9 Monte Carlo integration

The most common use for Monte Carlo methods is the evaluation of integrals. This is also the basis of Monte Carlo simulations (which are actually integrations). The basic principles hold true in both cases.

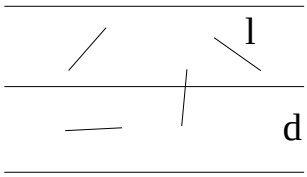
9.1 Basics

Basic idea of the Monte Carlo integration becomes clear from the "dartboard method" of integrating the area of an irregular domain:
Throw darts, i.e. choose points randomly within the rectangular box

$$\frac{A}{A_{\text{box}}} = p(\text{hit inside area}) \leftarrow \frac{\text{\# hits inside area}}{\text{\# hits inside box}}$$

as the # hits $\to \infty$.

An early example of Monte Carlo integration is Buffon's (1707–1788) needle experiment to determine π:



- Throw a needle, length ℓ, on a grid of lines distance d apart, with $\ell < d$.
- The probability that the needle intersects one of the lines is (homework)

  $$P = \frac{2\ell}{\pi d}$$

  Thus, repeating this N times and observing H intersections we obtain

  $$\pi \approx \frac{2\ell N}{dH}$$

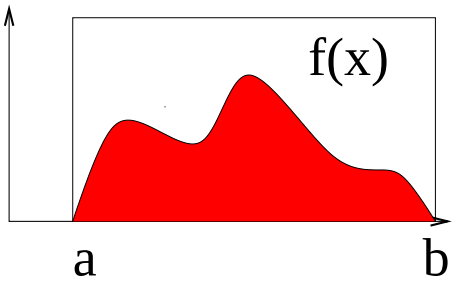
- As an aside, Lazzarini 1901 did Buffon's experiment with 3408 throws, and got 1808 intersections (using $\ell = 5/6\, d$)

  $$\pi \approx \frac{10 \times 3408}{6 \times 1808} = \frac{355}{113} \approx 3.14159292$$

  This is accurate to 3×10^{-7}, way too good a result! Obviously, Lazzarini was aiming for the known value and interrupted the experiment when he got close to that, explaining the peculiar number of throws 3408. Thus, this shows the dangers of fine-tuning the number of samples in Monte Carlo!

  (For more information, see the Wikipedia entry on Buffon's needle.)

9.2 Standard Monte Carlo integration



- Consider 1-dimensional definite integral:

  $$I = \int_a^b dx f(x)$$

  This can be approximated by pulling N random numbers x_i, $i = 1 \ldots N$, from a distribution which is constant in the interval $a \leq x \leq b$:

  $$I \approx \frac{b-a}{N} \sum_{i=1}^{N} f(x_i).$$

  The result becomes exact when $N \to \infty$.

  (Note: the integral can also be performed in the dartboard fashion described on the previous page. However, this is not efficient, why?)

- If the number of random numbers N is large enough, the error in the approximation is $\propto 1/\sqrt{N}$, due to *central limit theorem*, discussed below.

- On the other hand, if we divide the a, b-interval into N steps and use some regular integration routine, the error will be proportional to $1/N^p$, where $p = 1$ even with the most naive midpoint integral rule. Thus, Monte Carlo is not competitive against regular quadratures in 1 dimensional integrals (except in some pathological cases).

- What now happens in higher dimensional integrals? Let us consider d dimensions:

  $$I = \int_V d^d x f(x)$$

  For simplicity, let V be a d-dim hypercube, with $0 \leq x_\mu \leq 1$. Now the Monte Carlo integration proceeds as follows:

  - Generate N random vectors x_i from flat distribution ($0 \leq (x_i)_\mu \leq 1$).

  - As $N \to \infty$,

    $$\frac{V}{N} \sum_{i=1}^{N} f(x_i) \to I \, .$$

  - Error: $\propto 1/\sqrt{N}$ independent of d! (Central limit)

- "Normal" numerical integration methods (Numerical Recipes):

  - Divide each axis in n evenly spaced intervals

  - Total # of points $N \equiv n^d$

  - Error:

    $\propto 1/n$ (Midpoint rule)

    $\propto 1/n^2$ (Trapezoidal rule)

    $\propto 1/n^4$ (Simpson)

- If d is small, Monte Carlo integration has much larger errors than standard methods.

- When is MC as good as Simpson? Let's follow the error

  $$1/n^4 = 1/N^{4/d} = 1/\sqrt{N} \qquad \rightarrow \qquad d = 8.$$

  In practice, MC integration becomes better when $d \gtrsim$6–8.

- In lattice simulations $d = 10^6 - 10^8$!

- In practice, N becomes just too large very quickly for the standard methods: already at $d = 10$, if $n = 10$ (pretty small), $N = 10^{10}$. This implies simply too many evaluations of the function!

9.3 Why error is $\propto 1/\sqrt{N}$?

- This is due to the *central limit theorem* (see, f.ex., L.E.Reichl, Statistical Physics).

- Let $y_i = Vf(x_i)$ (absorbing the volume in f) be the value of the integral using one random coordinate (vector) x_i. The result of the integral, after N samples, is

  $$z_N = \frac{y_1 + y_2 + \ldots + y_N}{N}\,.$$

- Let $p(y_i)$ be the probability distribution of y_i, and $P_N(z_N)$ the distribution of z_N. Now

  $$P_N(z_N) = \int dy_1 \ldots dy_N \; p(y_1) \ldots p(y_N) \delta(z_N - \sum_i y_i/N)$$

- Now

  $$\begin{aligned} 1 &= \int dy\, p(y) \\ \langle y \rangle &= \int dy\, y\, p(y) \\ \langle y^2 \rangle &= \int dy\, y^2\, p(y) \\ \sigma &= \langle y^2 \rangle - \langle y \rangle^2 = \langle (y - \langle y \rangle)^2 \rangle \end{aligned}$$

Here, and in the following, $\langle\alpha\rangle$ means the expectation value of α (mean of the distribution $p_\alpha(\alpha)$), and σ is the *width* of the distribution.

- The error of the Monte Carlo integration (of length N) is *defined* as the width of the distribution $P_N(z_N)$, or more precisely,

$$\sigma_N^2 = \langle z_N^2\rangle - \langle z_N\rangle^2 .$$

- Naturally $\langle z_N\rangle = \langle y\rangle$.
- Let us calculate σ_N:

- Let us define the Fourier transformation of $p(y)$:

$$\phi(k) = \int dy\ e^{ik(y-\langle y\rangle)}\, p(y)$$

Similarly, the Fourier transformation of $P_N(z)$ is

$$\begin{aligned}\Phi_N(k) &= \int dz_N\, e^{ik(z_N-\langle z_N\rangle)}\, P_N(z_N)\\ &= \int dy_1\ldots dy_N\ \ e^{i(k/N)(y_1-\langle y\rangle+\ldots+y_N-\langle y\rangle)}\ \ p(y_1)\ldots p(y_N)\\ &= [\phi(k/N)]^N\end{aligned}$$

- Expanding $\phi(k/N)$ in powers of k/N (N large), we get

$$\phi(k/N) = \int dy\ e^{i(k/N)(y-\langle y\rangle)}\, p(y) = 1 - \frac{1}{2}\frac{k^2\sigma^2}{N^2} + \ldots$$

Because of the oscillating exponent, $\phi(k/N)$ will decrease as $|k|$ increases, and $\phi(k/N)^N$ will decrease even more quickly. Thus,

$$\Phi_N(k) = \left(1 - \frac{1}{2}\frac{k^2\sigma^2}{N^2} + O(\frac{k^3}{N^3})\right)^N \longrightarrow e^{-k^2\sigma^2/2N}$$

- Taking the inverse Fourier transform we obtain

$$\begin{aligned}P_N(z_N) &= \frac{1}{2\pi}\int dk\ e^{-ik(z_N-\langle y\rangle)}\Phi_N(k)\\ &= \frac{1}{2\pi}\int dk\ e^{-ik(z_N-\langle y\rangle)}e^{-k^2\sigma^2/2N}\\ &= \sqrt{\frac{N}{2\pi}}\frac{1}{\sigma}\exp\left[-\frac{N(z_N-\langle y\rangle)^2}{2\sigma^2}\right].\end{aligned}$$

- Thus, the distribution $P_N(z_N)$ approaches Gaussian as $N\to\infty$, and

$$\sigma_N = \sigma/\sqrt{N}.$$

9.4 In summary: error in MC integration

If we integrate

$$I = \int_V dx\, f(x)$$

with Monte Carlo integration using N samples, the error is

$$\sigma_N = V\sqrt{\frac{\langle f^2\rangle - \langle f\rangle^2}{N}}\,.$$

In principle, the expectation values here are exact properties of the distribution of f, i.e.

$$\langle f\rangle = \frac{1}{V}\int dx f(x) = \int dy\, y\, p(y) \qquad \langle f^2\rangle = \frac{1}{V}\int dx f^2(x) = \int dy\, y^2 p(y).$$

Here $p(y)$ was the distribution of values $y = f(x)$ when the x-values are sampled with flat distribution. $p(y)$ can be obtained as follows: the probability of the x-coordinate is flat, i.e. $p_x(x) = C =$const., and thus

$$\begin{aligned} &p_x(x)dx = C\frac{dx}{dy}dy \equiv p(y)dy \qquad \Rightarrow \\ &p(y) = C/(dy/dx) = C/f'(x(y))\,. \end{aligned}$$

Of course, we don't know the expectation values beforehand! In practice, these are substituted by the corresponding Monte Carlo estimates:

$$\langle f\rangle \approx \frac{1}{N}\sum_i f(x_i) \qquad \langle f^2\rangle \approx \frac{1}{N}\sum_i f^2(x_i)\,.$$

Actually, in this case we must also use

$$\sigma_N = V\sqrt{\frac{\langle f^2\rangle - \langle f\rangle^2}{N-1}}$$

because using (9.4) here would underestimate σ_N.

Often in the literature the real expectation values are denoted by $\langle\langle x\rangle\rangle$, as opposed to the Monte Carlo estimates $\langle x\rangle$.

The error obtained in this way is called the 1-σ error; i.e. the error is the width of the Gaussian distribution of

$$f_N = 1/N\sum_i f(x_i).$$

It means that the true answer is within $V\langle f\rangle \pm \sigma_N$ with $\sim 68\%$ probability. This is the most common error value cited in the literature.

This assumes that the distribution of f_N really is Gaussian. The central limit theorem guarantees that this is so, provided that N is large enough (except in some pathological cases).

However, in practice N is often *not* large enough for the Gaussianity to hold true! Then the distribution of f_N can be seriously off-Gaussian, usually skewed. More refined methods can be used in these cases (for example, modified jackknife or bootstrap error estimation). However, often this is ignored, because there just are not enough samples to deduce the true distribution (or people are lazy).

9.5 Example: convergence in MC integration

- Let us study the following integral:

$$I = \int_0^1 dx\, x^2 = 1/3\,.$$

- Denote $y = f(x) = x^2$ and $x = f^{-1}(y) = \sqrt{y}$.
- If we sample x with a flat distribution, $p_x(x) = 1$, $0 < x \leq 1$, the probability distribution $p_f(y)$ of $y = f(x)$ can be obtained from

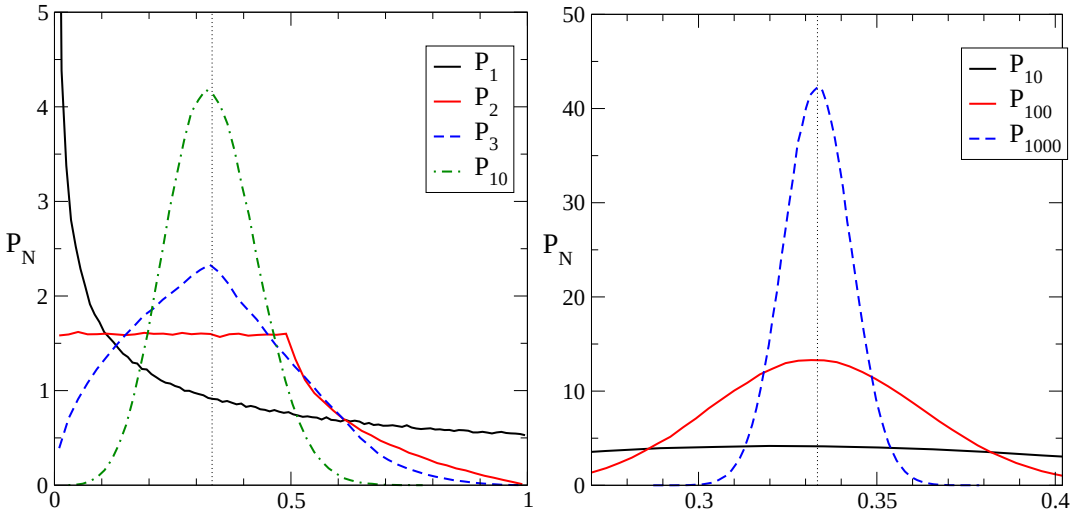
$$\begin{aligned} p_x(x)dx = p_x(x)|\frac{dx}{dy}|dy \equiv p_f(y)dy \qquad & \Rightarrow \\ p_f(y) = |\frac{dx}{dy}| = |1/f'(x)| = \frac{1}{2}x^{-1} = \frac{1}{2}y^{-1/2} & \end{aligned}$$

  where $0 < y \leq 1$.

- In more than 1 dim: $p_f(\vec{y}) = \left|\left|\frac{\partial x_i}{\partial y_j}\right|\right|$, the Jacobian determinant.
- The result of a Monte Carlo integration using N samples x_i, $y_i = f(x_i)$, is $I_N = (y_1 + y_2 + \ldots + y_N)/N$.
- What is the probability distribution P_N of I_N? This tells us how badly the results of the (fixed length) MC integration scatter.

$$\begin{aligned} P_1(z) &= p_f(z) = 1/\sqrt{z} \\ P_2(z) &= \int_0^1 dy_1 dy_2\; p_f(y_1)p_f(y_2)\; \delta(z - \frac{y_1+y_2}{2}) \\ &= \begin{cases} \pi/2 & 0 < y \leq 1/2 \\ \arcsin\frac{1-y}{y} & 1/2 < y \leq 1 \end{cases} \\ P_3(z) &= \int_0^1 dy_1 dy_2 dy_3\; p_f(y_1)p_f(y_2)p_f(y_3)\, \delta(z - \frac{y_1+y_2+y_3}{3}) = \ldots \\ &\ldots \end{aligned}$$

P_N, measured from the results of 10^6 independent MC integrations for each N:



The expectation value = 1/3 for all P_N.

The width of the Gaussian is

$$\sigma_N = \sqrt{\frac{\langle\langle f^2\rangle\rangle - \langle\langle f\rangle\rangle^2}{N}} = \sqrt{\frac{4}{45N}}$$

where

$$\langle\langle f^2\rangle\rangle = \int_0^1 dx f^2(x) = \int_1^\infty dy y^2 p_f(y) = 1/5.$$

For example, using $N = 1000$, $\sigma_{1000} \approx 0.0094$. Plotting

$$C \exp\left[-\frac{(x-1/3)^2}{2\sigma_{1000}^2}\right]$$

in the figure above we obtain a curve which is almost undistinguishable from the measured P_{1000} (blue dashed) curve.

In practice, the expectation value and the error is of course measured from a single Monte Carlo integration. For example, using again $N = 1000$ and the Monte Carlo estimates

$$\langle f\rangle = \frac{1}{N}\sum_i f_i\,, \quad \langle f^2\rangle = \frac{1}{N}\sum_i f_i^2\,, \quad \sigma_N = \sqrt{\frac{\langle f^2\rangle - \langle f\rangle^2}{N-1}}$$

we obtain the following results (here 4 separate MC integrations):
1: 0.34300 ± 0.00950
2: 0.33308 ± 0.00927
3: 0.33355 ± 0.00931
4: 0.34085 ± 0.00933
Thus, both the average and the error estimate can be reliably calculated in a single MC integration.

9.6 Another example:

What about integral $I = \int_0^1 dx\, x^{-1/2} = 2$? Now $\langle\langle f^2\rangle\rangle = \int_0^1 dx\, x^{-1} = \infty$, diverges logarithmically. What happens to the MC error estimate?

The assumptions in the central limit theorem do not hold in this case. However, in practice the Monte Carlo integration works also now, and the distribution P_N approaches something like a Gaussian shape, and the characteristic width (and thus the error of the MC integration) behaves as $\propto 1/\sqrt{N}$. However, the standard formula with $\langle\langle f^2\rangle\rangle$ does not work.

Now

$$P_1(y) = p_f(y) = 1/f' = y^{-3},$$

where $1 \leq y < \infty$. Thus, $y^2 P_1(y)$ is not integrable, and since $P_2(y) \sim y^{-3}$ when $y \to \infty$, neither is $y^2 P_2(y)$. This remains true for any N(?). Thus, formally the error σ_N is infinite!

Nevertheless, we can perform standard MC integrations with MC estimates for $\langle f\rangle$, $\langle f^2\rangle$ and plug these into the formula for σ_N:

For example, some particular results
$N = 1000:$ $\quad 1.897 \pm 0.051$
$N = 10000:$ $\quad 1.972 \pm 0.025$
$N = 100000:$ $\quad 2.005 \pm 0.011$
$N = 1000000:$ $\quad 1.998 \pm 0.003$
Results may fluctuate quite a bit, since occasionally there may occur some very large values of $f(x) = 1/\sqrt{x}$.

Why does this work? Even though

$$\langle f^2\rangle_N = \frac{1}{N}\sum_i f_i^2\,,$$

does not have a well-defined limit as $N \to \infty$, the error estimate

$$\sigma_N = \sqrt{\frac{\langle f^2\rangle_N - \langle f\rangle_N^2}{N-1}}$$

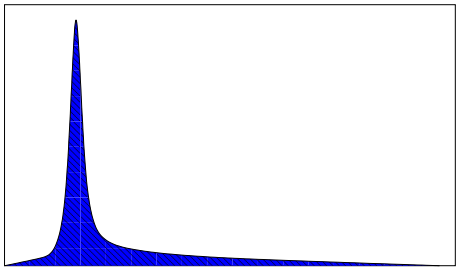
does, becaues of the extra factor of $(N-1)$ in the denominator! Thus, usually one can use the formula above as long as $f(x)$ is integrable. However, the convergence of the error is *slower than* $1/\sqrt{N}$, depending on the function!
What about integrals with non-integrable subdivergences? Example:

$$\int_{-1}^{1} dx \frac{1}{x} = 0$$

(in principal value sense). Usually Monte Carlo methods are not able to handle these (test the above!). The integral requires special treatment of the subdivergences (not discussed here).

9.7 Importance sampling

Often the integrand has a very small value on a dominant fraction of the whole integration volume:



If the points are chosen evenly in the integration volume, the small minority of the points close to the "peak" give the dominant contribution to the integral.

Importance sampling: choose the random points so that more points are chosen around the peak, less where the integrand is small.

$\mapsto$ reduces the variance σ, and thus reduces the error.

Importance sampling is essential in Monte Carlo simulations! In that case we integrate $\int [d\phi] \exp[-S]$, which is exponentially strongly peaked function (peak near the corner where S is small).

- Let us integrate $I = \int_V dV\, f(x)$
- Choose a distribution $p(x)$, which is close to the function $f(x)$, but which is simple enough so that it is possible to generate random x-values from this distribution.
- Now

$$I = \int dV p(x) \frac{f(x)}{p(x)} \,.$$

- Thus, if we choose random numbers x_i from distribution $p(x)$, we obtain

$$I = \lim_{N\to\infty} \frac{1}{N} \sum_i \frac{f(x_i)}{p(x_i)}$$

- Since f/p is flatter than f, the variance of f/p will be smaller than the variance of $f \to$ error will be smaller (for a given N).
- Ideal choice: $p(x) \propto |f(x)|$. Now the variance vanishes! In practice not usually possible, since the proportionality constant is the unknown integral!
- However, exactly this choice is done in Monte Carlo simulations.
- In Monte Carlo simulations, we want to evaluate integrals of type

$$I = \int [d\phi]\, f(\phi)\, e^{-S(\phi)} / \int [d\phi]\, e^{-S(\phi)}$$

(using here notation $[d\phi] = \prod_x d\phi$.)

- Importance sampling: we generate random $\phi(x)_i$ -configurations with probability

  $$p(\phi) = \text{const.} \times e^{-S(\phi)}$$

  (with unknown normalization constant).

- Applying the previous result:

  $$I \approx \frac{1}{N} \sum_i f(\phi_i)$$

- Note that

  $$Z = \int [d\phi]\, e^{-S(\phi)}$$

  cannot be directly calculated with importance sampling, due to the unknown normalization constant.

- In a sense, this importance sampling turns the plain Monte Carlo integration into simulation: the configurations $\phi(x)$ are sampled with the Bolzmann probability $\propto e^{-S}$, which is the physical probability of a configuration. Thus, the integration process itself mimics nature!

- How to generate ϕ's with probability $\propto e^{-S(\phi)}$? That is the job of the Monte Carlo update algorithms.

9.8 Example: importance sampling

Integrate

$$I = \int_0^1 dx\, (x^{-1/3} + x/10) = 31/20 = 1.55$$

Standard MC gives error

$$\sigma_N = \sqrt{\frac{\langle f^2 \rangle - \langle f \rangle^2}{N-1}} \approx \frac{0.85}{\sqrt{N-1}}$$

Let us now assume that we don't know the function but we can evaluate it. We recognize that it diverges pretty much like $x^{-1/3}$ as x is small. Let us therefore propose to do the integral with importance sampling, using sampling probability

$$p(x) = 2/3x^{-1/3}, \qquad 0 < x \leq 1.$$

How to obtain random numbers with distribution $p(x)$? Let us make a change of variable so that the distribution in the new variable is flat:

$$p(x)dx = p(x)\frac{dx}{dy}dy = dy \qquad \Rightarrow \qquad y = \int_0^x dx'\, p(x') = x^{2/3}$$

Thus, if y is from a flat distribution $y \in (0,1]$, $x = y^{3/2}$ is from distribution $p(x) = 2/3x^{-1/3}$. In the new variable y the integral becomes

$$I = \int_0^1 dx p(x)\frac{f(x)}{p(x)} = \int_0^1 dy \frac{f(x(y))}{p(x(y))}$$

Thus, if we denote $g = f/p$, we have $\langle g \rangle = 31/20$ and $\langle g^2 \rangle = 2.4045$. Thus, the width of the result in MC integration of g is

$$\sigma_N = \sqrt{\frac{\langle g^2 \rangle - \langle g \rangle^2}{N-1}} \approx= \frac{0.045}{\sqrt{N-1}}$$

This is ~ 20 times narrower than with the naive method!

The recipe for the MC integration with importance sampling is thus:
- generate N random numbers y_i from flat distribution
- $x_i = y_i^{3/2}$ are from distribution $p(x)$
- calculate the average of $g_i = f(x_i)/p(x_i)$

Indeed:

N	naive	importance
100	1.4878 ± 0.0751	1.5492 ± 0.0043
10000	1.5484 ± 0.0080	1.5503 ± 0.0004

9.9 Note: some standard MC integration routines

Powerful Monte Carlo integration routines are included in several numerical packages. However, typically these tend to fail when number of dimensions is larger than ~ 15. Thus, these are useless for Monte Carlo simulations.
The following routines are included in the GSL (GNU scientific library, available for free). See also Numerical Recipes 7.8 for further details.

- VEGAS: uses approximate importance sampling (together with several other tricks). It does several passes over the integral, and uses the results of the previous passes to improve its estimate of the importance sampling function.

  Described in:

  G.P. Lepage, 'A New Algorithm for Adaptive Multidimensional Integration', Journal of Computational Physics 27, 192-203, (1978)

- MISER: uses *stratified sampling*

9.10 Note: quasirandom sequences

- Quasirandom sequences are "random" number sequences which are not random at all, but are (for our purposes) specially constructed to cover the large-dimensional volume (hypercube) as evenly as possible.
- Error in quasirandom integration usually $\propto 1/N$ or even faster, where N is the number of points in the sequence (cf. $1/\sqrt{N}$ in random Monte Carlo).
- Sequences depend on d and N.
- *Does not work with importance sampling $\rightarrow$ not useful for Monte Carlo simulation.*
- Numerical Recipes sec. 7.7; implementations in GSL (GNU scientific library, available for linux and other systems).

10 Importance sampling and update algorithms

10.1 Canonical ensemble

- Common uses for the Monte Carlo method is to study either *thermodynamics* of some statistical model defined on a lattice, or study latticized quantum field theories. For example, we may have a (real-valued,say) field ϕ_x, which is defined at each lattice coordinate x, with the Hamiltonian energy functional $H(\phi)$. We shall discuss here regular (hyper) cubic lattices, where

  $$x_i = an_i, \qquad n_i = 0 \ldots N_i,$$

  where a is the lattice spacing (in physical units).

- The most common statistical ensemble we meet is the *canonical ensemble*, which is defined by the partition function

  $$Z = \int [d\phi] \, \exp\left[-\frac{H(\phi)}{k_B T}\right].$$

  Here $[d\phi] \equiv \prod_x d\phi_x$. The free energy is defined as $F = -k_B T \ln Z$. This is now in statistical physics language, but naturally readily generalises to QFT with the substitution $H/(k_B T) \to S_{\rm Eucl.}$.

- We would like to evaluate Z, and especially *thermal average* of some quantity A:

  $$\langle A \rangle = \frac{1}{Z} \int [d\phi] \, A(\phi) \, \exp\left[-\frac{H(\phi)}{k_B T}\right].$$

  The dimensionality of the integrals is huge:

  (# points in space) $\times$ (dim. of field ϕ_x) $\sim 10^6 - 10^9$.

  Thus, *some* kind of (quasi?) Monte Carlo integration is required.

- The integral is also *very strongly peaked:*

  $$\int [d\phi] \, \exp\left[-\frac{H(\phi)}{k_B T}\right] = \int dE n(E) e^{-E/k_B T} \, ,$$

  which is an integral over sharply localized distribution. Here $n(E)$ is the density of states at energy E, i.e.

  $$n(E) = \int [d\phi] \, \delta(H(\phi) - E)$$

  Only states with $E \lesssim \langle E \rangle$ contribute, which is an exponentially small fraction of the whole phase space. This kills standard "simple sampling" Monte Carlo, which is homogeneous over the phase space. This is a form of the so-called **overlap problem** – we are not sampling the interesting configurations nearly often enough.

10.2 Importance sampling

Importance sampling takes care of the overlap problem:

- Select N configurations ϕ_i, chosen with the *Boltzmann probability*

$$p(\phi) = \frac{\exp[-H(\phi)/k_B T]}{Z}$$

The set $\{\phi_i\}$ is called an *ensemble* of configurations.

Note that in practice we cannot calculate Z! Thus, we actually do not know the normalization of $p(\phi)$. However, this is sufficient for Monte Carlo simulation and measuring thermal averages.

- Measure observable $A_i = A(\phi_i)$ from each configuration.
- The expectation value of A is now

$$\langle A \rangle \equiv \frac{1}{N} \sum_i A_i \quad \to \quad \langle\langle A \rangle\rangle, \qquad \text{as } \; N \to \infty.$$

If now the value of A_i's are $\sim$ equal for all i, all of the configurations contribute with the same order of magnitude. This is the central point of importance sampling in Monte Carlo simulations.

The result (10.2) follows directly from the general results we got in Sect. 1. about importance sampling Monte Carlo integration, or even more directly by realizing that the A_i-values obviously come from distribution

$$\begin{aligned} p_A(A') &= \int [d\phi]\; p(\phi)\; \delta(A' - A[\phi]) \\ &\Rightarrow \quad \bar{A} \equiv \int dA\; A\; p_A(A) = \int [d\phi]\; A[\phi]\; p(\phi) = \langle\langle A \rangle\rangle. \end{aligned}$$

10.3 How to generate configurations with $p(\phi) = \exp[-H(\phi)/k_B T]/Z$?

- Directly – from scratch – we do not know how to do this! (except in some limited simple cases.)

- The basic method (which is really behind almost all of the modern Monte Carlo simulation technology) is based on the proposal by *Metropolis, Rosenbluth*[2] *and Teller*[2] (J. Chem. Phys 21 1087 (1953)): we can modify existing configurations in small steps, Monte Carlo update steps. Thus, we obtain a *Markov chain* of configurations

  $$\phi_1 \mapsto \phi_2 \mapsto \phi_3 \mapsto \ldots \phi_n \ldots$$

  With suitably chosen updates and large enough n the configuration ϕ_n is a "good" sample of the canonical probability $p(\phi)$.

  How does this work? Let us look at it in detail:

- Let f be an "update", a modification of an existing configuration ϕ: $\phi \stackrel{f}{\mapsto} \phi'$. It can be here a "small" update, one spin variable or even a component of it (if ϕ is a vector), or arbitrarily "large" one, for example all of the d.o.f's (usually just repeated application of small updates).

- *Transition probability* $W_f(\phi \mapsto \phi')$ is the probability distribution of configurations ϕ', obtained by one application of f on some (fixed) configuration ϕ. It naturally has the following properties:

  $$W_f(\phi \mapsto \phi') \geq 0\,, \qquad \int d\phi'\, W_f(\phi \mapsto \phi') = 1$$

  The latter property comes from the fact that f must take ϕ *somewhere* with probability 1.

- We can also apply the update f to probability distributions $p(\phi)$:

  $$p(\phi) \stackrel{f}{\mapsto} p'(\phi') \equiv \int d\phi\, p(\phi)\, W_f(\phi \mapsto \phi').$$

  This follows directly from the fact that we can apply f to an ensemble of configurations from distribution $p(\phi)$.

- What requirements must a "good" update f fulfill:

  A) f must preserve the equilibrium (Boltzmann) distribution $p_{\mathrm{eq.}}(\phi) \propto \exp[-H(\phi)/k_B T]$:

  $$\int d\phi\, W_f(\phi \mapsto \phi')\, p_{\mathrm{eq.}}(\phi) = p_{\mathrm{eq.}}(\phi')$$

  In other words, $p_{\mathrm{eq.}}(\phi)$ is a ***fixed point*** of f.

  B) f must be **ergodic:** starting from *any* configuration, repeated application of f brings us to arbitrarily close to any other configuration. This implies that $\int d\phi\, W_f(\phi \mapsto \phi') > 0$ for any ϕ'.

- These two simple and and rather intuitive properties are sufficient to guarantee the following 2 important results:

  I) *Any* initial ensemble approaches the equilibrium canonical ensemble $\rightarrow p_{\text{eq.}}$ as f is applied to it ("thermalization").

  II) If we sum up the distribution of all of the configurations in a single Markov chain
  $$\phi_0 \mapsto \phi_1 \mapsto \phi_2 \mapsto \phi_3 \ldots$$
  the distribution approaches $p_{\text{eq.}}$, as the number of configurations $\rightarrow \infty$.

- **Proof:** let $p(\phi)$ be the probability distribution of the initial ensemble of configurations. After applying f once to all configs in the ensemble, we obtain the distribution $p'(\phi')$. Let us now look how the norm $||p - p_{\text{eq}}||$ evolves:

$$\begin{aligned} ||p' - p_{\text{eq.}}|| &\equiv \int d\phi' \, |p'(\phi') - p_{\text{eq.}}(\phi')| \\ &= \int d\phi' \, | \int d\phi \, W_f(\phi \mapsto \phi')(p(\phi) - p_{\text{eq.}}(\phi))| \\ &\leq \int d\phi' \int d\phi \, W_f(\phi \mapsto \phi')|p(\phi) - p_{\text{eq.}}(\phi)| = ||p - p_{\text{eq.}}|| \end{aligned}$$

  Thus, we get closer to the equilibrium when f is applied. Nevertheless, this is not sufficient to show that we really get there; it might happen that $||p - p_{\text{eq.}}||$ reaches a plateau without going to zero. This would mean that there exists another fixed point ensemble besides the equilibrium one, i.e. an ensemble which is preserved by f.

  However, the *ergodicity* forbids this: let us assume that p_{fix} is another fixed point, and let $||p_{\text{fix}} - p_{\text{eq.}}|| > 0$. Then we can split ϕ's into two non-empty sets: set A has those configurations where $(p_{\text{fix}}(\phi) - p_{\text{eq.}}(\phi)) \geq 0$ and set B the rest. Because of ergodicity, there must be some configurations ϕ' for which $W_f(\phi \mapsto \phi')$ is non-zero for some ϕ in set A and some other in set B. Thus, on the 3rd line $\leq \quad \rightarrow \quad <$, and p cannot be a fixed point.

10.4 Detailed balance

The standard update algorithms satisfy the following detailed balance condition:

$$\frac{W_f(\phi \mapsto \phi')}{W_f(\phi' \mapsto \phi)} = \frac{p_{\text{eq.}}(\phi')}{p_{\text{eq.}}(\phi)} = e^{(H(\phi) - H(\phi'))/k_B T}$$

Obviously, $p_{\text{eq.}}(\phi)$ is a fixed point of this update. Thus, (I) *detailed balance* and (II) *ergodicity* are sufficient for a correct Monte Carlo update. Detailed balance is not a necessary condition; there may be updates which do not satisfy detailed balance. But it is much more difficult then

to prove that these updates preserve the Bolzmann distribution. Thus, all of the commonly used update algorithms satisfy detailed balance.

The physical interpretation of detailed balance is *microscopic reversibility* of the update algorithm.

10.5 Common transition probabilities

The detailed balance condition is a restriction for W_f; however, it is still very general. For concretenss, let us now consider an update step of one variable (lattice spin, for example). Some of the most common choices for W_f are (all of these satisfy detailed balance):

- **Metropolis:** (Metropolis *et al*, 1953)

  $$W_f(\phi \mapsto \phi') = C \times \begin{cases} \exp[-\delta H/k_B T] & \text{if } \delta H > 0 \\ 1 & \text{otherwise} \end{cases},$$

  where $\delta H \equiv H(\phi') - H(\phi)$ is the change in energy, and C is a normalization constant.

- **Heat bath:**

  $$W_f(\phi \mapsto \phi') = C \exp[-H(\phi')/k_B T]$$

  This does not depend on old ϕ at all!

- **Overrelaxation:** The standard overrelaxation as presented in the literature is a special non-ergodic update. It assumes that the energy functional H has a symmetry wrt. some reflection in the configuration space. For example, there may exist some α so that

  $$H(\alpha - \phi) = H(\phi)$$

  for any ϕ, and $p(\phi) = p(\alpha - \phi)$. Then the reflection ("overrelaxation")

  $$\phi \mapsto \phi' = \alpha - \phi$$

  is a valid update. This satisfies detailed balance, because $H(\phi)$ and $p(\phi)$ do not change. However, it obviously is not ergodic! One has to mix overrelaxation with other updates to achieve ergodicity. Nevertheless, overrelaxation is a very useful update, because it generally reduces autocorrelations better than other updates (to be discussed).

10.6 Updating O(2) sigma model:

- Let us illustrate these update methods with a concrete example, O(2) σ-model or the XY model:

$$\begin{aligned} H/k_BT \equiv S &= -\beta \sum_{<xy>} s_x \cdot s_y = -\beta \sum_{<xy>} \cos(\theta_x - \theta_y) \\ Z &= \int_{-\pi}^{\pi} [\prod_x d\theta_x] e^{-S[\theta]} \end{aligned}$$

  Here s_x is a 2-component vector with $|s_x| = 1$, and θ_x is its angle from, say, 1 -axis. x and y are discrete coordinate vectors (in 2 or 3 dimensions), and $< x, y >$ refers to *nearest-neighbour* coordinate pairs.

- **Physics of the model:** In 3 dimensions the XY model has a phase transition at $\beta = \beta_c = 0.454165(4)$. The model exhibits *spontaneous symmetry breaking*; there is spontaneous magnetization if $\beta > \beta_c$ ($T < T_c$), i.e.

$$\langle |M| \rangle = \frac{1}{V} \left\langle \left| \sum_x s_x \right| \right\rangle \neq 0 \quad \text{as } V \equiv N^3 \to \infty.$$

  The transition is of second order, and the most important critical exponents have been measured/calculated; for example,

$$|M| = (\beta - \beta_c)^b \quad b \approx 0.35$$

  Universality: Phase transitions in systems which have 2d (internal) rotational symmetry (O(2)) have the same critical exponents as the XY model:

  – superfluidity λ-transition in liquid ^{4}He (space shuttle experiment)

  – ferromagnets with an "easy plane"

  – some liquid crystals

  – density waves

  – type II superconductors (?)

  In 2 dimensions, there is no magnetization,[1] but there is still a phase transition, *Kosterlitz-Thouless* transition.

  A lot more information about XY model can be found in [Pelisetto, Vicari, cond-mat/0012164], for example.

---

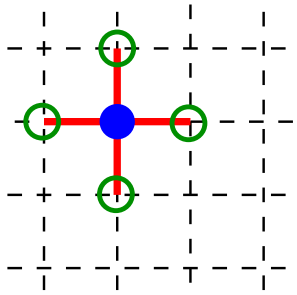
[1] This is due to the *Coleman-Mermin-Wagner theorem:* in 2d, continuous symmetries cannot break spontaneously!

How to update O(2) model:

- Choose a variable at site x for updating
- Calculate the *local action*, the part of the action which depends on the variable s_x:

$$S_x(s_x) = -\beta \sum_{y=\text{n.n.of } x} s_x \cdot s_y = -s_x \cdot V = -v \cos\alpha, \qquad V = \beta \sum_{y=\text{n.n.of } x} s_y$$

Here $v = |V|$ and α is the angle between s_x and V. When we modify the variable s_x, the change in total action is $\delta S = \delta S_x$.



- **Heat bath:**

Choose new s_x with probability

$$p(s_x) \propto e^{-S_x(s_x)} = e^{v\cos\alpha}$$

- Satisfies detailed balance

- Computer gives random numbers from uniform distribution, $X \in [0, 1)$. We obtain s from distribution $p(s)$ from the inversion

$$X = \int_{\min}^{s} ds' p(s') = C_\alpha \int_{-\pi}^{\alpha} d\alpha' e^{v\cos\alpha'}$$

where $C_\alpha = \int_{-\pi}^{\pi} d\alpha' e^{v\cos\alpha'}$ is a normalization constant.

- We're unlucky: $p(s)$ is not very easily integrable, thus, making heat bath updates complicated. Integrability is often a problem for many actions. For an easier case, consider O(3) sigma model. However, we can always generate random numbers with distribution $p(s)$ by using some version of the rejection method.

- Heat bath update is quite common in Monte Carlo simulations. In gauge theories, efficient implementation $\exists$ for SU(2)

- **Metropolis:**

  How to generate the new angle variable with the Metropolis probability form? Integrating W_M is as complicated as the heat bath integral. However, we may do an efficient *restricted* Metropolis transition by using the following accept/reject method:

  Choose new θ'_x with

  $$\theta'_x = \theta_x + C(X - 0.5) \qquad (\mod 2\pi),$$

  where C is a tunable constant. θ'_x is *accepted* with probability

  $$W_M(\theta_x \mapsto \theta'_x) = \min(1, e^{S_x(\theta_x) - S_x(\theta'_x)}),$$

  if rejected, either repeat or leave θ_x as is.

  - Satisfies detailed balance.

  - More generally: instead of using the uniform distribution $\theta'_x \in [\theta_x - C/2, \theta_x + C/2]$, we could choose θ'_x from any distribution $f(\theta_x - \theta'_x)$, which satisfies $f(a) = f(-a)$. For example, $f(a)$ could be a Gaussian distribution.

  - Function f (or constant C) is tuned to optimize the performance; usually so that the acceptance is $\sim 50 - 70\%$.

  - If $C = 2\pi$, θ' will be evenly distributed from 0 to 2π. However, the rejection rate will be (usually) very large.

  - Metropolis is very versatile method, and can be applied to almost any problem. We only have to evaluate e^S! No integrals or inversions. One Metropolis update is also usually quite fast. It is also the "original" update, first one to be used in a real simulation.

  - If you repeat Metropolis update of ϕ_x (fixed site) many times, the final distribution $\mapsto$ heat bath update!

- **Overrelaxation:**

  Reflect s_x to the "other side" of the potential S_x:

  $\alpha \to -\alpha$, or equivalently

  $$s_x \to 2\frac{s_x \cdot V}{v^2}\, V - s_x.$$

  - Deterministic, very fast to implement.

  - Satisfies detailed balance

  - *Not ergodic:* S never changes: in other words, the update is *microcanonical*.

  - Must be usually mixed with other updates to achieve ergodicity.

  - Nevertheless, often overrelaxation is more efficient than the other updates above.

- In this case $S_x(s)$ is always symmetric wrt. V (V varies from site to site and update to update, but it is always simple to find!). If S_x is not symmetric, overrelaxation is much more difficult (often not worth it).

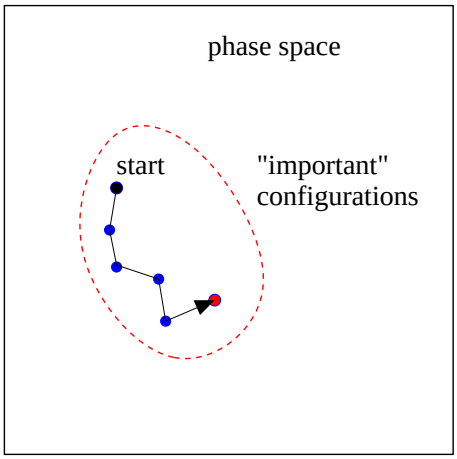
- Not applicable to discrete spins.

10.7 Update sweep

Repeat the above single-site (ϕ_x) updates for all of the sites on the lattice $\rightarrow$ update sweep.

Typical program structure:
1) 1 or more update sweeps
2) measure what you want, accumulate or print
3) repeat from 1) until enough statistics

Even after all of the sites have been updated, the system evolves only a finite step in the phase space; it "remembers" the previous configuration.



Autocorrelation time τ_A: number of update sweeps required to make the configuration statistically independent from the starting configuration.

$\Rightarrow$ The number of independent configurations in a Monte Carlo simulation of N configurations is N/τ_A.

$\Rightarrow$ $\boxed{\text{Errors} \propto \sqrt{\tau_A/N}.}$

Thus, a good update minimises τ_A. Rule of thumb:

Metropolis $<$ heat bath $<$ overrelaxation

where $<$ shows the "goodness" relation of the algorithm, i.e. $1/\tau_A$. However, the real performance depends on the details. We always should compare to the (wall-clock) time, not to the # of updates!

Because of non-ergodicity, overrelaxation is usually mixed with heat bath or Metropolis: for example, we can do
5 overrelaxation sweeps + 1 heat bath sweep.

Autocorrelation time diverges at phase transition points (or more generally, when the correlation length diverges): *critical slowing down*.

10.8 Ising model

The Ising model is the simplest discrete spin model. In this case the field variable $\phi_x \to s_x = \pm 1$, and the partition function is

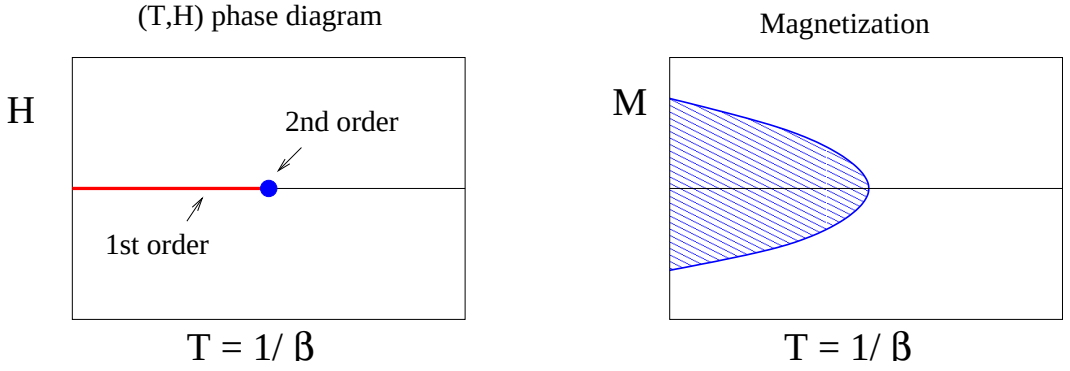
$$Z = \sum_{\{s_x = \pm 1\}} \exp\left[-\beta \left(\tfrac{1}{2} \sum_{<xy>} (1 - s_x s_y) + H \sum_x s_x \right) \right]$$

Here $< xy >$ goes over the nearest-neighbour sites of the rectangular lattice, and H is an external magnetic field. We shall consider here mostly the case where $H = 0$.

When $H = 0$ the Ising model has a second-order phase transition in all dimensions > 1. In 2 dimensions, the Ising model has been solved analytically (Onsager), and the transition is at $\beta_c = \ln(1 + \sqrt{2}) \approx 0.88137$. In 3 dimensions, $\beta_c \approx$???
If $\beta > \beta_c$ ("$T < T_c$"), the system shows spontaneous magnetization:

$$\langle |M| \rangle = \langle \left| \frac{1}{V} \sum_x s_x \right| \rangle \neq 0$$

Phase diagram:



Universality:
– Ferromagnetism, Curie point
– Liquid-vapour transition (lattice gas)
– Generic transition for systems without explicit symmetry

10.9 Updating the Ising model

- Let us assume here $H = 0$, and that we are in 2 dimensions.
- Choose spin s_x. The *local energy functional*

$$S_x(s_x) = \beta/2 \sum_{y=\text{n.n.of } x} (1 - s_x s_y) = \beta/2 \left(4 - s_x \sum_{y=\text{n.n.of } x} s_y \right)$$

S_x has only 5 possible values! (Can be calculated beforehand.)

- **Heat bath:** Choose new $s_x = \pm 1$ with probability

$$p(s_x) = \frac{e^{-S_x(s_x)}}{e^{-S_x(+1)} + e^{-S_x(-1)}}$$

(see sect. 2.)

- **Metropolis:** Flip the spin $s_x \mapsto s'_x = -s_x$, and calculate $\delta S_x = S_x(s'_x) - S_x(s_x)$. Accept this flip with probability

$$p_{\mathrm{accept}} = \min(1, e^{-\delta S_x})$$

If the update is rejected, leave s_x as is (do not redo until accepted! You can redo Metropolis a fixed number of times, though.)

- **Kawasaki:** Choose a nearest-neighbour pair s_x, s_y. If $s_x \neq s_y$, consider exchanging the spins $s_x \mapsto s'_x = s_y$ and $s_y \mapsto s_x$. This is accepted with the Metropolis probability

$$p_{\mathrm{accept}} = \min(1, e^{-\delta S})$$

If the update is rejected, leave the pair as is.

Kawasaki update preserves the magnetization M. It is used if we really want to study an ensemble with fixed magnetization (lattice gas: fixed particle number!)

10.10 Boundary conditions

How do we choose the boundary conditions on a rectangular lattice? There are several choices, usually dictated by the physics we want to study. Let us assume 2-dimensional $N \times N$ lattice, with coordinates $x, y = 1 \ldots N$.

- **Periodic boundaries:** This is the most popular choice when we want to *minimize* the effect of boundaries. The neighbours wrap around the edges, thus the coordinate pairs $(N, y) \leftrightarrow (1, y)$ and $(x, N) \leftrightarrow (x, 1)$ are neigbours.

  Topology: 2-dim. torus (surface of a donut)

  The boundaries are not special points, the system has translational invariance!

- **Fixed boundaries:** The boundary layers ($x = 1, N$ or $y = 1, N$) are fixed either all to the same value or to some specially chosen values. Strong boundary effects.

- **Free boundaries:** The boundary sites just have less neighbour sites. Strong boundary effects.

- **Twisted periodic boundaries:** For example, the spins $s(N, y)$ and $s(1, y)$ are neighbours, but with "inverted" sign – i.e. the action has opposite sign for these particular links. When $\beta > \beta_c$ the system has magnetization, but because of the "twist" there must be an interface (kink) somewhere in the system. Despite the appearance, this boundary is also translationally invariant: the boundary does not form a special point.

  This is used to study the properties of the interface between states with +1 and -1 magnetization.

10.11 Structure of the Monte Carlo program

Traversal order: in which order should we go through the points on the lattice? In principle this is (largely) up to you, except in some special cases. The common orders are:

- **Typewriter ordering:** go through the sites row-by-row, from left to right. Fast & recommended, for standard uses.

  However: breaks detailed balance for Ising model + Metropolis update! (at least in 1 dimensions...)!!

- **Random ordering:** pick the site at random. Good, but in practice computationally slower than typewriter. Used in some *real-time* calculations.

- **Checkerboard ordering:** divide the sites into black and white sets ($x + y$ even/odd), as in the checkerboard. Since each black site has only white neighbours and vice versa, we can update all of the black sites independently from each other while keeping white sites fixed. Repeat for white sites.

  This must be used in vector computers (Crays, many other older supercomputers) and in parallel programming.

Sample: heat bath for Ising

```
#define NX 32
#define NY 32
...

  int s[NX][NY];
  int sum,x,y;
  double beta,p_plus,p_minus;
  ...

  /* sum over the neighbour sites - typewriter fashion */
  for (x=0; x<NX; x++) for (y=0; y<NY; y++) {
```

```
  sum = s[xup[x]][y] + s[xdn[x]][y]
      + s[x][yup[y]] + s[x][ydn[y]];

  /* Heat bath update - calculate probability of +-1 */
  p_plus  = exp( (beta/2.0) * sum );   /* prob. of +1, unnormalized */
  p_minus = 1.0/p_plus;                 /* and -1 */
  p_plus  = p_plus/(p_plus + p_minus);  /* normalized prob of +1 */

  /* and choose the state appropriately */
  if (mersenne() < p_plus) s[x][y] = 1; else s[x][y] = -1;
}
```

10.12 Some implementation details

I. **Looping over the lattice:**

2 common ways to organize the lattice arrays:

A) 1 variable/dimension:

```
int s[NX][NY];
```

In this case one should loop over y-coordinate in the *inner* loop:

```
for (x=0; x<NX; x++) for (y=0; y<NY; y++)
```

in C, the last index is the "fastest," i.e. these variables are stored in consecutive locations.

In Fortran, this is opposite:

```
integer s(NX,NY)
```

Here the first index (x) is faster.

The speed difference varies a lot depending on the problem/computer.

B) 1 loop variable only:

```
#define VOLUME (NX*NY)
int s[VOLUME];
```

Looping:

```
for (i=0; i<VOLUME; i++) s[i] ...
```

Used a lot in old vector supercomputers – long loops, vectorizes effectively.

II. **Fetching the neighbours:**

Typically (simple) MC programs use a significant amount of time fetching the neighbours of a site (periodic boundaries!). There are several ways to do this:

A) Use modulus to get the neighbours:

$x_{\rm up} = (x+1) \bmod NX$, $x_{\rm down} = (x+NX-1) \bmod NX$

– works in C [$x = 0 + \ldots (NX-1)$], has to be modified a bit in Fortran.

– slow, *unless* $NX = 2^n$ and NX constant (so that the compiler knows what it is).

B) Tabulate the neighbours beforehand:

```
for (x=0; x<NX; x++) {
   xup[x] = (x+1) % NX;
   xdn[x] = (x-1+NX) % NX;
}
```

(same for y-coordinate) and use the tables `xup[]` etc. in the loop. This was used in the example.

– Usually pretty good. If the size is always the same to x- and y-directions, then single up,down -arrays can be used.

– Has to be modified a bit in Fortran:

```
xup(x) = mod(x,NX)+1
```

etc.

C) If single volume index (I.B) is used, then it is usually easiest to use global neighbour pointer arrays, which are again initialized at the beginning:

```
#define ixy(x,y)  ((x+NX)%NX + NX*((y+NY)%NY))
...
  for (i=0; i<VOLUME; i++) {
     x = i % NX;
     y = i / NX;
     xup[i] = ixy(x+1,y);
     xdn[i] = ixy(x-1,y);
     yup[i] = ixy(x,y+1);
     ydn[i] = ixy(x,y-1);
  }
```

Here `xup[]` etc. is an integer array of size `VOLUME`.

– Easy to use in the loops over volume. However, neighbour arrays are large, and memory access can become expensive.

– Again, this style vectorizes easily. It is also used in parallel programming.

III. **Variable type:**

For discrete models (like Ising), it is usually worthwhile to use the smallest easy variable (in this case, (unsigned) `char`).

However, gains depend very much on details.

IV. **Structure of the program:** The program flow of a Monte Carlo simulation program is usually as follows:

1. Initialize what is needed:
   – seed the random numbers
   – initialize the configuration, or
   – load a stored configuration
   – initialize neighbour arrays etc.
2. Update, i.e. do one or more update sweeps through the system. Different algorithms can be used.
3. Measure, and either
   – accumulate the results or
   – write results to a file for post-processing (preferred).
4. Return to 2, until we have the desired amount of measurements.
5. Do whatever maintenance is needed (for example, save the configuration)

10.13 Measurements

The configurations of the model are generated with some algorithm, such as Metropolis. We want to measure numerically thermodynamic quantities of interest, for example (for Ising model)

- Energy density $E = -\frac{1}{V}\sum_{<i,j>} s_i s_j$
- Magnetization $M = \frac{1}{V}\sum_i s_i$
- Correlation function $\Gamma(z) = \frac{1}{V}\sum_i s_i s_{i+z}$
- Correlation length ξ: $\Gamma(z) \sim e^{-z/\xi}$
- Specific heat: $C_V = \frac{1}{V}\frac{\partial E}{\partial T} = \langle E^2\rangle - \langle E\rangle^2$
- Magnetic susceptibility: $\chi_M = \frac{1}{V}\frac{\partial M}{\partial T} = \langle M^2\rangle - \langle M\rangle^2$

Note that the last 2 measurements do not require "new" measurements; these can be calculated directly from measurements E_i, M_i (if these have been stored in a file, for example).

The correlation function requires usually special methodology.

10.14 Phase transitions and critical exponents

Most phase transitions are described by an **order parameter** which is zero in one phase (disordered phase), non-zero in the other phase (ordered phase). Thus, it cannot be an analytic function at the critical point.

- First order — the order parameter (and in general almost any thermodynamical quantity) has a discontinuous jump; i.e. the 1st derivative of the partition function.

  – latent heat, discontinuity in energy

- Second order — the susceptibility or specific heat are divergent (in general, second derivatives of partition function).

Second order transitions are classified by their **critical exponents**, which measure the divergence at the critical point:

$$\begin{aligned} M &\sim |T - T_c|^{\beta} \\ \chi_M &\sim |T - T_c|^{-\gamma} \\ C_V &\sim |T - T_c|^{-\alpha} \\ \xi &\sim |T - T_c|^{-\nu} \end{aligned}$$

For the 2d Ising model, these exponents are $\alpha = 0$, $\beta = 0.125$, $\gamma = 1.75$, $\nu = 1$.

However, on a finite lattice we have finite number of degrees of freedom and *everything* is analytic! Thus, on a finite lattice the order parameter is either always non-zero *or* always zero. Indeed:

$$M = \left\langle \frac{1}{V} \sum_i s_i \right\rangle \equiv 0 \qquad\qquad |M| = \left\langle \left| \frac{1}{V} \sum_i s_i \right| \right\rangle > 0$$

always on a finite lattice! Careful **finite size analysis (FSS)** is needed. (Return to that later)

10.15 Autocorrelations

In sect. 3.5. we already mentioned that the Monte Carlo simulations suffer from autocorrelations: since the update step is a smaller or larger modification of some configuration (Markov chain!), successive configurations and measurements are correlated.

- Let X_i be the measurements of some quantity X from configuration number $i = 1 \ldots N$. At finite N, the error is

  $$\delta X = \sqrt{\frac{\sum_i (X_i - \langle X \rangle)^2}{N(N-1)}}$$

  if and only if the measurements X_i are statistically independent.

- We can define an **autocorrelation function** of quantity X:

$$C(t) = \frac{\frac{1}{N-t}\sum_{i=1}^{N-t} X_i X_{i+t} - \langle X\rangle^2}{\langle X^2\rangle - \langle X\rangle^2} \quad \sim \quad e^{-t/\tau_{\rm exp}}$$

  where the last point holds if $N \to \infty$ and $t \to \infty, t \ll N$.

- The denominator is there to normalize $C(0) = 1$.

- $\tau_{\rm exp}$ is the exponential autocorrelation time. This is in principle (almost) unique; i.e. almost all observables show the unique longest autocorrelation time, which really measures when the configurations become thoroughly uncorrelated.

- However, for error analysis, the relevant quantity is the **integrated autocorrelation time:**

$$\tau_{\rm int} = \frac{1}{2} + \sum_{t=1}^{\infty} C(t)$$

  Note that $\tau_{\rm int} \approx \tau_{\rm exp}$ if the autocorrelation function is purely exponential, $C(t) \approx e^{-t/\tau_{\rm exp}}$. However, usually $\tau_{\rm int} < \tau_{\rm exp}$.

- In Monte Carlo analysis with correlated measurements, the error estimate is

$$\boxed{\text{error of } X \equiv \delta_X = \sqrt{2\tau_{\rm int}}\,\delta_{x,\rm naive} = \sqrt{2\tau_{\rm int}\,\frac{\sum_i (X_i - \langle X\rangle)^2}{N(N-1)}}}$$

  Here $\delta_{X,\rm naive}$ is the naive error shown on previous page.

- Rule of thumb: lag of $2 \times \tau_{\rm int}$ gives new independent configuration. (When there were no autocorrelations, $C(t) = 0$ and $\tau_{\rm int} = 1/2$.)

- How to calculate $\tau_{\rm int}$ in practice? The definitions for $C(t)$ and $\tau_{\rm int}$ above are valid in the limit $N \to \infty$, i.e. infinite statistics. However, in practice finite N modifies the procedure. In practice:

  – We assume that all measurements are written to a file, allowing free post-processing.

  – The optimised autocorrelation function $C(t)$ for lag t now becomes

$$C(t) = \frac{\frac{1}{N-t}\sum_{i=1}^{N-t} X_i X_{i+t} - \langle X\rangle_1 \langle X\rangle_2}{\langle X^2\rangle - \langle X\rangle^2},$$

  where

$$\langle X\rangle_1 = \frac{1}{N-t}\sum_{i=1}^{N-t} X_i \quad \text{and} \quad \langle X\rangle_2 = \frac{1}{N-t}\sum_{i=1}^{N-t} X_{i+t}$$

In the denominator the expectation values can be the usual ones without significant effect.

– Because of finite statistics, $C(t)$ becomes noisy when t is large, $t \gg \tau_{\rm int}$. This causes the summation in $\tau_{\rm int}$ actually not converge. Thus, the summation over the lag t should be cut self-consistently to a value which is larger than $\tau_{\rm int}$ but not by a large factor; for example, to $6 \times \tau_{\rm int}$. Remember that the real signal comes from the integral of function $\sim \exp[-t/\tau]$; thus, by cutting at 6τ the error in $\tau_{\rm int}$ will be less than 0.5%.

  Now the optimised integrated autocorrelation time can be shown to be

$$\tau_{\rm int} = \frac{1}{2} + \sum_{t=1}^{6\tau_{\rm int}} C(t)\,\frac{N-t}{N}.$$

  Here the correction factor $(N-t)/N$ corrects the statistical significance of the data. Clearly, as $N \gg \tau_{\rm int}$, the result goes into the one on page 113.

– Course home page includes a general purpose program `errors.c` which calculates averages and errors (from data in files) using the autocorrelations as described above.

10.16 Error estimates in Monte Carlo measurements

There are 2 commonly used methods for estimating the errors in Monte Carlo measurements:

1. Use

$$\delta_X = \sqrt{2\tau_{\rm int}}\,\delta_{x,\rm naive}$$

   as above.

2. Block the measurements in M **bins** of length m. Calculate averages of observables in each bin, X_k^b, $k = 1 \ldots M$. If the bin length $m \gg \tau_{\rm int}$, the bins are statistically independent, and we can use the naive formula

$$\delta X = \sqrt{\frac{\sum_{i=1}^{M}(X_i^b - \langle X\rangle)^2}{M(M-1)}}$$

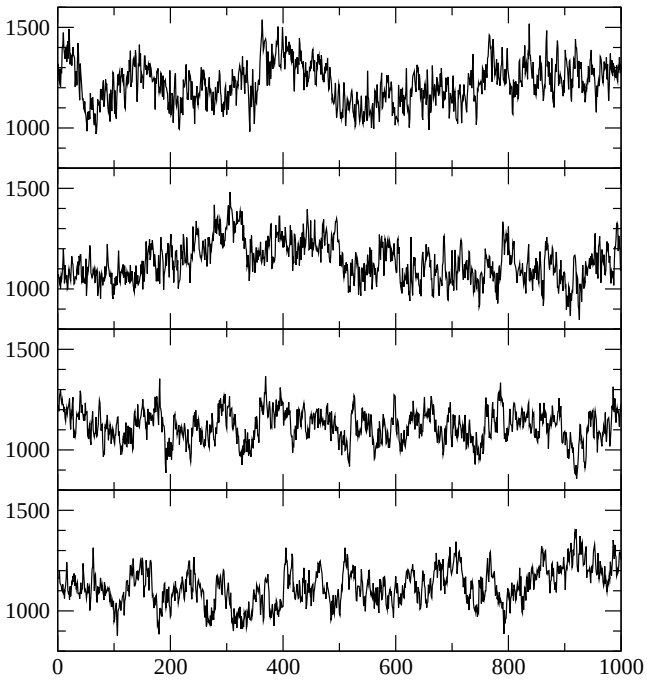
   In practice, one divides the set in variable number of blocks. The error estimate should be $\sim$ constant if the bin length is large enough (sometimes one has to extrapolate to block length $\to \infty$.).

Note that neither the autocorrelation nor the blocking method change the expectation value $\langle X \rangle$ in any way! Just the error bars.

Clearly, in order to minimize the errors, we want to use an update which has as small τ as possible – but measured in CPU-time used, not in iterations!

If we manage to improve $\tau \mapsto \tau/10$, then either
– we need a factor of 10 less configurations for given accuracy
– for a fixed number of configurations, the errors are reduced by a factor of $\sqrt{10} \sim 3$.

10.17 Example: Ising model



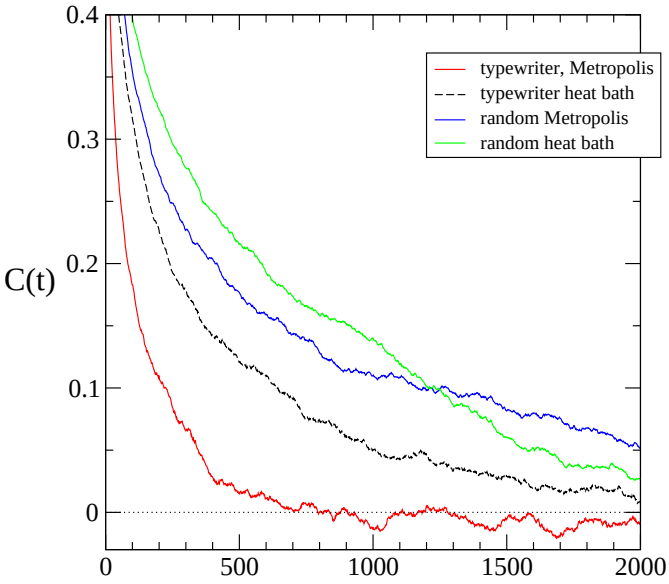
Time histories of the measurements of the total energy, measured from 64^2 Ising model at β_c. Sample of 1000 (from a total of 400000 each).

From top to bottom: Metropolis and heat bath with "typewriter" ordering, then Metropolis and heat bath with random ordering.

It is next to impossible to see by eye which of these algorithms has the shortest autocorrelations.

The autocorrelation function of the total energy from the previous case. Clearly, the Metropolis update with typewriter ordering is the fastest of these algorithms.

However, this is characteristic for the Ising model - in general heat bath tends to be faster than Metropolis!



The *integrated* autocorrelation time τ_{int}, measurement average, and error calculated in various ways are as follows (with 400000 update+measurement cycles for each case):

	τ_{int}	$\langle E \rangle$	δ_{naive}	δ_τ	δ_{100}	δ_{1000}	δ_{10000}
Metro, typew.	54	1182.44	0.17	1.78	1.03	1.76	2.18
HB, typew.	157	1176.45	0.17	3.01	1.18	2.50	3.04
Metro, random	271	1181.29	0.17	3.99	1.24	2.80	3.52
HB, random	316	1180.66	0.17	4.26	1.26	2.97	4.34

The Metropolis with typewriter ordering is the best of the bunch - for fixed # of updates. The quantity δ_{100} means the error using binned naive estimate, the number is the bin size. Clearly, the bin length has to be $\gg \tau$ in order for the method to work!

The real figure of merit is obtained when we compare the time used to the number of configurations:

	time(ms)/iteration	time(ms)/iter. $\times 2\tau_{\mathrm{int}}$
Metro, typew.	1.0	108
HB, typew.	1.2	364
Metro, random	1.4	748
HB, random	1.6	1004

Here the times are in milliseconds. The last column is the real measure of the efficiency of the algorithm (and implementation): how much time is needed to obtain one *independent* configuration ($t \sim 2\tau$).

However, the random ordering is still useful: using random ordering the evolution of the system can be in some cases interpreted as a **real-time** evolution ($\sim$ Glauber dynamics).

10.18 Thermalization

Because of autocorrelations, and because we typically start the simulation from a "bad" configuration (often fully ordered, "cold", or disordered, "hot" configuration), the initial measurements are just wrong. We should get rid of these.

One should discard at least $n \gg \tau$ measurements from the beginning, often values

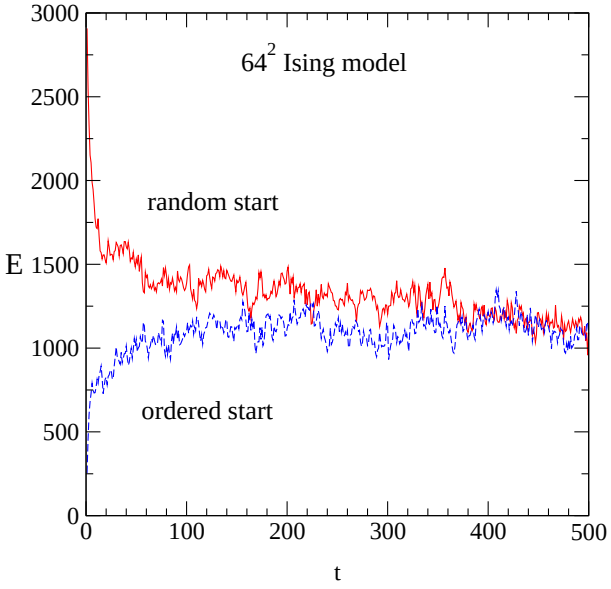
$$n = 5 \ldots 10\,\tau$$

are used. This depends on the problem, though.

Autocorrelation time τ can be very large – if the thermalization time $\lesssim \tau$ and measurement time $\sim \tau$, the results are very much suspect! This can happen in practice in Monte Carlo calculations, because of **critical slowing down**.

In the binning method of error analysis (previous subsection), the data from the initial bin(s) should be discareded.

Hot and cold starts are also often used to estimate the needed thermalization:



This is often actually used to investigate metastable states – for example, in 1st order phase transitions.

10.19 Critical slowing down

- How does the statistical error (and hence, the cost in cpu-time) behave in Monte Carlo simulations?

- We have already established ($N =$ number of sweeps)

$$\delta \sim \frac{1}{(\#\text{ indep. configs})^{1/2}} \sim \sqrt{\tau_{\rm int}/N}.$$

- More precisely, when we consider also the system size, the error behaves as

$$\delta \sim (\tau_{\rm int}/N)^{1/2}(\xi/L)^{d/2}$$

where d is the dimensionality of the system, ξ is the correlation length and L the *linear* system size (volume $= L^d$).

- Why $(\xi/L)^{d/2}$? Now $(L/\xi)^d =$ # of independent ξ-sized domains in the volume. These are, in practice, statistically independent, so that the total # of independent correlation volumes $= N/2\tau \times (L/\xi)^d$.

- The autocorrelation time $\tau \sim \xi^z$, where z is a dynamical exponent which depends on the update algorithm.

- How about the cpu-time cost? Naturally, the time for one sweep is $\propto L^d$. Thus, if we require errors of some definite level δ, the time behaves as

$$\text{time} \sim \tau_{\rm int}\xi^d/\delta^2$$

Surprisingly, if ξ is constant (in lattice units), this is independent of L! Because large volumes help to get rid of finite-size effects, it pays to use as large volumes as practically possible.

- However, the situation is very different when we are at or near a critical point (or continuum limit). In this case ξ diverges, and, in principle, the autocorrelation time

$$\tau \sim \xi^z \sim |T - T_c|^{-z\nu}$$

diverges also at the critical point! This is known as the *critical slowing down*.

- However, on a finite volume, the correlation length is cut-off by the system size $\xi \sim L$ — or, in other words, when the action has no built-in mass/length scale, the scale is set by the infrared cutoff — the system size.

- Thus, at the critical point,

$$\boxed{\tau \sim L^z.}$$

The exponent z is called the *dynamical critical exponent.* The cost of the simulation now behaves as

$$\boxed{\text{cost} \sim L^{d+z}.}$$

instead of being $\sim$ constant![2] Here L^d just reflects the fact that the correlation volume $\sim$ total volume, due to the physics. This we cannot get rid of. However, different update algorithms have different values of z.

- Because interesting physics happens at the phase transition points, and because obtaining meaningful results at a critical point requires finite-size scaling (FSS) scaling analysis, it is important to use algorithms with as small z as possible.

- Common stochastic algorithms — heat bath, Metropolis — have $z \approx 2$: the nearest-neighbour update is a stochastic process, so that the signal propagates over a distance ξ in time $\propto \xi^2$ (diffusion, random walk).

  More precisely, it has been argued that for stochastic algorithms $z \geq \gamma/\nu$.

- Some **non-stochastic** update algorithms may have $z = 1$, or the signal propagates over a distance ξ in time $\propto \xi$. Physically, this means that the update algorithms support some kind of **wave motion** instead of diffusion.

  Examples: Overrelaxation in many cases has $z \approx 1$; also evolving the lattice fields using equations of motion. However, if the process involves randomness, $z \approx 2$ at the limit $L \to \infty$.

  Thus, for many practical purposes, a good update is a mixture of overrelaxation and Metropolis/heat bath.

- $z \ll 1$, and even $z = 0$ can be achieved in some cases using **non-local update algorithms** (cluster, multigrid ...).

  Cluster algorithms are the most succesful non-local update methods.

[2] In reality, the cost depends quite a lot on the observable: some observables have non-trivial volume dependence at a critical point.

10.20 Common (and subtle) mistakes in update algorithms

- **Measure factors:** Note that the detailed balance condition is correctly written in terms of probabilities:

$$\frac{W_f(\phi \mapsto \phi')}{W_f(\phi' \mapsto \phi)} = \frac{p_{\text{eq.}}(\phi')}{p_{\text{eq.}}(\phi)} = e^{(H(\phi)-H(\phi'))/k_BT}$$

  The second equality is true only if $p(\phi) \propto e^{-H(\phi)/k_BT}$. We have implicitly assumed this for most of the discussion in this section, but this is not always the case!

  As a simple example consider variables (x, y), with distribution $p(x, y) \propto e^{-H(x^2+y^2)/k_BT}$. Now, one might want to use polar coordinates (r, θ) instead. Now

$$p(x, y)\, dx\, dy = p(x, y)\, r\, dr\, d\theta \equiv p(r, \theta)\, dr\, d\theta \Rightarrow p(r, \theta) \propto re^{-H(r^2)/k_BT}$$

  The Jacobian factor r follows the Boltzmann factor everywhere.

  Thus, for example, if we do restricted Metropolis update

$$r \to r' = r + S(X - 0.5)$$

  where X is a uniform random number in interval [0,1], this is accepted with the probability

$$p_{\text{accept}}(r \mapsto r') = \min\left(1, \frac{r'e^{-H(r'^2)/k_BT}}{re^{-H(r^2)/k_BT}}\right)$$

  It is important to keep track of the correct measure (Jacobian) factors!

  Excercise: how would you update a 3-dim. vector $\vec{v}$ which is restricted to unit length (i.e. traces a surface of 2-sphere) and which has interaction energy $H = -\vec{v} \cdot \vec{c}$, with some constant vector $\vec{c}$? The measure is assumed homogeneous on the 2-sphere.

- **Adjustable Metropolis scale:** Metropolis updates have adjustable scale factor. This can be automatically tuned by the update algorithm to acceptance $\sim 60\%$, for example. However, this tuning must not happen during measurements, or it can ruin the detailed balance. Thus, automatic tuning should be done only in the "thermalisation" phase.

- **Other tunables in updates/measurements:** Like the Metropolis scale, adjusting some other tunables (e.g. the measurement interval) on the fly lead to incorrect sampling. These have to be done before the measurements are taken.