generators is the Mersenne Twister, a combination of a lagged Fibonacci generator with a bit twister, shuffling around the bits of the random numbers to improve the quality of the generator. The python numpy generator `numpy.random` is based on a Mersenne twister.

Instead of coding a pseudo random number generator yourself, use one of the libraries (`numpy.random` on python, `boost` random on C++), or if you need high performance implementations use one of the vendor libraries that provide very fast pseudo random number generators: `MKL`, `ACML`, or on IBM machine the `essl` library.

1.3.2 Testing pseudo random numbers

Before using a pseudo random number gnerator you will have to test the generator to determine whether the numbers are sufficiently random for your application. Standard tests that are applied to all new generators include:

- The period has to be longer than the number of

pseudo random numbers required for your simulation.

- The numbers have to be uniformly distributed. This can be tested by a statistical test, e.g. a χ^2 test or a Kolmogorov-Smirnov test.

- Successive numbers should not be correlated. This can again be tested by a χ^2 test. It can also be tested by a simple graphical test: fill a square with successive points at the coordinates (x_{2i-1}, x_{2i}) and look for patterns.

All of these tests, and a large number of similar tests are necessary but by no means sufficient. Landau, Ferrenberg and Wong have demonstrated that one of the standard and well-tested generators gave very wrong results when applied to the simulation of an Ising model. The reason were long-range correlations in the generators that had not been picked up by any test. The consequence is that no matter how many tests you run, you can never be sure about the quality of the pseudo random number generator. The only reliable test is to rerun the simulation with another random number generator and test whether the result changes or not.

1.3.3 Non-uniformly distributed random numbers

Non-uniformly distributed random numbers can be obtained from uniformly distributed random numbers in the following way. Consider the probability that a random number y, distributed with a distribution function

f, is less than x. This probability is just the integral

$$P_f(y < x) = \int_{-\infty}^{x} f(y)dy =: F(x). \tag{1.12}$$

This is also the probability that a uniformly distributed random number u is less than $F(x)$

$$P_f(y < x) = F(x) = P_u(u < F(x)) \tag{1.13}$$

If the integrated probability distribution function F can easily be inverted, one can obtain an f-distributed random number x from an uniformly distributed random number u through $x = F^{-1}(u)$.

This can be used to obtain exponentially distributed random numbers with a distribution $f(x) \propto \exp(-\lambda x)$ through

$$x_{\exp} = -\frac{1}{\lambda}\log(1-u) \tag{1.14}$$

The normal distribution cannot easily be inverted in one dimension. In two dimensions it can be inverted, leading to the Box-Muller method in which two normally distributed numbers n_1 and n_2 are generated from two uniformly distributed numbers u_1 and u_2:

$$n_1 = \sqrt{-2\log(1-u_1)}\cos(2\pi u_2) \tag{1.15}$$

$$n_2 = \sqrt{-2\log(1-u_1)}\sin(2\pi u_2) \tag{1.16}$$

For general but bounded distributions $f(x) \leq h$, with arbitrary $h < \infty$ defined on an interval $[a, b[$ the rejectance method can be used. One picks a uniformly distributed random number u in the interval $[a, b[$ and accepts it with probability $f(u)/h$. If the number is rejected, a new uniformly distributed random number u is chosen and tested.

1.4 Markov chains and the Metropolis algorithm

The methods for non-uniformly distributed random numbers discussed above are useful for simple distributions in low dimensions. In general the integrated probability distribution function F cannot be inverted, and the rejectance methods will almost never accept a uniformly drawn number. Then a Markov process can be used to create a sequence of pseudo-random numbers distributed with an arbitrary distribution p.

Starting from an initial point $\mathbf{x}_0$ a Markov chain of states is generated:

$$\mathbf{x}_0 \to \mathbf{x}_1 \to \mathbf{x}_2 \to \cdots \to \mathbf{x}_n \to \mathbf{x}_{n+1} \ldots \tag{1.17}$$

A transition matrix $W_{\mathbf{xy}}$ gives the transition probabilities of going from state $\mathbf{x}$ to state $\mathbf{y}$ in one step of the Markov process. As the sum of probabilities of going from state $\mathbf{x}$ to any other state is one, the columns of the matrix W are normalized:

$$\sum_{\mathbf{y}} W_{\mathbf{xy}} = 1 \tag{1.18}$$

A consequence is that the Markov process conserves the total probability. Another consequence is that the largest eigenvalue of the transition matrix W is 1 and the corresponding eigenvector with only positive entries is the equilibrium distribution. For typical vectors $\mathbf{x}$, successive application of the Markov steps will lead to that equilibrium distribution.

We want to determine the transition matrix W so that we asymptotically reach the desired probability $p_{\mathbf{x}}$ for a configuration $\mathbf{x}$. A set of sufficient conditions is:

- *Ergodicity*: It has to be possible to reach any configuration $\mathbf{x}$ from any other configuration $\mathbf{y}$ in a finite number of Markov steps. This means that for all $\mathbf{x}$ and $\mathbf{y}$ there exists a positive integer $n < \infty$ such that $(W^n)_{\mathbf{xy}} \neq 0$.

- *Detailed Balance*: The probability distribution $p_{\mathbf{x}}^{(n)}$, with n enumerating the steps, changes at each step of the Markov process:

$$\sum_{\mathbf{x}} p_{\mathbf{x}}^{(n)} W_{\mathbf{xy}} = p_{\mathbf{y}}^{(n+1)}. \tag{1.19}$$

  For long ties it converges to the equilibrium distribution $p_{\mathbf{x}}$. This equilibrium distribution $p_{\mathbf{x}}$ is an eigenvector with left eigenvalue 1 and the equilibrium condition

$$\sum_{\mathbf{x}} p_{\mathbf{x}} W_{\mathbf{xy}} = p_{\mathbf{y}} \tag{1.20}$$

  must be fulfilled. This equilibrium condition is also called 'balance' condition. Eq. 1.21 is fulfilled if *detailed balance* is fulfilled (i.e. detailed balance is sufficient, but not necessary, for Eq. 1.21):

$$p_{\mathbf{x}} W_{\mathbf{xy}} = p_{\mathbf{y}} W_{\mathbf{yx}}. \tag{1.21}$$

1.4.1 Metropolis algorithm

The *Metropolis algorithm* is one of the simplest Monte Carlo algorithms. Given are configurations $\mathbf{x}$ in some configuration space. We generate a Markov chain of configurations $\mathbf{x}_1 \to \mathbf{x}_2 \to \cdots \mathbf{x}_n \to \mathbf{x}_{n=1} \to$ and measure the observable A:

1. start with a point $\mathbf{x}_i$ and propose a random change $\Delta\mathbf{x}$ to $\mathbf{x}_i$, resulting in a proposed new configuration $\mathbf{x}' = \mathbf{x}_i + \Delta\mathbf{x}$.

2. Calculate the probability of the old state, $p(\mathbf{x}_i)$, and of the proposed state, $p(\mathbf{x}')$, and compute the probability ratio $P = p(\mathbf{x}')/p(\mathbf{x})$.

3. If $P > 1$ then choose $\mathbf{x}'$ as the new configuration x_{i+1}.

4. If $P < 1$ then choose $\mathbf{x}_i$ with probability P, *i.e.* choose the old configuration as the new configuration. Otherwise choose $\mathbf{x}'$ as the new configuration. We do that by drawing a random number r uniformly distrbuted in the interval $[0, 1[$ and set $\mathbf{x}_{i+1} = \mathbf{x}'$ if $r < P$.

5. measure the quantity A at the new point $\mathbf{x}_{i+1}$.

This algorithm is *ergodic* if it is possible to reach any configuration during the course of the simulation from any other point in configuration space.

The algorithm also fulfills detailed balance, if there is a step $-\Delta x$ for each step Δx, which corresponds to the inverse change: if the probability of proposing the step Δx is $\frac{1}{N}$, then

$$\frac{W_{\mathbf{xy}}}{W_{\mathbf{yx}}} = \frac{\frac{1}{N}\min(1, p(\mathbf{y})/p(\mathbf{x}))}{\frac{1}{n}\min(1, p(\mathbf{x})/p(\mathbf{y}))} = \frac{p(\mathbf{y})}{p(\mathbf{x})}. \tag{1.22}$$

Example: to integrate a one-dimensional function we take the limit $N \to \infty$ and pick any change $\delta \in [-\Delta, \Delta]$ with equal probability. The detailed balance equation is

then:

$$\frac{W_{\mathbf{xy}}}{W_{\mathbf{yx}}} = \frac{\frac{d\delta}{2\Delta}\min(1, p(\mathbf{y})/p(\mathbf{x}))}{\frac{d\delta}{2\Delta}\min(1, p(\mathbf{x})/p(\mathbf{y}))} = \frac{p(\mathbf{y})}{p(\mathbf{x})}. \tag{1.23}$$

1.5 Autocorrelations, Equilibration, and Monte Carlo error estimates

1.5.1 Autocorrelation effects

So far we were concerned with Monte Carlo errors of indepedendent samples. In the case of Markov chains, we have to take into account that there are correlations between successive points in the Markov chain: $\mathbf{x}_n$ is not independent from $\mathbf{x}_{n+1}$! These correlations between configurations manifest themselves in correlations between the samples of the measured observables. Denote the measurement of the observable A at Monte Carlo step t by $A_t = A(\mathbf{x}_t)$. The autocorrelations decay exponentially for large time differences Δ:

$$\langle A_t A_{t+\Delta}\rangle - \langle A\rangle^2 \propto \exp(-\Delta/\tau_A^{(\exp)}). \tag{1.24}$$

the quantity τ_A has dimensions of (Monte Carlo) time and is called the *exponential* auto-correlation time.

An alternative definition is the so-called integrated autocorrelation time $\tau_A^{(\mathrm{int})}$, defined by

$$\tau_A^{(\mathrm{int})} = \frac{\sum_{\Delta=1}^{\infty}(\langle A_t A_{t+\Delta}\rangle - \langle A\rangle^2)}{\langle A^2\rangle - \langle A\rangle^2} \tag{1.25}$$

The expectation value of the observable A can be estimated by the mean $\overline{A}$, using Eq. **??**. However, because of correlations between different observables, the error estimate $(\Delta A)^2$ has to be modified. The error estimate 1.5 is

the expectation value of the squared difference between sample average and expectation value:

$$(\Delta A)^2 = \langle(\overline{A} - \langle A\rangle)^2\rangle = \langle\left(\frac{1}{N}\sum_{t=1}^{N} A(t) - \langle A\rangle\right)^2\rangle \tag{1.26}$$

$$= \langle\frac{1}{N^2}\sum_{i=1}^{N}\Big(A(t)^2 - \langle A\rangle^2\Big)\rangle + \frac{2}{N^2}\sum_{t=1}^{N}\sum_{\Delta=1}^{N-t}\Big(\langle A(t)A(t+\Delta)\rangle - \langle\langle A\rangle^2\rangle\Big) \tag{1.27}$$

$$\approx \frac{1}{N}\mathrm{Var}A(1+2\tau_A^{(\mathrm{int})}) \tag{1.28}$$

$$\approx \frac{1}{N-1}\langle\overline{A^2} - \overline{A}^2\rangle(1+2\tau_A^{(\mathrm{int})}) \tag{1.29}$$

In going from the second to the third line we assumed that the autocorrelation time is much smaller than N and extended the summation over Δ to infinity. In the last line we replaced the variance by an estimate obtained from the sample. We see that the number of statistical uncorrelated samples is reduced from N to $N/(1+2\tau_A^{(\mathrm{int})})$.

In many Monte Carlo simulations the error analysis is unfortunately not done accurately. Proper error analysis will be one of the important points treated in the projects. You can find much more information in the excellent book by Werner Krauth[2].

1.5.2 The binning analysis

The integrated autocorrelation times are a priori unknown and have to be estimated. The binning analysis is

[2] Statistical Mechanics: Algorithms and Computations, Werner Krauth, Oxford University Press, 2006, ISBN 978-0198515364

Good reference for the math of all this

a reliable way to estimate the integrated autocorrelation times. Starting from the original series of measurements A_i with $i = 1, \cdots, N$ we iteratively create 'binned' series by averaging over consecutive entries:

$$A_i^{(l)} := \frac{1}{2}\left(A_{2i-1}^{(l-1)} + A_{2i}^{(l-1)}\right), i = 1, \ldots, N_l \equiv N/2^l \tag{1.30}$$

These bin averages $A_i^{(l)}$ are less correlated than the original values $A_i^{(0)}$. The mean value is still the same.

The errors $\Delta A^{(l)}$, estimated incorrectly using Eq. 1.5, are

$$\Delta A^{(l)} = \sqrt{\frac{\mathrm{Var} A^{(l)}}{N_l - 1}} \sim \frac{1}{N_l}\sqrt{\sum_{i=1}^{N_l}\left(A_i^{(l)} - \overline{A^{(l)}}\right)^2}. \tag{1.31}$$

These errors increase as a function of bin size 2^l. For $2^l \gg \tau_A^{(\mathrm{int})}$ the bins become uncorrelated and the errors converge to the correct error estimate:

$$\Delta A = \lim_{l\to\infty} \Delta A^{(l)}. \tag{1.32}$$

This binning analysis gives a reliable recipe for estimating errors and autocorrelation times. One has to calculate the error estimates for different bin sizes l and check if they converge to a limiting value. If convergence is observed the limit ΔA is a reliable error estimate, and $\tau_A^{(\mathrm{int})}$ can be obtained from Eq. 1.30 as

$$\tau_A^{(\mathrm{int})} = \frac{1}{2}\left[\left(\frac{\Delta A}{\Delta A^{(0)}}\right)^2 - 1\right]. \tag{1.33}$$

If no clear convergence is observed as a function of bin size, we know that $\tau_A^{(\mathrm{int})}$ is longer than the simulation

time and that we have to perform much longer simulations to obtain converged observables and reliable error estimates.

To be really sure about convergence and autocorrelation it is very important to start simulations always on tiny systems and to check convergence carefully before simulating larger systems!!

1.5.3 Jackknife analysis

The binning procedure is a straightforward way to determine errors and autocorrelation times for Monte Carlo measurements. For functions of measurements like $U = \langle A \rangle / \langle B \rangle$ it becomes difficult because of error propagation and cross-correlations.

In this case the jackknife procedure can be used. We again split the masurements into M bins of size $N/M \gg \tau^{(\mathrm{int})}$ that should be much larger than any of the autocorrelation times.

We could now evaluate the quantity U in each of the M bins and obtain an error estimate from the variance of these estimates. As each of the bins contains only a rather small number of measurements N/M the statistics will not be good. The jackknife procedure instead works with $M+1$ evaluations of U: an estimate using all bins, and M estimates where all *but* the i-th bin are used. That way we always use a large data set and obtain good statistics. Let U_0 denote the estimate using all available data/bins, and U_i the data using all bins except of the i^{th} bin.

Typical value of M = 128; want bin size to be at least a few autocorrelation timescales

The resulting estimate for U will be

$$U = U_0 - (M-1)(\overline{U} - U_0), \tag{1.34}$$

with a statistical error

$$\Delta U = \sqrt{M-1}\left(\frac{1}{M}\sum_{i=1}^{M}(U_i)^2 - (\overline{U})^2\right)^{\frac{1}{2}}, \tag{1.35}$$

where

$$\overline{U} = \frac{1}{M}\sum_{i=1}^{M} U_i. \tag{1.36}$$

1.5.4 Equilibration or Thermalization

Thermalization is as important as autocorrelations. The Markov chain converges only asymptotically to the desired distribution. Consequently, Monte Carlo measurements should be started only after a large number N_{eq} of equilibration steps, when the distribution is sufficiently close to the asymptotic distribution. N_{eq} has to be much larger than the thermalization time which is defined similar to the autocorrelation time as:

$$\tau_A^{(\text{eq})} = \frac{\sum_{\Delta=1}^{\infty}(\langle A_0 A_\Delta\rangle - \langle A\rangle^2)}{\langle A_0\rangle\langle A\rangle - \langle A\rangle^2} \tag{1.37}$$

The thermalization time is the maximum of all autocorrelation times for all observables and is related to the second largest eigenvalue of the Markov transition matrix. Typically one should thermalize the system for at least ten times the thermalization time before starting measurements.