Computer Simulations

WS 2020

H. G. Evertz

Based on earlier lecture notes

by W. von der Linden and E. Schachinger

Contents

Preface

Traditionally, physics has been divided into two fields of activities: theoretical and experimental. Due to the development of powerful algorithms and the stunning increase of computer power, each of which have resulted in improvements by many orders of magnitude, a new branch of physics has established itself: Computational Physics. Computer simulations play an increasingly important role in physics and other sciences, as well as in industrial applications. They involve different strategies and serve different kinds of purposes, among them

- Direct solution of complex mathematical problems, e.g.
  - Eigenvalue problems
  - Minimization / Optimization
  - Differential Equations (Finite Element Methods)
- Stochastic methods for very high dimensional problems, especially by Markov Chain Monte Carlo (importance sampling from a distribution), for e.g.
  - Statistical Physics
  - Percolation problems like oil search, forest fires, avalanches
  - Minimization / Optimization
  - Quantum Mechanics
- Simulation of time evolutions in high dimensional spaces, e.g.
  - Classical physics, using Molecular Dynamics methods
  - Quantum Mechanics, using special Hilbert space methods or mappings to Molecular Dynamics
- Newly evolved applications, e.g.
  - Machine Learning, especially with Neural networks (so-called Deep Learning)
  - Abstract dynamical models like cellular automata for forest fires, traffic flow, avalanches

Computer simulations have become essential tools, enabling the investigation and the understanding of complex problems which are inaccessible to analytical methods. Amazingly, they have also led to the discovery of pervasive phenomena in nature ,like fractal structures, which had previously escaped recognition, and to completely new insight by the development of abstract dynamical models like cellular automata for diverse topics like forest fires, traffic flow, or avalanches.

From the multitude of strategies and applications of computer simulations, the present lectures, with a restricted available time of only one hour per week, aim to serve as an introduction to some of the most important aspects and techniques.

There are several additional important goals addressed thoughout these lectures, beyond the introduction of specific techniques and applications. One important aspect is **efficiency** in terms of scaling of effort with problem size. This does *not* concern details of a method or its implementation, but instead the much more important issue of whether a method dealing with N variables takes a computational effort of, say, $O(N)$ or $O(N^2)$ or even more. For typical problem sizes like $N = 10^6$ or 10^9 the difference between $O(N)$ and $O(N^2)$ is giant. This question of scaling applies both to the algorithm used and indeed to its implementation. It is easy to introduce bad scaling and thus gross inefficiencies in a program by accident (especially when using abstract vector or matrix operations like in Matlab), and in small testss cases one can overlook the problem. When big system sizes are eventually intended, one *must* make sure that the actual computational effort scales correctly.

The most important overarching goal is on how to produce **reliable results**. It is tempting to treat numerical methods (as well as other machinery) as black boxes. One presses some buttons, the machine runs, and out come the results. But are they reliable? Are they even remotely true?? It is very easy to produce nonsense, even with a formally correct program and a lot of computing time. Therefore, it is *essential to proceed very carefully*, starting from very easy cases with known results, for which one *must verify* the program and its results. Then one *must* proceed very slowly to more difficult cases, e.g. bigger simulated systems, and check for signs of difficulties. Usually one *does not know the limitations* of one's procedures in advance, and easily oversteps them. The catchword is to **think small**. In everyday life this is obvious, for example for mountain climbing: if one would proceed from a first climbing course inside a building directly to the Himalayas, the result would be desaster. But different from falling off a mountain, the failure of an algorithm or program may be completely hidden. There will almost always be results, but they may be entirely wrong. The only safe way to proceed is to start with easy cases and *slowly* increase the difficulty. Note that this careful procedure usually does not take much extra time, since it is the biggest sensible system which will in the end dominate the total computational effort.

Finally, an essential part of every quantitative result is a **reliable estimate of the uncertainties** (often called “error estimate”). A numerical result *must* always be accompanied by a sensible error estimate. **Stating a numerical result without providing a reliable error estimate is useless.** Related issues often arise in every-day life. In an unfortunate sense, every quantitative statement already contains an error estimate of sorts, namely by the number of digits specified. For example, it is common to see numbers like 34.21% as the result of some poll, without any explicitely specified error estimate. This seems to imply that the last digit "1" is still correct or almost correct. But is this true ? The actual uncertainty in polls is usually two orders of magnitude larger. If one takes all the digits to be true, then one is led, often in practice, to very wrong conclusions, like the supposed gains or losses of a political party, with ensuing political consequences – based on a wrong interpretation of data that was specified without uncertainties. A discussion of such topics, including the related question of errors of histograms, will be provided in an appendix to these lecture notes.

In computational physics (and elsewhere), the control of uncertainties, and hopefully a good quantitative estimate, are one of the major tasks, and can sometimes even dominate the effort invested. But, to emphasize, without the error estimate the results would be meaningless. The present lecture notes and the accompanying excercises will attempt to provide some guidance.

Additional information and other topics can be found in the book *Basic Concepts in Computational Physics* (Springer, 2nd edition 2016) by B. Stickler and E. Schachinger, with which these lecture notes share a common ancestry.

Chapter 1

Sampling from probability distributions

Literature

1. F.J. VESELY, *Computational Physics: An Introduction*, Springer 2001.
2. W. KINZEL and G. REENTS, *Physik per Computer. Programmierung physikalischer Probleme mit Mathematica und C*, Springer 1996 (english: Springer 1997).
3. K. BINDER and D.W. HEERMANN, *Monte Carlo Simulation in Statistical Physics: An Introduction*, Springer 2002.
4. D.P. LANDAU and K. BINDER, *A Guide to Monte Carlo Simulations in Statistical Physics*, Cambridge University Press 2009.
5. B.A. BERG, *Markov Chain Monte Carlo Simulations and their Statistical Analysis*, World Scientific 2004.
6. M.E.J. NEWMAN and G.T. BARKEMA, *Monte Carlo Methods in Statistical Physics*, Oxford University Press 1999.
7. J.S. LIU, *Monte Carlo Strategies in Scientific Computing*, Springer 2008 (mathematically oriented).

1.1 Introduction

Physical systems, like a gas, a fluid, or a more complicated system, can occur in many states x. It is often useful to describe such a system by its (equilibrium) **statistical** properties, namely the probability distribution $\pi(x)$, for example the Boltzmann distribution in statistical physics. Quantum mechanical systems in d dimensions can often also be mapped to such systems, which formally look like a statistical system in d+1 dimensions, where the extra dimension corresponds to time.

Many physical quantities (including dynamical response functions in the case of quantum mechanical systems) can then be expressed as an **expectation value** of some function f,

$$\langle f \rangle \; := \; \sum_x \, f(x)\, \pi(x) \; , \tag{1.1}$$

where f is for example the energy, or a correlation, or any other quantity. We shall *denote the expectation value by angular brackets*, like in quantum mechanics.

The state space of physical systems is usually huge. For example, imagine a crystal of some atoms, for which we just examine the spin direction of the outermost electrons, i.e., spin $\frac{1}{2}$ degrees of freedom. In the simplest case, we treat the system classically, with spin values $\pm\frac{\hbar}{2}$ for each electron, and some Boltzmann-like probability distribution π for each configuration x of the spins. (This leads to the so-called Ising model, discussed later in this chapter). Then for n atoms, there are 2^n possible spin configurations, i.e. exponentially many, so that for large n one *cannot* sum over all of them to calculate the partition function or expectation values like $\langle f \rangle$. Precise or even just useful analytic calculations are often not possible; thus such problems are very difficult to treat directly.

However, if we find a way (called **importance sampling**) to draw states x_i *directly from the distribution* π then we can calculate the **sample mean (average value)** of N such states, *denoted by a bar in these lecture notes*

$$\overline{f} \; := \; \frac{1}{N} \sum_{i=1}^{N} f(x_i) \; , \tag{1.2}$$

One needs to strictly distinguish between the expectation value $\langle f \rangle$ with respect to a probability distribution, and a sample mean $\overline{f}$.[1] The sample mean obviously depends on the sample taken and is a random variable itself, with fluctuations across different samples. In the limit $N \to \infty$, $\overline{f}$ approaches $\langle f \rangle$. Note that specifying this sample mean (like any quantitative result) is *meaningless* without an estimate of its error !

The variance of f with respect to $\pi(x)$ is defined as an expectation value

$$\sigma^2 = \Big\langle \; (\, f(x) - \langle f(x) \rangle \,)^2 \; \Big\rangle \; . \tag{1.3}$$

Note that this variance refers to the distribution of the values which f can take. (A very simple example would be a distribution π of values ± 1 with equal probability, thus an expectation value of 0 and a variance of 1.)

The variance can be estimated by the **sample variance**

$$s^2 \; := \; \frac{1}{N-1} \; \sum_{i=1}^{N} \, \big(f(x_i) - \overline{f} \big)^2 \, . \tag{1.4}$$

[1] In the present lecture notes, we will adhere to these names and notations, but note that they may differ elsewhere in the literature.

If the states x_i are drawn *independently* of each other, then the error of the sample *mean* $\overline{f}$ for large N can be estimated as

$$\sigma_{\widetilde{f}} \;\simeq\; \sqrt{s^2/N}. \qquad (1.5)$$

It falls off like $1/\sqrt{N}$. (In the simple example above, where f has a variance of 1, it would just be $1/\sqrt{N}$). If the states x_i are not independent, then the error can be much larger, as we will discuss later (but it will usually still fall off like $1/\sqrt{N}$).

It is therefore highly desirable to develop and employ **numerical methods for drawing states from a given distribution** π, where π might be complicated and very high dimensional.

For low dimensional state spaces, there are algorithms, so-called random number generators, to draw from some given probability distribution "directly". We will soon introduce some important simple methods to do this. They are usually based on some basic generator for uniformly distributed numbers.

For high dimensional state spaces where such direct methods become extremely inefficient, so called **Markov Chain Monte Carlo** methods have been developed. By a stochastic process, they generate a sequence of configurations x_i which together sample from the full ensemble according to some desired distribution $\pi(x)$. We will treat such methods in some detail in this chapter. As a basic ingredient, these methods use the simpler random number generators just mentioned.

Monte Carlo methods have even been constructed for quantum mechanical systems. They make use of the Feynman path integral formulation to map a d-dimensional quantum mechanical system to a classically looking system in $(d + 1)$ dimensional space-time, with a distribution π related to the path integral. Such **Quantum Monte Carlo** simulations also provide information about spectra and other time dependent quantities. They are however beyond the scope of the present lectures.

1.2 Direct Sampling Methods

In this section we describe different techniques to draw random numbers from a distribution $g(x)$, independent of each other. We treat the single-variable case, since random sampling from multivariate distributions can always be reduced to single variable sampling. In fact all other generators (including Monte Carlo) are built on these simple direct methods.

We are thus concerned with a *random variable* X. It is characterized by a (possibly infinite) interval $x_{\mathrm{m}in} \leq x \leq x_{\mathrm{m}ax}$ or a discrete set of x-values, which can be viewed as the outcome of "experiments", and a Cumulative Distribution Function (**CDF**) $G(\lambda)$ which specifies the probability that in a particular realization, i.e. one instance of the "experiment", the outcome satisfies $x \leq \lambda$. It is connected by

$$G(\lambda) \;=\; \int_{x_{\mathrm{m}in}}^{\lambda} dx\; g(x) \tag{1.6}$$

to the probability *density* $g(x) \geq 0$, also called the *Probability Distribution Function* (**PDF**). It is normalized ($G(x_{max}) = 1$). We want to generate realizations of such a random variable, i.e. to generate random values of x, distributed in the interval $[x_{\mathrm{min}}, x_{\mathrm{max}}]$ with density $g(x)$.

Note that with a computer program we can never generate truly random numbers. Instead we produce **"pseudorandom"** numbers, usually in sequential fashion, which should satisfy certain criteria, but will always be inadequate in some aspects. Some of the simpler criteria are statistical means, variances, etc., as well as visible correlations of subsequent numbers. Note that the actual sequence generated by some generator can be modified by specifying a parameter, the so-called *seed-number*.

It is important to realize that the ″**quality**″ of any given random number generator **depends on the application** for which it is used. A generator which is very good for one application may fail badly for others, even closely related ones. One should therefore compare results employing several random number generators of *different* design, especially when precise results are desired.

1.2.1 Uniform Random Numbers

Most random sampling algorithms have as their *basic ingredient* random numbers r which are uniformly distributed in the interval $[0, 1]$, described by the PDF:

$$u(r) = \begin{cases} 1 & r \in [0,1] \\ 0 & \text{otherwise.} \end{cases} \tag{1.7}$$

This results in the cumulative distribution function

$$U(r) = \int_0^r dr'\, u(r') = \begin{cases} 0 & r < 0 \\ r & 0 \leq r < 1 \\ 1 & r \geq 1. \end{cases} \tag{1.8}$$

There are many algorithms. Among the simplest are the *linear congruential generators* (LCG) which are based on the recursion

$$x_{i+1} = (a\,x_i + c) \bmod m,$$

where the integers a, c, and m are constants. These generators can further be classified into mixed $(c > 0)$ and multiplicative $(c = 0)$ types. An LCG generates a sequence of pseudo random integers X_1, X_2,... between 0 and $m-1$. Each X_i is then scaled into the interval $(0,1)$ (or sometimes $[0,1)$). When m is a prime number, the period of the generator can be up to $m-1$. It depends on the choice of parameters a, c.

One choice of parameters for the LCG sometimes used is $a = 16\,807 = 7^5$ and $m = 2^{31} - 1$. This yields a generator (called GGL or CONG or RAND0) which is given by:

$$R_n = \left(7^5\,R_{n-1}\right) \bmod \left(2^{31} - 1\right), \quad r_n = \frac{R_n}{2^{31} - 1}.$$

It produces a sequence of random numbers r_n uniformly distributed in $(0,1)$ from a given initial **seed number** $R_0 < 2^{31} - 1$, which for this generator needs to be an odd number. This generator is known to have reasonably good random properties for many purposes. The sequence is periodic, with a period of the order of $2^{31} \approx 2 \times 10^9$. This is not large enough to prevent recurrence of random numbers in large simulations (which may or may not matter). *Caution:* the least significant bits of this and most other generators have far worse properties than the most significant ones; they should thus not be used by themselves.

Another known problem of this and other linear congruential generators is that D-dimensional vectors $(x_1, x_2, \ldots, x_D)$, $(x_{D+1}, x_{D+2}, \ldots, x_{2D})$, ... formed by consecutive normalized random numbers x_i may lie on a relatively small number of parallel hyperplanes.

Operating systems and programming languages usually provide random numbers, sometimes like (1.2.1), sometimes of other designs. In many cases they are good enough. But, again, it depends on the precise problem and method used whether a given generator is "good" or "bad".

1.2.2 Inverse Transformation Method

We want to draw from a probability distribution function $g(x)$. The cumulative distribution function

$$G(x) = \int\limits_{x_{\min}}^{x} dx'\; g(x')$$

is a non-decreasing function of x. Note that $G(x)$ is constant (i.e. does not increase) in regions of x where $g(x) = 0$. When X is a random number

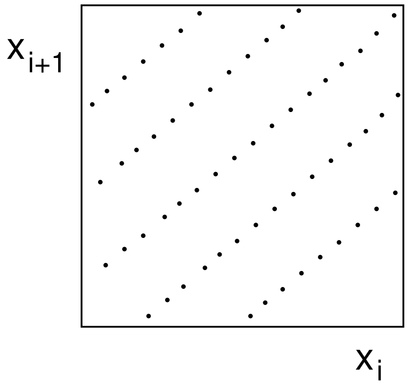


Figure 1.1: Bad generator (schematical): Pairs of random numbers lying on hyperplanes.

drawn from g, then the above definition of $G(x)$ is equivalent to specifying the probability

$$P(X < x) = G(x) \ . \tag{1.9}$$

For uniform random numbers u this relation becomes

$$P(u < x) = x \ . \tag{1.10}$$

We can define an inverse of the cumulative distribution function:

$$G^{-1}(\xi) \ := \ \inf\{x|G(x) = \xi\} \ . \tag{1.11}$$

We need the infimum (or some similar function) here because $G(x)$ can be constant in some intervals.

Because of the monotonicity of $G(x)$, the function G^{-1} obeys

$$G^{-1}(\xi) \leq x' \quad \Leftrightarrow \quad \xi \leq G(x') \ . \tag{1.12}$$

Now *let us draw ξ from a uniform distribution between 0 and 1,* i.e. for ξ we use a random number u. Then (1.12) implies

$$P(G^{-1}(u) \leq x') \ = \ P(u \leq G(x')) \ = \ G(x') \ ,$$

where the last equality follows from (1.10). Thus $G^{-1}(u)$ satisfies (1.9) and it is therefore a random number distributed like g, as desired.

To repeat, if ξ is chosen as a uniformly distributed random number, then the variable defined by

$$x = G^{-1}(\xi) \ := \ \inf\{x|G(x) = \xi\}$$

is randomly distributed in the interval $[x_{\min}, x_{\max}]$ with the desired PDF $g(x)$. The randomness of x is guaranteed by that of ξ. Eq. (1.11) is equivalent to solving

$$\xi = \int\limits_{\mathbf{x}_{\min}}^{x} dx' \, g(x'), \tag{1.13}$$

for (the smallest possible) x. This is called the sampling equation. The inverse transformation method is **useful if** $g(x)$ is given by a simple analytical expressions such that **the inverse of** $G(x)$ **can be calculated easily**.

Example. Consider the uniform distribution in the interval $[a, b]$:

$$g(x) = u_{a,b}(x) = \frac{1}{b-a}.$$

We need to solve

$$\xi = G(x) = \frac{x-a}{b-a}$$

for x, namely

$$x = a + \xi(b-a), \quad \xi \in [0,1].$$

As another important example consider the exponential distribution

$$g(s) = \frac{1}{\lambda} \exp\left\{-\frac{s}{\lambda}\right\}, \quad s > 0,\ \lambda > 0, \tag{1.14}$$

which describes, e.g., the free path s of a particle between interaction events. The parameter λ represents the mean free path. In this case, the sampling equation (1.13) is easily solved to give the sampling formula

$$s = -\lambda \ln(1-\xi) \ \text{ or } \ s = -\lambda \ln \xi.$$

The last equality follows from the fact that $1 - \xi$ is also a random number distributed equally in $(0, 1]$.

1.2.3 Rejection Method

The inverse transformation method for random sampling is based on a one-to-one correspondence between x and ξ values. There is another kind of sampling method, due to von Neumann, that consists of sampling a random variable from a suitable other distribution and subjecting it to a random test to determine whether it will be accepted for use or rejected. This rejection method leads to very general techniques for sampling from *any* PDF.

We want to generate random numbers following a PDF $g(x) \geq 0$

$$\int_a^b dx\, g(x) = 1, \quad x \in [a,b].$$

We need to find another PDF $h(x) \geq 0$,

$$\int_a^b dx\, h(x) = 1,$$

which should satisfy

$$g(x) \leq c\,h(x), \quad \forall x \in [a,b], \tag{1.15}$$

with c a constant. Thus, in the interval $[a,b]$, $c\,h(x)$ is an *envelope* of the PDF $g(x)$. (See Fig. 1.2.)

The interval $[a,b]$ can also be infinite, in which case $h(x)$ cannot be a constant because of normalization. *The distribution $h(x)$ needs to be easy to sample.* In practice this means that we should be able to compute the inverse $H^{-1}(\xi)$ of the CDF $H(x) = \int_a^b h(x)dx$. In Fig. 1.2, $h(x)$ is composed of two parts, the first (e.g. $\sim 1/x^2$) taken to envelope $g(x)$ at large x, and the second constant part at small x in order to avoid the large area underneath a function like $1/x^2$ at small x (see below). The corresponding CDF $H(x)$ can still be inverted easily in an analytical way.

Random numbers distributed according to the PDF $g(x)$ are then generated according to the following procedure:

Algorithm 1 Rejection Method

```
begin:
Generate a trial random variable $x_T$ from $h(x)$
Generate a uniform random number $r$ from $u(x)$, Eq. (1.7).
if $r\,c\,h(x_T) < g(x_T)$ then
    accept $x_T$
else
    go to begin
endif
```

Proof: The probability of accepting a value x_T is $p(A|x_T\,\mathcal{B}) = g(x_T)/(ch(x_T))$ (with $p := 0$ when $h = g = 0$). Thus

$$\begin{aligned} p(x|\mathcal{B}) &\propto p(x = x_T|\mathcal{B})\, p(A|x_T\mathcal{B}) \\ &= h(x_T)\frac{g(x_T)}{c\,h(x_T)} \\ &\propto g(x_T). \end{aligned}$$

The accepted random numbers x_T indeed follow the PDF $g(x)$.

Probability of acceptance:

The overall probability of acceptance $P(A|\mathcal{B})$ is simply the area under $g(x)$

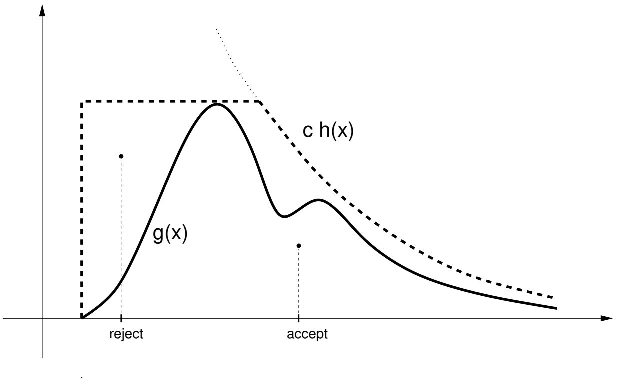


Figure 1.2: The rejection method.

divided by the area under $ch(x)$, i.e. it is $\frac{1}{c}$. Formally:

$$\begin{aligned}
P(A|\mathcal{B}) &= \int dx_T\, p(Ax_T|\mathcal{B}) \\
&= \int dx_T\, p(A|x_T\mathcal{B})p(x_T|\mathcal{B}) \\
&= \int dx_T\, \frac{g(x_T)}{c\,h(x_T)}h(x_T) \\
&= \int dx_T \frac{g(x_T)}{c} \\
&= \frac{1}{c}.
\end{aligned}$$

Thus, the bigger c, the worse the probability of acceptance becomes. We should therefore choose the constant c reasonably small, i.e. $c\,h(x)$ should be reasonably close to $g(x)$. There is usually no need to optimize c much, since the effect on overall computation time of some calculation in which the random numbers are used is usually negligible.

However, if we apply a rejection method for a probability distribution π in d dimensions then we will usually get an exponential behaviour roughly like

$$P(A|\mathcal{B}) \simeq \left(\frac{1}{c}\right)^d$$

(unless the dimensions are independent such that we can draw independent random numbers for each direction). The rejection method is therefore *very inefficient in high dimensions*. Markov Chain Monte Carlo is then often the method of choice.

1.2.4 Probability Mixing

This is a simple approach to cases in which the PDF $f(x)$ is a sum of several PDFs in an interval $[a, b]$:

$$f(x) = \sum_{i=1}^{N} \alpha_i \; f_i(x), \tag{1.16}$$

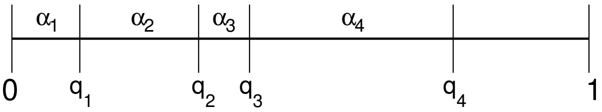
with

$$\alpha_i \geq 0, \; f_i(x) \geq 0, \qquad \int_a^b dx \, f_i(x) = 1, \text{ and } \sum_{i=1}^{N} \alpha_i = 1.$$

We define the partial sums

$$q_i = \sum_{j=1}^{i} \alpha_j \; .$$

Thus $q_n = 1$ and the interval $[0, 1]$ has been divided according to this figure:



Now we choose an equally distributed random number $r \in [0, 1]$ and determine the index i for which the condition

$$q_{i-1} < r < q_i$$

is fulfilled. Then we draw a random number according to $f_i(x)$. This procedure is correct because α_i specifies the importance of the PDF $f_i(x)$. This in turn gives us the probability that the random variable X is to be sampled using the PDF $f_i(x)$.

Example: The function $h(x)$ in Fig. 1.2 is composed of two parts. We can draw from $h(x)$ by using separate generators for each of the parts.

1.3 Markov Chain Monte Carlo: Importance Sampling

Markov Chain Monte Carlo (MCMC) is an efficient approach to perform sampling in many dimensions, where often the probability density $\pi(x)$ is strongly dominated by only a small part of the total state space. Prime examples are physical systems consisting of many similar particles (electrons, atoms, ...) which interact with each other. The dimension d of state space is some multiple of the number of particles, typically $d = O(10^3 \ldots 10^7)$ and larger, so that any sort of simple sampling is entirely impossible. In this lecture we will consider an especially simple model, the Ising model as an example.

In Markov Chain Monte Carlo, configurations x are created iteratively in such a way that their distribution corresponds to a desired distribution $\pi(x)$. Because of the iterative construction, consecutive configurations are usually highly correlated. Thus the samples in average values (1.2) consist of numbers which are also correlated. This is entirely permissible. It does however require careful consideration of the convergence of sample means and especially of their statistical errors.

Many methods have been developed over time for different applications, some of them very efficient. We will be concernced in details just with the basic "Metropolis-Hastings" method, which is almost always applicable. An appendix provides an overview over more modern strategies. In physics, Markov Chain Monte Carlo is often just called "Monte Carlo".[2]

1.3.1 Markov Chains

Chains of dependent events were first studied by A. Markov in 1906. We will first collect a number of definitions and properties.

Markov property

A Markov chain is specified in a discrete sequence, usually called (Markov- or Monte-Carlo-) time. It does normally *not* correspond to physical time. The Markov "time" is just $t_n = n = 0, 1, 2, \ldots$. At each of these times the system is in one state x_{t_n} of the set of possible states

$$Z_1,\, Z_2,\, \ldots,\, Z_k,$$

which together span the state space $\mathcal{S}$. Example: $(x_{t_1}\!=\!Z_9,\; x_{t_2}\!=\!Z_5,\; x_{t_3}\!=\!Z_8,\; \ldots)$. We shall restrict our discussion to **finite state spaces**.[3]

[2] The name "Monte Carlo integration" is sometimes used for *simple* sampling with a random choice of values of the integration variable x. This should not be confused with the Importance Sampling of Markov Chain Monte Carlo.

[3] This avoids cumbersome mathematical issues. Indeed, it is realistic, since the representation of a system on a computer using a finite number of bits for the discrete or "continuous" degrees of freedom corresponds to a finite state space.

The defining **Markov property** is that the probability of being in a particular state Z_j at time t_{n+1} depends *only* on the state at the *current* time t_n, but not on any previous history.

Transition matrix

The Markov chain is specified by the initial probability distribution $\pi(x, t = 0)$ of states at time zero, and by the transition probability p_{ij} **from state** Z_i at time t_n **to state** Z_j at time t_{n+1}. *We demand that the transition probabilities do not depend on time.* Transition probabilities are organized in the *transition matrix*:

$$\boldsymbol{P} = \{p_{ij}\} = \begin{pmatrix} p_{11} & p_{12} & \cdots & p_{1k} \\ \vdots & \vdots & \ddots & \vdots \\ p_{k1} & p_{k2} & \cdots & p_{kk} \end{pmatrix} . \tag{1.17}$$

$\boldsymbol{P}$ is a *stochastic matrix, i.e.:* all elements of $\boldsymbol{P}$ obey the inequality $0 \leq p_{ij} \leq 1$, and all row-sums of the $k \times k$ matrix are equal to unity:

$$\sum_{j=1}^{k} p_{ij} = 1, \qquad i = 1, \ldots, k. \tag{1.18}$$

The transition probabilities $p_{ij}^{(k)}$ from state Z_i at time t_n to a state Z_j at time t_{n+k} k steps later are simply $(\boldsymbol{P}^k)_{ij}$.

Irreducibility

A Markov chain is called *irreducible*, if for every pair of states Z_i and Z_j there is a finite number n, such that one can get get from Z_i to Z_j in n steps with finite probability, i.e. if the state space is not subdivided into non-communicating regions. Note that in order to draw samples from some distribution π, we only need the chain to be irreducible for the space of all states for which π_i is larger than zero.

Non-Periodicity

A state Z_i has period d, if any return to state Z_i occurs only in some multiple of d time steps and d is the largest number with this property. A state is *aperiodic* when $d = 1$. A sufficient condition for aperiodicity of state Z_i is that the diagonal matrix element is not zero: $p_{ii} > 0$. In an irreducible chain, all states have the same period; so we can then speak of the chain having the period d.[4]

Example: A transition matrix in a space of 2 states, which is irreducible but

[4] For an irreducible chain, non-periodicity is already ensured when $p_{ii} > 0$ for at least one state.

not aperiodic:

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} .$$

With this matrix one can move from any state to any other state, so the chain is irreducible. But it is also periodic with period 2, since only an even number of steps will lead back to the original state.

Regular chain

The transition matrix $\boldsymbol{P}$ is called *regular* if some power of $\boldsymbol{P}$ has only strictly positive elements. This is equivalent to the chain being both irreducible and aperiodic.

Ergodicity

A Markov chain with the same properties, irreducibility and aperiodicity, is often called *ergodic*.
Caution: Ergodicity is sometimes defined differently.

1.3.2 Stationarity and convergence

Stationarity, Global balance

The vector of probabilites π_i, with $\sum_i \pi_i = 1$, is called a **stationary distribution** when

$$\boxed{\pi_j \;=\; \sum_i \pi_i \, p_{ij}} \tag{1.19}$$

This means that when one starts with an *ensemble* of states occuring with probabilities π_i and performs a step of the Markov chain, then the resulting ensemble will have the same probabilities for each state.

One can write the equation (1.19) for stationarity (= ”Global Balance”) in matrix form:

$$\boldsymbol{\pi} \;=\; \boldsymbol{\pi}\, \boldsymbol{P} \,,$$

showing that π is a *left eigenvector* of the transition matrix $\boldsymbol{P}$, and it is a *fixed point* of repeated applications of $\boldsymbol{P}$.

Convergence of a Markov chain

Theorem 1.1 *A Markov chain has a stationary* distribution *if it is irreducible. Then the stationary distribution π is* unique.

Theorem 1.2 *If the chain is also* aperiodic*, then it converges to the stationary distribution,* independent of the starting distribution.

$$\boxed{\lim_{n\to\infty} (\boldsymbol{P}^n)_{ij} \;=\; \pi_j \qquad \forall i \,.} \tag{1.20}$$

This is a central theorem for Markov chains. Equation (1.20) is equivalent to

$$\lim_{n\to\infty} \boldsymbol{P}^n \;=\; \boldsymbol{1\pi}\;, \tag{1.21}$$

i.e., $\boldsymbol{P}^n$ converges to a matrix in which every row is the distribution π.

1.3.3 Detailed balance

A Markov chain is called *reversible* if it satisfies the *detailed balance* condition[5]

$$\boxed{\pi_i\, p_{ij} \;=\; \pi_j\, p_{ji}} \tag{1.22}$$

This is a sufficient condition for stationarity. Proof: We take the sum $\sum_i$ on both sides of eq. (1.22). Because of $\sum_i p_{ji} \;=\; 1$, we get eq. (1.19) immediately.

Example: Spread of a rumor

Z_1 and Z_2 are two versions of a report, namely, Z_1: Mr. X is going to resign, and Z_2: Mr. X is not going to resign. We can write the following transition matrix:

$$\boldsymbol{P} = \begin{pmatrix} 1-p & p \\ q & 1-q \end{pmatrix},$$

to the effect that:

(i) Some person receives the report Z_1. It will then pass this report on as Z_2 with a probability p and as Z_1 with a probability $(1-p)$. Consequently, the report will be modified with a probability p.

(ii) In the same way, the report Z_2 will be modified with a probability q.

Realistically $0 < p < 1$ and $0 < q < 1$. In this simple case we can analyze the Markov chain exactly, namely by diagonalizing $\boldsymbol{P}$ such that $U^\dagger \boldsymbol{P} U$ is diagonal. Then $\boldsymbol{P}^n = U\,\left(U^\dagger \boldsymbol{P} U\right)^n\, U^\dagger$, and one obtains

$$\lim_{n\to\infty} \boldsymbol{P}^n = \frac{1}{p+q}\begin{pmatrix} q & p \\ q & p \end{pmatrix}$$

and, consequently

$$\pi_1 = \frac{q}{p+q}, \qquad \pi_2 = \frac{p}{p+q}.$$

Indeed, this distribution also satisfies the detailed balance condition, namely

$$\frac{p_{12}}{p_{21}} \;=\; \frac{\pi_2}{\pi_1}\,.$$

[5] The same equation occurs in chemistry for concentrations and reaction rates.

No matter what the initial probabilities of the reports Z_1 and Z_2 were, they will in the end converge to π_1 and π_2. Thus the public will eventually be of the opinion that Mr. X will resign with a probability π_1 that is independent of his real intentions.

Markov chains have applications in many diverse fields, among them statistical physics, evolution of populations, Bayesian analysis of experiments, and many others.

The numerically biggest application of all is quite recent, namely the original Google page rank algorithm. This rank is the probability in the stationary distribution of a Markov chain which assigns a constant transition probability to each link on a web page, and additional tiny transition probability from any web page to any other to ensure ergodicity.

1.3.4 Metropolis-Hastings method

We want to *construct* a Markov chain such that it has a desired stationary distribution π_i. This means that we need to find a suitable transition probability matrix. The Metropolis-Hastings algorithm provides a transition matrix which *satisfies detailed balance* with respect to a given distribution π_i. If the Markov matrix is also ergodic (irreducible and aperiodic) it will then converge to π.

The Metropolis method was the first Markov chain Monte Carlo method, later generalized by Hastings. It is widely applicable since it is very simple. For specific applications, there are often much better methods, some of which are mentioned at the end of this chapter.

We construct the probability matrix p_{ij} for going from a state Z_i to some other state Z_j. Let the chain be in a state Z_i. Then we perform the following steps:

1. We *propose* to move to a state Z_j according to some *proposal probability* q_{ij}. It needs to be chosen in such a way that the Markov chain will be ergodic (irreducible and aperiodic). One needs to carefully make sure that this is indeed the case.

2. We *accept* the state Z_j as the next state of the chain with probability
$$p_{ij}^{accept} \;=\; \min\left(1,\, \frac{\pi_j\, q_{ji}}{\pi_i\, q_{ij}}\right) . \tag{1.23}$$

3. When Z_j is not accepted, then the next state of the chain will again be Z_i (!).

The overall transition matrix p_{ij} is the product of the probability q_{ij} to propose a step from i to j and the acceptance probability p_{ij}^{accept}:
$$p_{ij} \;=\; q_{ij}\, p_{ij}^{accept} . \tag{1.24}$$

Proof of detailed balance: We look at the case that the nominator in eq. (1.23) is smaller than the denominator. The opposite case is just an exchange of indices. Then

$$\begin{aligned}
\pi_i \, p_{ij} &= \pi_i \, q_{ij} \, p_{ij}^{accept} \\
&= \pi_i \, q_{ij} \, \frac{\pi_j \, q_{ji}}{\pi_i \, q_{ij}} \\
&= \pi_j \, q_{ji}
\end{aligned}$$

which is indeed the same as

$$\begin{aligned}
\pi_j \, p_{ji} &= \pi_j \, q_{ji} \, p_{ji}^{accept} \\
&= \pi_j \, q_{ji} \, 1 \; .
\end{aligned}$$

1.3.5 Making use of a single chain

With ergodicity and detailed balance satisfied, an algorithm like Metropolis will produce a Markov chain with π_i as the stationary distribution. Regardless of the original state of the chain, after a large amount of steps the final state will be from the distribution π.

One could then simulate a large number of Markov chains from some arbitrary initial state(s) (using different random numbers) and use the ensemble of final states as a sample of π. This strategy is inefficient, since it often takes a long time for the chains to "forget" the initial state, so that one must have very long chains for each individual sample, and even then the samples will still tend to be biased.

A much better strategy is to use a single chain (or a few with different initial states). First one waits for a rather long time to let the chain forget its initial state. After this “thermalization” one then takes many samples for measurements from the single chain, with a fairly small number of steps in between (described later). These samples will all be (approximately) from π because of the thermalization, but they will be more or less correlated with each other. One therefore needs a careful error analysis to make use of these numbers, which we will discuss in section 1.4.

1.3.6 Example: The Ising model

The Ising model was invented to describe ferromagnets. It is one of the simplest models of statistical mechanics. This model and related ones have applications in many other parts of physics and beyond. More information on these models as well as on Markov Chain Monte Carlo can be found in the lectures on "Phase transitions and Critical Phenomena".

The Ising model is concerned with variables called spins s_i living on a *lattice* of sites i, $i = 1, \ldots, N$, e.g. on a square or 3D lattice with N sites.

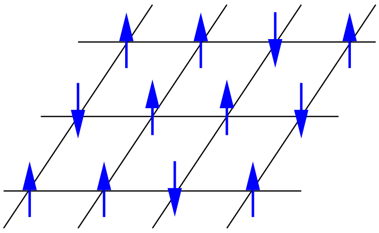


Figure 1.3: A spin configuration of the Ising model.

In the Ising model, the variables take only two values

$$s_i \in \{-1, +1\} .$$

These two values represent, for example, two spin states of an atom located at site i. A completely different incarnation of the Ising model is for example a binary alloy, where on each lattice site there is either an atom of species "A" or of species "B". The value of s_i then specifies this species. We will in the following use the language of ″spins″ and corresponding magnetism.

In a *ferromagnet*, spins prefer to be aligned. In the simplest Ising model, only spins on sites i and j that are nearest neighbors on the lattice interact. They have an energy

$$-J\, s_i s_j \ , \quad J > 0$$

which favors alignment. In addition, a magnetic field h changes the energy of each spin by $-h\, s_i$. The total energy of each configuration of spins $\mathbf{S} = (s_1, s_2, \ldots, s_N)$ is then

$$E(\mathbf{S}) = -\sum_{\langle ij \rangle} J\, s_i\, s_j \ - \ h \sum_i s_i \ . \tag{1.25}$$

The sum extends over all pairs $\langle ij \rangle$ of nearest neighbor sites.[6]

The magnetization of a spin configuration S is

$$M(\mathbf{S}) \ = \ \sum_i s_i \ . \tag{1.26}$$

For the Ising model, the spin configurations S are the states “x” of the probability distribution π. It is given by the *Boltzmann weight*

$$\pi(\mathbf{S}) \ = \ \frac{e^{-\beta E(\mathbf{S})}}{Z} \ , \tag{1.27}$$

[6] The expression for $E(\mathbf{S})$ can be interpreted as a quantum mechanical Hamilton operator. Then s_i represents the z-component of a quantum mechanical spin. However, there are no operators $\hat{S}_i^+$ or $\hat{S}_i^-$ which would *change* the value of s_i. The Hamilton operator is therefore already *diagonal* in the variables s_i. It is just a normal function of $\mathbf{S}$, without any operator effects, and the Ising model is in effect not quantum mechanical but a *classical* model of classical variables s_i.

where the normalization factor Z is the *partition function*

$$Z = \sum_{\mathbf{S}} e^{-\beta E(\mathbf{S})} . \tag{1.28}$$

The sum extends over all possible spin configurations. For N spins there are 2^N configurations. Therefore simple sampling is impossible except for very small toy systems.

Expectation values are given by (1.1): $\langle f \rangle := \sum_x f(x)\, \pi(x)$.
Some of the interesting observables of the Ising model are:

- Internal energy

$$\langle E \rangle = \sum_{\mathbf{S}} E(\mathbf{S}) \, \frac{e^{-\beta\, E(\mathbf{S})}}{Z} , \tag{1.29}$$

- Average magnetization of a configuration

$$\langle M \rangle = \sum_{\mathbf{S}} M(\mathbf{S}) \, \frac{e^{-\beta\, E(\mathbf{S})}}{Z} , \tag{1.30}$$

- Magnetic susceptibility

$$\chi = \frac{\partial}{\partial h} \langle M \rangle = \beta \big(\langle M^2 \rangle - \langle M \rangle^2 \big) . \tag{1.31}$$

In two or more dimensions, the Ising model exhibits a *phase transition*: At low temperatures (large β), allmost all spins are aligned: the system is in an ordered, ferromagnetic state.
At high temperatures (low β), the spins take almost random values and the system is in an unordered, paramagnetic state. For a 1D system and a 2D square lattice, the model can be solved exactly. All other cases need to be examined by numerical methods or approximate analytical techniques.

1.3.7 Choice of Monte Carlo steps

Because of its many states, we need Markov Chain Monte Carlo to evaluate observables numerically, without the systematic errors of analytical approximations.

There is a lot of freedom in choosing the Markov transition matrix. Different choices can affect the efficiency of a simulation drastically, even by many orders of magnitude, especially in the physically most interesting cases.

We will stay with the simple Metropolis-Hastings method for the Ising model. Let the current state "i" of the Metropolis method be the spin configuration S. We need to propose a change ("update") to a new state j, which here we call S′, namely S → S′ with some probability $q_{\mathbf{SS'}}$. We will propose the reverse transition $\mathbf{S}' \to \mathbf{S}$ with the same probability. Then $q_{ij} = q_{ji}$ for the proposal probability, and (only then) it cancels in (1.23)

$$p_{ij}^{accept} \;=\; \min\left(1,\; \frac{\pi_j\, q_{ji}}{\pi_i\, q_{ij}}\right) .$$

With $\pi(\mathbf{S}) = \frac{1}{Z} e^{-\beta E(\mathbf{S})}$ we get

$$p_{\mathbf{S}\,\mathbf{S}'}^{accept} \;=\; \min\left(1,\; \frac{e^{-\beta E(\mathbf{S}')}}{e^{-\beta E(\mathbf{S})}}\right) \;=\; \min\left(1,\; e^{-\beta \Delta E}\right) . \qquad (1.32)$$

The acceptance probability thus depends on the energy change $\Delta E = E(\mathbf{S}') - E(\mathbf{S})$. The proposed update will consist of some spin flips. If we propose to flip many spins at once, then the energy change will be large and will be accepted too rarely. For the Ising model, a suitable update proposal is to flip a *single* spin !

A single step of the Markov chain now consists of two parts:

- Choose a lattice site l at random. Propose to flip $s_l \to -s_l$, i.e. the configuration S′ differs from S only in the value of s_l.
- Compute ΔE. Accept $\mathbf{S}'$ with probability (1.32). If it is accepted, the next Markov chain configuration will be S′. If it is not accepted, the next configuration will *again* be **S**.

Irreducibility of this procedure is assured: with single spin flips one can eventually get from any spin configuration to any other. With the procedure just specified, aperiodicity also holds, because the diagonal probabilites p_{ii} to stay in the same state are finite (except for the lowest weight state(s)). Detailed balance is satisfied by construction. Thus this is a valid Monte Carlo procedure which will converge to the Boltzmann distribution.[7]

Obviously, the update of a single spin changes a configuration only by very little. Many such steps concatenated will eventually produce bigger changes. A useful unit of counting Monte Carlo steps is a *sweep*, defined to consist of N single spin steps, where N is the number of spins in the lattice. Observables are usually measured once every sweep, sometimes more rarely (see below).

1.3.8 Thermalization

A Markov chain is started in some configuration x_α. For instance, in an Ising model, x_α might be the configuration with "all spins up"; this is

[7] In practice, one often moves through lattice sites *sequentially* instead of at random. This violates ergodicity slightly but is usually harmless on big systems. In special cases (e.g. small system with periodic boundary conditions) it can induce a periodic Markov Chain, i.e. invalid results.

sometimes called a *cold* start. Alternatively, the Markov chain might be started in a random configuration, called a *hot* start. In any case, the initial state x_α tends to be far from the equilibrium distribution π. Therefore, the system is "out of equilibrium", and measurements $O(X_t)$ may initially have values far from their average. Theorems 1.1 and 1.2 guarantee that the system approaches equilibrium as $t \to \infty$, but we should know something about the *rate* of convergence to equilibrium. We will quantify this a little later.

One can take measurements before equilibrium is reached. This introduces a *bias* in the averages. When the number of measurements $n \to \infty$, the effect of the bias vanishes like $\frac{1}{n}$, so results eventually converge to the correct expectation values.

However, the bias can be quite large. In order to keep it small, one only starts measuring after a fairly long *equilibration phase* of some n_{therm} sweeps.

A good rule of thumb for choosing n_{therm} is about 10-20$\%$ (!) of the total number of sweeps in the simulation. Reason: It is good to have n_{therm} large to achieve a small bias, while reducing the available number of measurements by $10 - 20\%$ does not increase the statistical errors of averages significantly.

1.3.9 Parts of a Monte Carlo simulation

A Markov chain Monte Carlo consists of the following parts:

- Choose any starting configuration
- Thermalize for n_{therm} sweeps.
  The choice of starting configuration should not make a difference any more after thermalization ! This can be used as a check.
- Generate n configurations $X_t,\ t = 1, \ldots, n$, separated by one or more sweeps, with **measurements** $O_t := O(X_t)$. (t will from now on only count the sweeps after thermalization).
- **Analysis** of the resulting time series (see next section):
  - Compute averages.
  - Examine autocorrelations.
  - Perform error analysis.
  - Check against known results.

Thus a simulation can be divided into two parts, the data generation part and the data analysis. The interface between these two parts consists of time series of measurements of relevant physical observables taken during the actual simulation. Often a part of the analysis can be done during data generation, without the need to save complete time series.

1.4 Spectral Analysis of discrete time series

Before we analyze Monte Carlo time series, in this section we consider a generic stochastic process in discrete time, which may or may not be a Markov chain. There is a sequence of "times" $t_n = n = 1, 2, 3 \ldots$ (written without unit, for simplicity). Each time separately is associated with a random variable X_n taking real values x with some PDF $g_n(x)$.[8]

Each *realization* of the stochastic process is a so-called **trajectory** with coordinates $x_n = x(t_n)$. It might for example represent the motion of a particle. Note that there is a PDF for *whole trajectories* in the space of all possible trajectories. Expectation values are defined with respect to this overall PDF.

When a quantity like X_n only depends on a single time t_n, then its marginal PDF is g_n. *Expectation values* for single times t_n are

$$\langle X_n \rangle \;:=\; \int x \, g_n(x) \, dx \; . \tag{1.33}$$

1.4.1 Autocorrelation function

An important issue in the analysis of time series is the question of how strongly successive x_n are correlated with each other. Another related question is that of potential periodicities in the time series. In order to examine correlations, we look at the *covariance* of quantities.

We consider only the case that all expectation values are time-independent:

$$\begin{array}{lcl} \langle X_n \rangle & = \; \langle X_1 \rangle & =: \; \langle X \rangle \\ \mathrm{cov}\,(X_n, X_{n+t}) & = \; \mathrm{cov}\,(X_1, X_{1+t}) & =: \; R(t) \; , \end{array} \tag{1.34}$$

where in the second line we have given the name $R(t)$ to the covariance. Such time-independent time series are called *"stationary"*.[9]

Since expectation values are time independent, we can write the covariance as

$$\mathrm{cov}(X, Y) \;\equiv\; \Big\langle \; (X - \langle X \rangle) \; (Y - \langle Y \rangle) \; \Big\rangle \tag{1.35}$$

without time-indices. Note that the covariance measures the correlation of deviations from the expectation values. As usual,

$$\mathrm{cov}(X, X) \;=\; \mathrm{var}(X) \;=\; (\mathrm{std}(X))^2 \; . \tag{1.36}$$

[8] Note that X might be the energy of a system, or the magnetization, etc. In the general notation for a time series used here, each such case has a different PDF $g(x)$ for the values of the energy, or the magnetization, etc.. In a MCMC, these PDFs may all be due to an underlying distribution π_S (e.g. Boltzmann) in a phase space of *configurations* S, with different *functions* $f(S)$ providing energy, magnetization, etc. Then a notation $\langle f \rangle = \int f(S) \pi_S dS$ like at the beginning of this chapter (where the *configurations* were called "x") may be more intuitive.

[9] Caution: this is different from a "stationary Markov chain", which becomes a stationary *time series* only after and if it has exactly converged to its stationary distribution π.

We define the **autocorrelation function** $\rho(t)$ (also called *autocorrelation coefficient*) between quantities separated by a "temporal" distance t as the normalized covariance of X_n at some time n with X_{n+t} at time $n+t$:

$$\rho(t) \;:=\; \frac{\mathrm{cov}(X_n, X_{n+t})}{\mathrm{std}(X_n)\,\mathrm{std}(X_{n+t})} \;=\; \frac{\mathrm{cov}(X_0, X_t)}{\mathrm{std}(X_0)\,\mathrm{std}(X_0)} \;=\; \frac{R(t)}{R(0)}, \tag{1.37}$$

It tells us how much the *fluctuations* in X (i.e. the deviations from $\langle X\rangle$) are correlated over a time distance t. We will later examine the autocorrelation function in some detail for Monte Carlo Markov Chains.

1.4.2 Estimate of autocorrelations from a realization of a time series

Let (X_n) be a discrete time series which is stationary. We want to give estimates for the most important quantities on the basis of a **realization**, i.e. a set of data

$$x_1,\, x_2,\, \ldots,\, x_N$$

under the assumption that N is large enough for reliable estimates.

An estimator for the expectation value $\langle X\rangle$ is the average value

$$\bar{x} \;=\; \frac{1}{N}\sum_{i=1}^{N} x_i .$$

One possible estimate for the autocorrelation function $\rho(t)$ is

$$\frac{\sum_{j=1}^{N-t} (x_j - \bar{x})\,(x_{j+t} - \bar{x})}{\sum_{j=1}^{N-t} (x_j - \bar{x})^2}$$

which uses $\bar{x}$ as an estimator for $\langle X\rangle$. However, this estimator tends to be unstable, since for the first bracket the term $\bar{o}$ which is subtracted is different from the average of x_i, since the sum goes only up to $i = N - t$. Similarly for the second factor, and for nominator versus denominator.

A much more stable estimator is obtained by the so-called **empirical autocorrelation function**

$$\rho^E(t) \;:=\; \frac{\sum_{j=1}^{N-t} \left(x_j - \bar{x}_{(t)}\right)\left(y_j - \bar{y}_{(t)}\right)}{\sqrt{\sum_{i=1}^{N-t} \left(x_i - \bar{x}_{(t)}\right)^2 \;\sum_{j=1}^{N-t} \left(y_j - \bar{y}_{(t)}\right)^2}} \tag{1.38}$$

where

$$y_j \;:= x_{j+t}$$

is the time series shifted by t, and the averages $\bar{x}_{(t)}$ and $\bar{y}_{(t)}$ are calculated with the values of x_j and y_j which actually appear in the sums:

$$\begin{aligned}\bar{x}_{(t)} &:= \frac{1}{N-t}\sum_{j=1}^{N-t} x_j, \\ \bar{y}_{(t)} &:= \frac{1}{N-t}\sum_{j=1}^{N-t} y_j \;=\; \frac{1}{N-t}\sum_{j=1+t}^{N} x_j\end{aligned}$$

By construction, $-1 \leq \rho^E(t) \leq 1$. Large values of $|\rho^E(t)|$ indicate a large correlation (or anti-correlation) between data points at a time shift t.

For instance, when $\left|\rho^E(t)\right|$ is particularly large for $t = m, 2m, 3m$ in comparison to other times t, then there is a strong indication of a period $T = m$ in the data set (This does not usually happen in Markov chain Monte Carlo).

The autocorrelation coefficient $\rho^E(t)$ is called "empirical" because it refers to a particular realization of the stochastic process.

1.4.3 Computing covariances by Fast Fourier Transform

The calculation of the empirical autocorrelation function is very useful for the analysis of time series, but it is very time consuming, since for each value of t, one needs to sum O(N) terms. There are $O(N)$ possible values of the time distance t. Thus the direct calculation of the autocorrelation function needs $O(N^2)$ steps. This is very expensive, since the *generation* of the time series usually takes $O(N)$ time.[10]

Fast Fourier Transform (FFT)

A very efficient alternative makes use of the so-called Fast Fourier Transform. The FFT consists of an ingenious reorganization of the terms occuring in a Fourier transform. It allows the calculation of the Fourier series of a function $f(t_i)$, $i = 1, \ldots, N$ in only $O(N \log N)$ steps instead of $O(N^2)$. It works best when N is a power of 2, or at least contains only prime factors $2, 3, 5$. FFT is provided as a routine in numerical libraries.

Convolution Theorem for periodic functions

The second ingredient in the efficient calculation of covariances and autocorrelations is the convolution theorem.

[10] In the analysis of Monte Carlo time series, where $\rho(t)$ will hopefully decay fast, one usually needs only values $t < O(100) \ll N$, which is however still very expensive.

We start with functions $f(t_i)$, $g(t_i)$, with $t_i = i$ (again without units, for simplicity), defined in the range $1 \leq i \leq L$. We assume that these functions are *periodic* with period L.

The Fourier transform is

$$\tilde{f}(\omega_n) \;:=\; \frac{1}{\sqrt{L}} \sum_{j=1}^{L} e^{-i\omega_n t_j}\, f(t_j)\;, \tag{1.39}$$

where $\omega_n = 2\pi\frac{n}{L}$. We note that

$$\textit{If } f(t_i) \textit{ is real valued, then} \quad \tilde{f}(-\omega_n) \;=\; \Big(\tilde{f}(\omega_n)\Big)^{*}\;. \tag{1.40}$$

We calculate the convolution

$$\begin{aligned}
h(t) \;&:=\; \sum_{j=1}^{L} f(t-t_j)\, g(t_j) && (1.41)\\
&=\; \sum_{j=1}^{L} \frac{1}{\sqrt{L}} \sum_{n=1}^{L} e^{i\omega_n(t-t_j)}\, \tilde{f}(\omega_n) \quad \frac{1}{\sqrt{L}} \sum_{m=1}^{L} e^{i\omega_m t_j}\, \tilde{g}(\omega_m) && (1.42)\\
&=\; \sum_{n,m=1}^{L} e^{i\omega_n t} \quad \underbrace{\frac{1}{L} \sum_{j=1}^{L} e^{i(\omega_n-\omega_m)t_j}}_{=\delta_{n,m}} \quad \tilde{f}(\omega_n) \quad \tilde{g}(\omega_m) && (1.43)\\
&=\; \sqrt{L} \;\; \underbrace{\frac{1}{\sqrt{L}} \sum_{n=1}^{L} e^{i\omega_n t}\, \tilde{f}(\omega_n)\, \tilde{g}(\omega_n)}_{\text{inverse FT of } \tilde{f}\cdot\tilde{g}} && (1.44)
\end{aligned}$$

and see that in Fourier space it corresponds simply to a product.

We can now compute a convolution of periodic functions efficiently by first calculating the Fourier transforms of f and g by FFT, multiplying them, and finally computing the back-transform by FFT again, with a computational effort of $O(L \log L)$.

Correlation

We repeat the calculation for correlation functions. The only change is the sign of t_j in the argument of f:

$$\begin{aligned} C(t) &:= \sum_{j=1}^{L} f(t+t_j)\, g(t_j) && (1.45)\\ &= \sum_{n,m=1}^{L} e^{i\omega_n t} \quad \underbrace{\frac{1}{L}\sum_{j=1}^{L} e^{-i(\omega_n+\omega_m)t_j}}_{=\delta_{n,-m}} \quad \tilde{f}(\omega_n) \quad \tilde{g}(\omega_m) && (1.46)\\ &= \sqrt{L}\ \underbrace{\frac{1}{\sqrt{L}}\sum_{n=1}^{L} e^{i\omega_n t}\, \tilde{f}(\omega_n)\, \tilde{g}(-\omega_n)}_{\text{inverse FT}}\ . && (1.47) \end{aligned}$$

Autocorrelation

This is the case $f = g$.
If $f(t_i)$ is real-valued, then we get the simple and very useful result

$$\sum_{j=1}^{L} f(t+t_j)\, f(t_j) \quad = \quad \sqrt{L}\ \frac{1}{\sqrt{L}}\sum_{n=1}^{L} e^{i\omega_n t} \left|\tilde{f}(\omega_n)\right|^2 . \qquad (1.48)$$

Calculation of the autocorrelation therefore involves taking the Fourier transform, calculating the absolute square, and transforming back.

Non-periodic functions

We usually want to evaluate correlations for *non*-periodic functions

$$f(t_i), \quad i = 1, 2, \ldots N,$$

given on a finite set of N points, for a finite range of time intervals, $0 \le t \le t_{max}$. The trick to do this is by so-called **zero-padding**. We define

$$F(t_i) \;:=\; \begin{cases} f(t_i), & 1 \le i \le N, \\ 0, & N+1 \le i \le L, \end{cases} \qquad (1.50)$$

with $L \ge N + t_{max}$. (Similarly we extend a function $g(t_i) \to G(t_i)$.) Note that L is at most $2N$. Now the functions F and G are regarded as being periodic, with a long period L (see figure). The zeros in the definition of F ensure that $f(t+t_j)\, g(t_j) \;=\; F(t+t_j)\, G(t_j)$ in the desired range of arguments $1 \le j \le N$ and $0 \le t \le t_{max}$. Also, the right hand side is zero

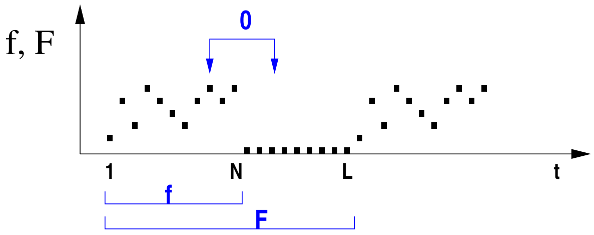


Figure 1.4: Zero Padding of a function f.

when $L < j \leq N$ because of $G(t_j)$. Therefore, the desired correlation of f and g can be calculated as

$$\sum_{j=1}^{N-k} f(t+t_j)\,g(t_j) \;=\; \sum_{j=1}^{L} F(t+t_j)\,G(t_j) \;=\; \sqrt{L}\,\frac{1}{\sqrt{L}}\sum_{n=1}^{L} e^{i\omega_n t}\,\tilde{F}(\omega_n)\,\tilde{G}(-\omega_n)\,, \tag{1.51}$$

as long as $t \leq t_{max}$.[11]

We can now compute the convolution (or correlation) of nonperiodic functions of length N by (i) zero-padding, (ii) FFT, (iii) multiplication, and (iv) back-FFT, with an effort of $O(L\log L) = O(N\log N)$ since $L \leq 2N$.

[11] Sometimes one defines the convolution as $\sum_{j=1}^{N-t_{max}} f(t+t_j)\,g(t_j)$ instead of $\sum_{j=1}^{N-t} \ldots$. Then one should define $G(t_j) = 0$ for $N + 1 - t_{max} \leq j \leq L$.

1.4.4 Spectrum

In order to see periodicities of a time series, it is useful to look at the Fourier transformation. We first examine a function

$$f(t_j), \;\; j = 0, \pm 1, \pm 2, \ldots$$

which is now defined for times from $-\infty$ to ∞, without boundaries in time. Its Fourier transform (Fourier series) is

$$\tilde{f}(\omega) \;:=\; \frac{1}{\sqrt{2\pi}} \sum_{j=-\infty}^{\infty} e^{-i\omega t_j}\, f(t_j), \tag{1.52}$$

with continuous frequencies $\omega \in [0, 2\pi)$.[12]

The inverse transformation is

$$f(t_j) \;=\; \frac{1}{\sqrt{2\pi}} \int_0^{2\pi} e^{i\omega t_j}\, \tilde{f}(\omega)\, d\omega\ . \tag{1.53}$$

Periodicities in $f(t_j)$ become peaks in $\tilde{f}(\omega)$ and can thus easily be detected. However, $\tilde{f}(\omega)$ is in general not real valued.

It is therefore useful to examine the so-called *"energy" spectral density*,[13]

$$\left| \frac{1}{\sqrt{2\pi}} \sum_{j=-\infty}^{\infty} e^{-i\omega t_j}\, f(t_j) \right|^2 \;\equiv\; \left| \tilde{f}(\omega) \right|^2 \tag{1.54}$$

which by definition is real and positive.

Parseval's theorem states that the total "energy" in the frequency domain is the same as that in the time domain:

$$\int_0^{2\pi} \left| \tilde{f}(\omega) \right|^2 \;=\; \sum_{j,k=-\infty}^{\infty} \underbrace{\frac{1}{2\pi} \int_0^{2\pi} e^{i\omega(j-k)}}_{=\delta_{jk}}\, f(t_k)\, f^*(t_j) \tag{1.55}$$

$$= \sum_{j=-\infty}^{\infty} |f(t_j)|^2\ , \tag{1.56}$$

which is also the same as the autocorrelation of f at distance zero.

Usually, the average value of $f(t_j)$ is non-zero. Then the sum above diverges and it is better to look at the Fourier series of the autocorrelation function $R(t)$ of $f(t_j)$, which is called the *power spectral density*

$$\tilde{R}(\omega) \;:=\; \frac{1}{\sqrt{2\pi}} \sum_{t=-\infty}^{\infty} e^{-i\omega t}\, R(t) \;\; , \omega \in [0, 2\pi)\, . \tag{1.57}$$

[12] When a time step Δt is used, $\omega \in [0, \frac{2\pi}{\Delta t})$

[13] When $f(t)$ is an electromagnetic field strength, then $|f(t)|^2$ is proportional to the electric power in the field.

By the convolution theorem, this is the same as the energy spectral density $|\tilde{f}(\omega)|^2$, except at $\omega = 0$ because of the subtraction of $\langle f\rangle^2$ in $R(t)$.

Example 1: Let only one autocorrelation coefficient be non-zero:

$$R(\pm t_n) \neq 0, \quad R(t_k) = 0, \quad \forall\, k \neq n$$

for a fixed number $n \geq 1, \in \mathbb{N}$. Then

$$\tilde{R}(\omega) = \frac{R(t_n)}{\sqrt{2\pi}} \left(e^{-in\omega} + e^{in\omega}\right) = \frac{R(t_n)}{\sqrt{2\pi}} 2\cos(n\omega), \quad \omega \in \mathbb{R}.$$

Example 2: Let only the variance $R(0)$ be unequal zero:

$$R(0) \neq 0, \quad R(t_k) = 0, \quad \forall\, k \neq 0,$$

then

$$\tilde{R}(\omega) = \frac{R(0)}{\sqrt{2\pi}}, \quad \omega \in \mathbb{R}.$$

Such an uncorrelated time series is called **white noise**.

Spectrum of finite length time series

Actual time series, say $f_1, f_2, \ldots f_N$, have a finite length N, as discussed before. One can formally write this finite length series as coming from an infinite series $x_1, x_2, \ldots$ (similar to zero-padding):

$$f_j \equiv f(t_j) \;:=\; x_j\, W(t_j), \;\; \text{where } W(t_j) := \begin{cases} 1, \; 1 \leq j \leq N \\ 0, \; \text{else} \end{cases} \tag{1.58}$$

using a *windowing function* $W(t_j)$. However, this affects the Fourier transform of f. By using the convolution theorem backwards, one gets

$$\tilde{f}(\omega) \;\equiv\; \left(\widetilde{x \cdot W}\right)(\omega) \;=\; \frac{1}{2\pi} \int d\nu\, \tilde{x}(\nu)\, \tilde{W}(\omega - \nu)\,, \tag{1.59}$$

i.e. the *convolution* of the desired $\tilde{x}$ *with the Fourier transform of the windowing function* W, which is a heavily oscillating function ('causing "ringing") in case of the "square" window W. Therefore one often dampens the cutoff of finite series with smoother windowing functions when one wants to analyze the spectrum.

Reversely, when one has a signal $f(t)$ that is the convolution of some desired signal x_n with some windowing function W (e.g. a Gaussian), then one can *try* to *deconvolve* $f(t)$ by dividing its Fourier transform by that of W. However, this often fails because of noise in the data and division by small numbers.

1.5 Statistical Analysis of Monte Carlo time series

1.5.1 Averages

When the time series data result from an importance sampling MC simulation, the expectation value $\langle O \rangle$ can be estimated as as a simple arithmetic mean over the Markov chain:

$$\bar{O} = \frac{1}{n} \sum_{t=1}^{n} O_t \, . \tag{1.60}$$

As pointed out at the beginning of this chapter, it is important to distinguish between the expectation value $\langle O \rangle$ which is an ordinary number, and the *estimator* $\bar{O}$ which is a random number fluctuating around the theoretically expected value. In principle, one could probe the fluctuations of the mean value directly, by repeating the whole MC simulation many times.[14] Usually, one estimates its variance.

$$\mathrm{var}(\bar{O}) \; := \; \left\langle \left[\bar{O} - \langle \bar{O} \rangle\right]^2 \right\rangle \; \equiv \; \left\langle \bar{O}^2 \right\rangle - \left\langle \bar{O} \right\rangle^2 \tag{1.61}$$

by first estimating the variance of the individual measurement

$$\sigma^2_{O_t} \equiv \mathrm{var}(O_t) = \left\langle O_t^2 \right\rangle - \left\langle O_t \right\rangle^2 \; , \tag{1.62}$$

by the sample variance $s^2(t)$, (1.4). If the n subsequent measurements were all uncorrelated, then the relation **would** simply be

$$\mathrm{var}(\bar{O}) = \frac{\mathrm{var}(O_t)}{n}, \tag{1.63}$$

For this relation to hold we also have to assume that the simulation is in equilibrium and time-translationally invariant. (See Sec. 1.3.) Equation (1.63) is true for any distribution of the values O_t.

Whatever form this distribution assumes, by the central limit theorem the distribution of the *mean* value is Gaussian, for uncorrelated data in the asymptotic limit of large n. The variance of the mean, $\mathrm{var}(\bar{O})$, is the squared width of this distribution. It is usually specified as the "one-sigma" squared error $\varepsilon^2_{\bar{x}} = \mathrm{var}(\bar{O})$, and quoted together with the mean value $\bar{O}$. Under the assumption of a Gaussian distribution, the interpretation is that about 68% of all simulations under the same conditions would yield a mean value in the range $[\bar{O} - \mathrm{std}(\bar{O}), \bar{O} + \mathrm{std}(\bar{O})]$.

[14] But this is expensive, and still requires a long time for each of the simulations to thermalize

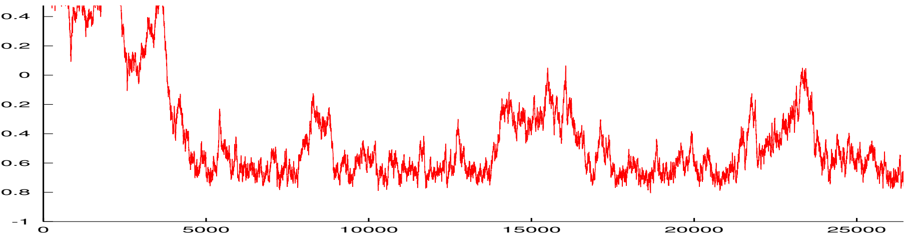


Figure 1.5: *Time series of successive measurements of the magnetization in an Ising model at low temperature. The* expectation *value of the magnetization is exactly zero. The average here is far from zero.*

1.5.2 Autocorrelations

In a simulation with very bad autocorrelations, successive measurements may for example look like Fig. 1.5. In the time series, we see long correlated regions. Furthermore, we know that the magnetization (without magnetic field) should average to zero. Therefore the negative values throughout most of the time series must be temporary, and actually most of the time series, over more than 25000 sweeps, is correlated ! As a result, the average value is far off its asymptotic value of zero.

In this example the long autocorrelations occur because at low temperature the spin configurations more or less "freeze" into a state with mostly spin-down (or mostly spin-up). With single-spin-flips, intermediate configurations with high energies must be overcome to get to the opposite magnetization. This will happen only very rarely. The situation is quite similar to the metastable behavior of real materials ($\rightarrow$ applications) even though the Monte Carlo time evolution with the Metropolis algorithm does not correspond to a physical Hamilton-evolution.

Another common reason for long Monte Carlo autocorrelations are large physical regions of similarly oriented spins. They occur close to second order phase transitions, which are physically very interesting to investigate. These regions have a typical size, which is called the *correlation length* ξ. Autocorrelation times with local updates then typically are of order $\tau \sim \xi^2$. Note though, that because of their very Markov nature, measurements from a Markov chain will *always* be autocorrelated to some smaller or larger degree[15] Thus we *always* have to take autocorrelations into account for error calculations.

As we have already discussed, they can be quantified by the **autocorrelation function**

$$\rho_O(t) \;=\; \frac{\langle O(t_0)\, O(t_0+t)\rangle \;-\; \langle O\rangle^2}{\langle O^2\rangle - \langle O\rangle^2} \;=\; \sum c_i(O)\, e^{-t/\tau_i}\,. \tag{1.64}$$

[15] Except for the rare cases that completely independent configurations can be chosen.

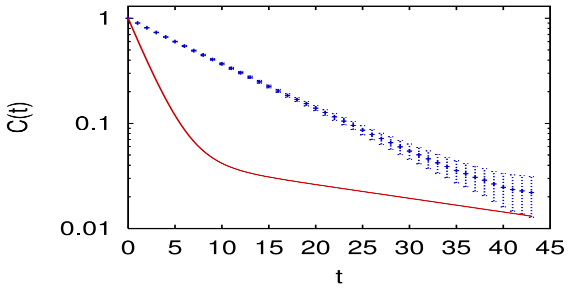


Figure 1.6: *Typical autocorrelation function. Note the log-scale.*

The last expression can be obtained by a lengthy calculation using the spectral representation of the Markov transition matrix P. It contains positive coefficients $c_i(O)$ and time scales τ_i, with $\frac{1}{\tau_i} = -\log \lambda_i$, which are related to the eigenvalues λ_i of the Markov matrix. These eigenvalues satisfy $|\lambda_i|^2 \in (0, 1]$ and are usually real valued. We see that the autocorrelation function is a **sum of exponentials**, and it should therefore always be analyzed on a *logarithmic* scale. Typical cases are depicted in Fig. 1.6.

The largest eigenvalue $\lambda_0 = 1$ of the Markov transition matrix corresponds to the equilibrium distribution π, since $\pi = \pi \, \mathbf{P}$. The next largest eigenvalue determines the largest time scale in the time evolution. It is called the *exponential autocorrelation time* $\tau_{exp}(O)$. For a specific operator O, the largest time scale $\tau_{exp}(O)$ occuring in $\rho_O(t)$ is determined by the largest eigenvalue with $c_i(O) \neq 0$. It sets the time scale for thermalization of O. In the autocorrelation function, plotted logarithmically, $\tau_{exp}(O)$ is the asymptotic slope, at large time separations t.

1.5.3 Statistical error and Integrated Autocorrelation time

From the autocorrelation function, one can compute the true variance of the average value $\bar{O} = \frac{1}{N} \sum_{i=1}^{n} \; O_i$. We now label the n measurements of O by O_i. We *assume* that the Markov chain has no memory of the initial configuration any more. Its probabilities are then time-translation invariant.

The variance of an individual measurement O_i can be expressed by *expectation values*:

$$\sigma^2_{O_i} \;=\; \langle O_i^2 \rangle - \langle O_i \rangle^2 \, .$$

We now compute the variance of the *average* value $\bar{O}$. If the measurements were independent, it would be equal to $\sigma^2_{O_i}/n$. In general, it is

$$
\begin{aligned}
\sigma^2_{\bar{O}} &\equiv \langle \bar{O}^2 \rangle - \langle \bar{O} \rangle^2 \\
&\equiv \frac{1}{n^2} \sum_{i,j=1}^{n} \langle O_i \, O_j \rangle \; - \; \frac{1}{n^2} \sum_{i,j=1}^{n} \langle O_i \rangle \, \langle O_j \rangle \\
&= \frac{1}{n}\frac{1}{n} \sum_{i=1}^{n} \left(\langle O_i^2 \rangle - \langle O_i \rangle^2 \right) + 2 \sum_{i=1}^{n} \sum_{j=i+1}^{n} \left(\langle O_i O_j \rangle - \langle O_i \rangle \, \langle O_j \rangle \right) \\
&= \frac{1}{n} \left[\sigma^2_{O_i} + 2 \sum_{t=1}^{n} \Big(\langle O_1 O_{1+t} \rangle - \langle O_1 \rangle \, \langle O_{1+t} \rangle \Big) \left(1 - \frac{t}{n} \right) \right]
\end{aligned}
$$

In the second line, we have inserted the definition of $\bar{O}$, and in the third line we collected diagonal terms $i = j$. The last line is obtained by using the assumed time translation invariance and writing $j = i + t$. The first bracket after the sum is equal to $\sigma^2_{O_i} \, \rho_O(t)$. We arrive at the important result that the actual variance of $\bar{O}$ is

$$
\boxed{\sigma^2_{\bar{O}} \; = \; \frac{\sigma^2_{O_i}}{n} \, 2 \, \tau_{int}(O)} \tag{1.65}
$$

with the so-called *integrated autocorrelation time*[16]

$$
\boxed{\tau_{int}(O) \; = \; \frac{1}{2} \; + \; \sum_{t=1}^{n} \rho_O(t) \left(1 - \frac{t}{n} \right) .} \tag{1.66}
$$

Equation 1.65 means that the effective number of measurements in the Monte Carlo run is a factor of $2\tau_{int}$ smaller than n !

$$
n_{eff} \; = \; \frac{n}{2\tau_{int}(O)} \, . \tag{1.67}
$$

It is this *effective* number of measurements which needs to be large in order to obtain reliable averages, whereas the number of measurements n itself has no direct meaning for error estimates !

The autocorrelation function $\rho_O(t)$ can be estimated from the data by the empirical autocorrelation function $\rho^E_O(t)$, (1.38). Note that in order to estimate it reliably, the time series needs to be about O(100) times as long as the largest time scale τ_i occuring in $\rho_O(t)$! Because of the exponential decay of $\rho_O(t)$ this also implies that the factor $(1 - \frac{t}{n})$ does not matter in practice.[17] It is therefore often omitted.

[16] When one performs the sum in (1.66) on Monte-Carlo data, one needs to be careful since $\rho_O(t)$ will eventually vanish into noise. Summing too far into the noise (which is like a random walk) could seriously influence the measured value of τ_{int}.

[17] When $n \gg \tau_{exp}$, then the autocorrelation function is exponentially small where $(1 - t/n)$ would matter. Whenn n is not $\gg \tau_{exp}$, then the time series is too short and has to be extended or thrown out, and $(1 - t/n)$ does not matter either.

Example: Let $\rho_O(t) = e^{-t/\tau_i}$ be a single exponential, with $n \gg \tau_i \gg 1$. Then (using $\rho_O(0) = 1$ in the second line)

$$\begin{aligned}
\tau_{int}(O) &\simeq \frac{1}{2} + \sum_{t=1}^{n} \rho_O(t) \;=\; -\frac{1}{2} + \sum_{t=0}^{n} (e^{-1/\tau_i})^t \\
&= -\frac{1}{2} + \frac{1-(e^{-1/\tau_i})^{n+1}}{1-e^{-1/\tau_i}} \\
&\simeq -\frac{1}{2} + \frac{1}{1-e^{-1/\tau_i}} \;\simeq\; -\frac{1}{2} + \frac{1}{1-(1-1/\tau_i)} \;\simeq\; -\frac{1}{2} + \tau_i \\
&\simeq \tau_i \, .
\end{aligned}$$

For the typical case $\rho_O(t) = \sum_{i=1}^{\infty} c_i(O)\; e^{-t/\tau_i}$ this means

$$\tau_{int}(O) \;\simeq\; \sum_i c_i\, \tau_i \tag{1.68}$$

when all $\tau_i \gg 1$. In a logarithmic plot, $\rho_O(t)$ consists of a sum of straight lines (but $\log \rho_O(t)$ itself is not straight when there is more than one term in in the sum) each of which contributes its inverse slope τ_i with a factor c_i equal to the intercept with the y-axis. Note that time scale with very small coefficients c_i can thus still be important (see Fig. 1.6).

1.5.4 Binning: Determine autocorrelations and errors

The autocorrelation function is rather cumbersome to calculate and to analyze. A much easier determination of errors can be done by *binning* the time series. The idea is that averages over very long parts of the time series, much longer than the autocorrelation time, are independent of each other (similar to independent Monte Carlo runs), and provide correct error estimates.

Since we do not know the autocorrelation time in advance, we have to use blocks of increasing lengths until the error estimate converges, which means that the blocks are then long enough. We perform the following steps:

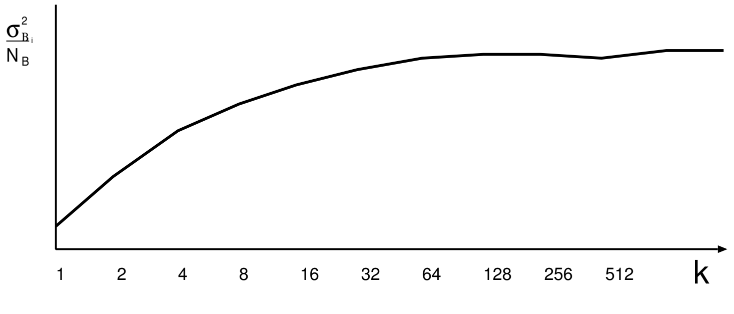


Figure 1.7: Error estimate by data binning. The value of $\frac{\sigma^2_{B_i}}{N_B}$ converges towards the true variance $\sigma^2_{\bar{O}}$ of the mean value $\bar{O}$.

- **Cut the time series into N_B blocks of length k**, where $k = 2^1, 2^2, 2^3, ...,$ $N_B = n/k$, and any remaining measurements are disregarded.[18] The block average on the i-th block is

$$\bar{O}_{B_i} = \frac{1}{k}\sum_{t=1}^{k} O_{(i-1)k+t} \;.$$

  All blocks together have a mean value of $\bar{O}_B$. The variance of the single-block values is estimated as usual

$$\sigma^2_{B_i} = \frac{1}{N_B - 1}\sum_{i=1}^{N_B}(\bar{O}_{B_i} - \bar{O}_B)^2 \tag{1.69}$$

  When the block length $k = 1$, then $\sigma^2_{B_i}$ is the "naive" estimate $\sigma^2_{O_i}$.

- The estimate of the variance of the *mean* $\bar{O}_{B_i}$ is $\sigma^2_{B_i}/N_B$. When the block length $k \gg \tau_{int}(O)$, then the block averages are uncorrelated, and this estimate converges to the correct variance of the mean $\bar{O}$:

$$\boxed{\text{Correct error}^2\text{ of mean:}\qquad \frac{1}{N_B}\,\sigma^2_{B_i} \;\overset{k\gg\tau_{int}(O)}{\longrightarrow}\; \sigma^2_{\bar{O}}} \tag{1.70}$$

  This is illustrated in figure 1.7.

- The longer the block length k, the fewer the number N_B of such blocks. When N_B becomes smaller than roughly 100, then the fluctuations in the estimated variance $\frac{\sigma^2_{B_i}}{N_B}$ become too large, as can be seen in fig. 1.7 at large k.

- **Convergence criterion:** The measurement of $\bar{O}$ can be considered to be converged when $\frac{\sigma^2_{B_i}}{N_B}$ stays constant (to within a few percent) over several bin-sizes.

  Then the run has been long enough to overcome autocorrelations, and this estimate of $\sigma^2_{\bar{O}}$ as well as the estimate $\langle O\rangle \approx \bar{O}$ are reliable. Since the number of blocks N_B needs to be at least about 100 for stable results, the simulation must be considerably longer than $100\tau_{int}$ for binning to converge.

  A useful early estimate can be obtained just by looking at the time series: the run should be longer than about **100 times the longest visible timescale**. (Of course longer time scales might become visible during long runs.)

[18] When the autocorrelation time is fairly small, then a less drastic increase of k, even $k = 1, 2, 3, ...$, may work better.

Caution: When $\frac{\sigma^2_{B_i}}{N_B}$ has not converged, then $\bar{O}$ can be completely wrong and the time series has to be extended a lot, or **thrown away** ! *Example:* Magnetization in the Ising model at low temperature: the exact value is $\langle M \rangle = 0$, but the mean $\bar{M}$ is often far from 0 because of extremely long autocorrelations.

- From (1.65), the integrated autocorrelation time for the quantity O is given by the ratio of the last and the first value in the binning plot (fig. 1.7):

$$2\tau_{int}(O) \;=\; \frac{\frac{\sigma^2_{B_i}}{N_B}(k \to \infty)}{\frac{\sigma^2_{B_i}}{N_B}(k = 1)} \tag{1.71}$$

- Note that autocorrelation times can be very different for different measured quantities. (Example: Magnetization and energy for the Ising model). The convergence has therefore to be checked independently for each measured quantity. Fortunately, the convergence criterion described above can easily be **automated** !

  However, if some quantities have not converged, then one has to carefully discuss the validity of other seemingly converged measurements (which may or may not be correct).

1.6 Summary: Recipe for reliable Monte Carlo simulations

Algorithm.

- We need a Markov Process that is ergodic and satisfies detailed balance.
- To verify the program, check Monte Carlo results against exact results when possible (e.g. limiting cases, small systems).

Simulation

- *Any starting configuration* should provide the same results, unless properties like metastability are to be analyzed.
- *Thermalization:* about 10% of the total number of sweeps. This tends to avoid bias from the starting configuration, but does not increase statistical errors noticeably.
- *Sweeps with measurements.* When $\tau_{int}(O) \gg 1$, one need not measure after every sweep.

Error analysis

- Check time series by eye ! All visible time scales must be *much* smaller than the simulation time.
- Binning: Check convergence for each measured quantity.
- Obtain reliable errors from binning and/or Jackknife.
- For precision results, check that the results are independent of the random number generator which is used.

Danger

When the system is too large, *autocorrelations may be much longer than the simulation*. The simulation can then *appear* to be converged (including binning !), but has in fact barely moved in phase space and is **wrong**.

⇒ **Think small !**

- One *must* start with very small systems.
- Check against exact results.
- Inspect for autocorrelations (especially by eye in time series).
- **Gradually** increase system size.

These essential steps are actually easy to do. They involve a few extra simulations on small systems, which run very quickly. Inspections for autocorrelations by eye are very easily done.

With these extra steps for verification, Monte Carlo calculations are very efficient and can provide reliable results even on extremely large systems.

1.6.1 Jackknife: simple procedure for automatic error propagation

Many interesting quantities are derived indirectly from actual (Monte-Carlo or other) measurements, and may for example involve ratios of expectation values, or fits, or a combination of other operations.

There is a very versatile and easy procedure for calculating errors with automatic error propagation in such cases, called *Jackknife*. Because of its generality far beyond Markov Chain Monte Carlo, it is described outside of the present chapter, in appendix A.4.

Note that it **can be combined with binning** (by using bigger and bigger bins in Jackknife) and can then also provide a proper autocorrelation analysis. In many cases, Jackknife is the method of choice for error analysis.

1.7 Appendix: Other Monte Carlo algorithms

This appendix provides a very quick overview over some other Monte Carlo methods, in addition to and beyond the original Metropolis method.

1.7.1 Some more advanced methods

Beyond the simple Metropolis Hastings method with local updates, many methods have been developed, which can be orders of magnitude better for suitable problems.

> **Gibbs sampler = Heat bath.** This is a different update proposal. In an update step, most degrees of freedom are held fixed. Then the distribution π amounts to a (conditional) distribution for the remaining degrees of freedom. They are sampled directly from this distribution. The proposed configuration is therefore always accepted. Of course this requires a method to do the sampling exactly within the restricted phase space. Note that in practice this method is not necessarily better that single site Metropolis updates.
>
> *Example:* All spins in the Ising model are held fixed except for spin s_i. In the next configuration it will have value $+1$ with probability
>
> $$\frac{e^{-\beta E(s_i=+1)}}{e^{-\beta E(s_i=+1)} + e^{-\beta E(s_i=-1)}} .$$
>
> This probability is independent of the current value of s_i.
>
> **Overrelaxation.** For continuous variables. When all other variables are held fixed, a variable x_i will see some effective potential. In overrelaxation, the proposal for the new value is not chosen at the minimum of this potential, but instead x_i "swings" beyond the minimum. This strategy anticipates that the surroundings will change in the future, likely following x_i. It can lead to faster convergence.
>
> **Fourier acceleration.** Also for continuous variables. Proposals are made in Fourier space. This is efficient when the Hamiltonian is close to bilinear, where the Fourier modes are independent.
>
> **Cluster methods.** For some classical model like the Ising model, and, more importantly, for a number of important Many-Particle Quantum Models (e.g. the Heisenberg model) there are efficient cluster methods with almost no autocorrelations and with other advantages. The clusters are generated stochastically, based on the current configuration. The methods work well, because these clusters turn out to be the physically relevant objects.

Introduction of additional degrees of freedom. Often, simulations become much easier when *additional* degrees of freedom are introduced. This way, barriers in phase space can be overcome, which is often advantageous, even when a large part of the extended phase space might be ″unphysical″. This is related to the extended ensembles discussed later. One very successful example are ″Worm Methods″ in Many Body quantum physics.

Exact Monte Carlo. Also called CFTP (″Coupling from the Past″). Surprisingly, it is possible to generate configurations which are guaranteed to be independent, without autocorrelations, and from the correct distribution (given a perfect random number generator). However, each new configuration takes a long time to generate.
See www.dbwilson.com/exact.

Multigrid methods. This is a very wide field in itself, with applications far beyond Monte Carlo. The strategy is to ″coarse grain″ the problem, finding effective degrees of freedom on a larger length scale, with some effective interaction. If the effective interaction is known precisely enough, the system can be simulated on the coarser scale.[19] When only an approximate effective interaction is known, then it can be used to generate an update proposal, which is then evaluated for the original variables and interaction.

1.7.2 Combination of algorithms

Sometimes it is advantageous to combine several Monte Carlo algorithms in order to improve performance.

Hybrid Monte Carlo:

We make use of the Metropolis-Hastings procedure, with eq. (1.23):

$$p_{ij}^{accept} \quad = \quad \min\left(1,\ \frac{\pi_j\, q_{ji}}{\pi_i\, q_{ij}}\right) \ .$$

Now we use as a proposal probability $q(x \to x')$ some procedure which *almost* satisfies detailed balance with respect to π. Then p_{accept} provides a *″filter″* to accept or reject these proposals in order to achieve detailed balance overall.

Example: In large scale simulations of Quantum Chromodynamics, update proposals are made by a molecular dynamics propagation of the quark and gluon degrees of freedom, and then accepted or rejected with the full

[19] Indeed, this is the strategy in most of physics, except for elementary particle physics and cosmology !

QCD weight, which is very expensive to evaluate. This way, larger steps in phase space and fewer evaluations of the full weight are possible.

A variant of the Hybrid MC method is to *split a Hamiltonian*: Let

$$Z\,\pi_i \;=\; e^{-\beta H_i} \;=\; e^{-\beta H_i^{(0)}}\, e^{-\beta H_i^{(1)}} \; .$$

If one has a Monte Carlo procedure to simulate $H^{(0)}$, one can use it in order to generate update proposals, which are then filtered with $H^{(1)}$.

Example: Efficient cluster updates for the Ising model work only without magnetic field. For a small field, it makes sense to use the cluster-updates as proposals and accept or reject them according to the proposed change of magnetization.

Random Mixing of methods

When each of the transition matrices P_1, P_2, ... satisfy detailed balance, then so does the mixture

$$P \;:=\; \sum \lambda_i\, P_i \tag{1.72}$$

with $\lambda_i \;\geq\; 0,\; \sum \lambda_i \;=\; 1$. One can therefore mix methods with different strengths, e.g. methods which update different degrees of freedom efficiently. *Example:* Mix a method which moves in phase space efficiently with another method that ensures ergodicity.
The following statement is related.

Sequence of methods

When the transition matrices P_1, P_2, ... are *stationary* with respect to π, then so is the sequence P_1P_2.... (Proof: $\pi P_1 P_2 = \pi P_2 = \pi$).

Again, this can be used in order to mix several methods.

Note that the analogous statement is not true for detailed balance, unless the transition matrices P_i commute.

1.7.3 Extended ensembles

Often, simulations are stopped by barriers in phase space from exploring important regions of phase space. This problem appears not only in standard Monte Carlo simulations of some probabilty distribution π, but **also in optimization problems** in a multitude of areas, which we will discuss in the next chapter.

Two successful strategies against such barriers are either to reduce their effective size, or to overcome them. Examples of these strategies are the so called *Multicanonical Ensemble* and *Tempering*. Both are versions of **umbrella sampling**.

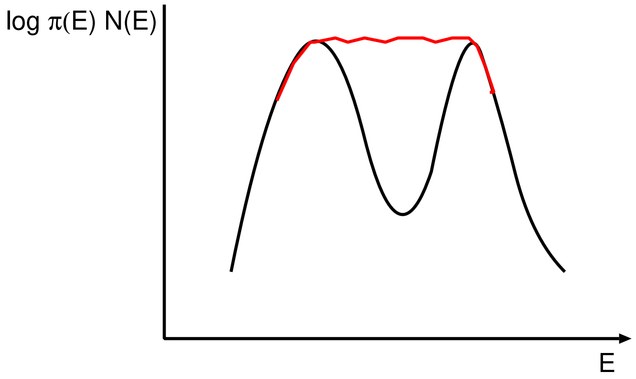


Figure 1.8: Histogram of how often specific energies occur in a Monte Carlo. There is a large barrier between small and high energies, which can be overcome by a suitably chosen new weight function.

Multicanonical ensemble

The probability of configurations with a certain energy E to appear in a Monte Carlo configuration is proportional to the Boltzmann weight $\pi(E) \sim \exp(-\beta E)$. It is also proportional to the *number of states* $N(E)$ with the energy E. This number is related to the entropy. Therefore the number of configurations with energy E in a Monte Carlo simulation is proportional to

$$\pi(E)\, N(E)\ .$$

We now consider the case of a large energy and/or entropy barrier, as sketched in fig. 1.8. The barrier between the two populated regions can often be many orders of magnitude high. For example, in the Potts model with $q > 4$, there is a first-order phase transition, with a jump in the average energy as a function of temperature. The barrier then corresponds to a surface tension between ordered and disordered phases, and can reach 10^{100} already for relatively small systems. It is then *impossible* for Monte Carlo configurations to move through this barrier. Sometimes, but rarely, one can devise methods to move from one favorable region directly to the other. For the Potts model, however, one even needs to explore the barrier itself in order to measure the surface tension.

The strategy of multicanonical simulations is to **reweight energies** in such a way that the **histogram** of energies becomes approximately **flat**. A flat histogram as depicted in fig. 1.8 means that the Monte Carlo will perform a **random walk in energies** (!) and will therefore move in energies fairly quickly, much quicker than the exponentially large time required to go through the minimum. The change of (unnormalized) weight is

$$\pi(E) \;=\; e^{-\beta E} \;\;\mapsto\;\; \tilde{\pi}(E) \;:=\; e^{-\beta E+\,f(E)} \;\;=\;\; W(E)\, e^{f(E)} \tag{1.73}$$

with a suitably *chosen* function $f(E)$, such that the histogram for a simulation with $\tilde{\pi}$ is approximately constant. The histogram does not need to be completely flat, it just needs to allow transitions between low and high energies. The function $f(E)$ is the difference between the two curves in fig. 1.8. In practise, this function is determined iteratively, by starting with cases (e.g. small systems) where the barrier is not high.[20]

The Monte Carlo is now performed with the new weights $\tilde{\pi}$. Fortunately, one can still calculate expectation values with respect to the original weight π, by reweighting the measurements back (this works for any reweighting function f_x):

$$\langle O \rangle_\pi \;\equiv\; \frac{\sum_x O_x \, \pi_x}{\sum_x \pi_x} \;=\; \frac{\sum_x O_x \, e^{-f_x} \, \tilde{\pi}_x}{\sum_x e^{-f_x} \, \tilde{\pi}_x} \;=\; \frac{\frac{\sum_x O_x \, e^{-f_x} \, \tilde{\pi}_x}{\sum_x \tilde{\pi}_x}}{\frac{\sum_x e^{-f_x} \, \tilde{\pi}_x}{\sum_x \tilde{\pi}_x}} \;=\; \frac{\langle O \, e^{-f_x} \rangle_{\tilde{\pi}}}{\langle e^{-f_x} \rangle_{\tilde{\pi}}} \,. \tag{1.74}$$

Indeed, one can also determine expectation values with respect to *different temperatures*.[21] The only requirement is that the probability distribution at the desired temperature is *contained* completely within the distribution that is actually simulated (fig. 1.8). I.e. one simulates an *umbrella* of all desired distributions. Note that one can also reweight in quantities other than the energy.

The multicanonical technique has found important applications in optimization problems, notably in the **Traveling salesman** and related problems, and in **Protein folding**. This will be discussed in the next chapter.

Free energy measurements

The free energy F of a statistical system can be defined via the partition function by

$$Z = e^{-\beta F}.$$

Knowledge of F is therefore equivalent to knowledge of the partition function, which in its parameter dependence contains all thermodynamic quantities, and also provides knowledge of the entropy S, via $F = U - TS$.

Unfortunately, in Markov Chain Monte Carlo, Z is just a normalization factor for the sum over all configurations, which in Monte Carlo becomes the normalization by the number of measured configurations, which does not give any information about Z.

However, by clever application of the Multicanonical Method (and similarly for Tempering, see below), Z or the free energy F can indeed be measured.

[20] Note that there should not be too many values for the argument of the reweighting function because that many values of $f(E)$ need to be determined. A quantity like the energy is ok here, but f should not be independently different for every configuration x.

[21] Hence the somewhat unfortunate name ”multicanonical”

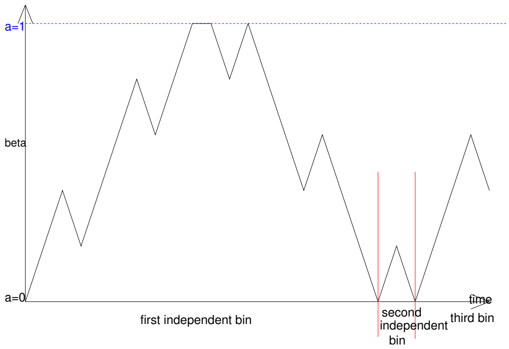


Figure 1.9: Tempering: Random walk in β.

We first classify the sum over all configurations x into the different energies:

$$Z = \sum_x e^{-\beta E(x)} = \sum_E \sum_{x|E} e^{-\beta E} = \sum_E N_E \, e^{-\beta E}, \tag{1.75}$$

where N_E is the number of states in phase space with energy E (density of states). We then measure how many configurations of a given energy occur in a Monte Carlo. The ratio of expectation values is

$$\frac{\langle N_{E_1} \rangle}{\langle N_{E_2} \rangle} = \frac{N_{E_1} e^{-\beta E_1}}{N_{E_2} e^{-\beta E_2}} \,. \tag{1.76}$$

The multicanonical method flattens the distribution of energies in the Monte Carlo over some chosen range of energies. We can choose to include all possible energies of the system, also the lowest (or highest), for which the number of configurations is usually known exactly (e.g. exactly two completely ferromagnetic states in the Ising model). From this exact knowledge and (1.76), all the other $N(E)$ can be measured, and then Z can be calculated via (1.75).

Tempering

The strategy of tempering is to overcome barriers by making the **temperature a dynamical variable** with discrete values β_i of β, which can now change during a simulation. The extended phase space of the simulation now consists of the original variables x and the index i of temperature.[22]

Simulated Tempering

The partition function in this extended space for Simulated Tempering is

$$Z = \sum_i \sum_x e^{-\beta_i H_x + g_i} = \sum_i Z(\beta_i) \,. \tag{1.77}$$

[22] Literally, this approach ought to be called ”multicanonical”, but it is not.

One chooses constants g_i in such a way that the system approximately performs a **random walk in temperatures**.

When it is at a sufficiently high temperature, then the system can easily overcome energy barriers. Indeed, if $\beta = 0$ is included in the simulation, the configurations will *decorrelate completely* whenever $\beta = 0$ is reached, cutting off any autocorrelations. When $\beta = 0$ is included, simulated tempering also allows the measurement of entropy, which is usually impossible in Monte Carlo (see below).

When one considers only the configurations at a certain inverse temperature β_i, then (1.77) implies that they are from the normal **canonical ensemble** $\exp(-\beta_i E)$! Therefore one can directly measure observables in the canonical ensemble at any of the β_i, including the properties of the system at some low temperature, where the barriers are felt fully.

For tempering to work, the β_i need to be chosen more densely close to a phase transition. At a strong first order phase transition, the multicanonical method works better.

Partition function: Equation 1.77 can be regarded as a partition function in the space of indices i, with weights $Z(\beta_i)$. The Monte Carlo is done in this space, and the number of configurations N_i at certain temperatures β_i must then obey

$$\frac{\langle N_i \rangle}{\langle N_j \rangle} = \frac{Z_i}{Z_j} \; . \tag{1.78}$$

Simulated tempering covers a range of β values. If one includes $\beta = 0$, the situation simplifies. At $\beta = 0$, the partition function is $Z_0 = \sum_x e^0 = N_x$, namely just the total number of configurations in phase space. Thus Z_0 is known, and measurements of $\langle N_i \rangle$ provide Z_i via (1.78) also for other temperatures.

Parallel Tempering

In parallel tempering, one simulates all inverse temperatures β_i in parallel, e.g. on different processors. Occasionally, the interchange of configurations at neighboring temperatures is proposed, and accepted with Metropolis probability according to (1.77). For such an interchange, the constants g_i *cancel*. They are therefore not required in parallel tempering and need not be determined. As a drawback, the entropy cannot be determined.

There are many applications of Tempering outside of physics, including again optimization problems like the **Traveling salesman** and **Protein folding**, as well as Bayesian **parameter determination from measured data**. This will be discussed in more detail in the next chapter.

(This page is empty)

Chapter 2

Minimization/Optimization – Problems

Literature

- W. PRESS *et al.*, *Numerical recipes*, Cambridge University Press
- F.S. ACTON, *Numerical Methods that work*, Mathematical Association of America

Optimization is of enormous importance in physics, engineering, economics, information technology and in many other fields. The layout of electric circuits on a chip, timetables, or the optimum load of a pipeline system are a few typical examples.

Usually we have a problem which is determined by a set of n real parameters. Each set of parameter values specifies a *state*. Together they span the *search space*. Each state is mapped to a real number, its *cost*, by a **cost function**. One has to find the specific values of the parameters for which the cost function develops a maximum/minimum. Often, one also requires **robustness of the solution** against small fluctuations of the parameters.

If the cost function is at least once differentiable in its parameters, then mathematics will provide us with well defined (deterministic) algorithms to find at least *local* maxima/minima of the cost function. Such methods will be discussed first.

However, the state space is often far too large for such methods, or there are too many extrema, or the value of the cost function is not differentiable. Then other approaches have to be used, often *stochastic methods* similar to Markov Chain Monte Carlo. In these one starts from some state and modifies it. If the cost function changes to a better value or does not worsen too much, according to some strategy, then the new state is taken. This is repeated until an optimum has been found, as specified by some suitable criterion.

2.1 Examples

Let us discuss some examples with many continuous degrees of freedom.

1. In many cases, the problem to be solved can be written as *a set of linear algebraic equations*. Its solution can be viewed as a minimization problem

$$\begin{aligned} \boldsymbol{B}\,\mathbf{x} = \mathbf{b} \quad \Longleftrightarrow \quad & \min_{\mathbf{x}} \|\boldsymbol{B}\,\mathbf{x} - \mathbf{b}\|^2 \\ &= \min_{\mathbf{x}} \left[\mathbf{x}^\dagger B^\dagger B\mathbf{x} - 2\mathrm{Re}\,\mathbf{b}^\dagger \boldsymbol{B}\mathbf{x} + \|\mathbf{b}\|^2\right]. \end{aligned} \tag{2.1}$$

   For large-dimensional matrices $\boldsymbol{B}$, the direct inversion may easily exceed present computer power. We will see that iterative minimization procedures extend the accessible sizes by many orders of magnitude.

2. In quantum mechanics, the *ground state energy* E_0 of a system is given by the minimum of the energy expectation value

$$E_0 = \min_{\psi} \frac{\left\langle \psi \middle| \hat{H} \middle| \psi \right\rangle}{\langle \psi \,|\, \psi \rangle}. \tag{2.2}$$

   When expressed in a complete orthonormal basis set $\{|\Phi_i\rangle\}$ this is equivalent to

$$E_0 = \min_{\mathbf{c}} \frac{\mathbf{c}^\dagger \boldsymbol{H} \mathbf{c}}{\mathbf{c}^\dagger \mathbf{c}}, \tag{2.3}$$

   where **c** stands for the vector of expansion coefficients of the ground state vector and $\boldsymbol{H}$ is the Hamilton matrix in the basis $\{|\Phi_i\rangle\}$, *i.e.:*

$$H_{ij} = \left\langle \Phi_i \middle| \hat{H} \middle| \Phi_j \right\rangle.$$

   A standard approach to finding the ground state energy (and with less precision also its wave function) is the *variational ansatz* with a set of parameters $\boldsymbol{\eta}$, spanning some part of the total Hilbert space.

$$|\psi\rangle = |\psi(\boldsymbol{\eta})\rangle, \quad \boldsymbol{\eta} \in \mathbb{R}^n \text{ parameters}. \tag{2.4}$$

   The minimum energy

$$E_{0,\vec{\eta}} = \min_{\boldsymbol{\eta}} \frac{\left\langle \psi(\boldsymbol{\eta}) \middle| \hat{H} \middle| \psi(\boldsymbol{\eta}) \right\rangle}{\langle \psi(\boldsymbol{\eta}) \,|\, \psi(\boldsymbol{\eta}) \rangle}. \tag{2.5}$$

   in this n-dimensional space is guaranteed to approximate E_0 from above (see introductory quantum mechanics).

   However, with many degrees of freedom (many particles, many electrons), the size of the problem grows exponentially and can quickly become too large.

3. To compute static or dynamic properties of molecules or solids, one can take an *entirely classical approach*, in which only the positions of the nuclei are accounted for. Empirical forces among the nuclei are introduced, which mimic parts of the quantum mechanical features. Usually, only effective two-particle forces are employed, which for the i-th particle are of the form

$$\mathbf{F}_i = \sum_j \mathbf{f}(\mathbf{R}_i, \mathbf{R}_j), \tag{2.6}$$

   with $\mathbf{R}_i$ being the coordinate of the i-th particle. For static quantities, the equilibrium positions need to be determined, thus the energy needs to be minimized. Such an approach is for example taken to model the structure of large *proteins* from knowledge of their amino acid sequence.

   For the dynamical time evolution of such a system, the equations of motion can be solved by so-called *molecular dynamics* techniques, which will be treated in the next chapter.

4. A significant improvement towards a complete quantum mechanical description can be made by treating the electrons of a material quantum mechanically, while the nuclei are still treated as classical degrees of freedom. The nuclei now provide an external potential V_{ext} for the electrons. This approach is known as the *Born-Oppenheimer approximation*. It is often (but not always) justified because the mass of protons and neutrons is roughly 2000 times larger than that of electrons, so that nuclei tend to move much more slowly than electrons.

   There are different possible strategies for solving the electronic part of the problem. If the system is not too large, one can employ a variational approach as described above, often together with some approximation of the electron-electron forces.

5. The most widespread method to calculate materials properties employs the Born-Oppenheimer approximation and then uses *Density Functional Theory (DFT)* for the electronic problem. It is based on two theorems by Hohenberg and Kohn (1964) showing that for a given external potential V_{ext}, the *ground state* of a many-body electronic system is uniquely determined by just the electron density $n(\mathbf{x})$ (!), and that the ground state energy can be obtained by minimizing a functional (an energy) in the space of all density functions $n(\mathbf{x})$. Kohn and Sham (1965, Nobel prize for Kohn 1998) showed that the problem of N interacting electrons can be mapped to N solutions of a non-interacting Schrödinger equation, coupled indirectly through $n(\mathbf{x})$, subject to an effective potential V_{eff} which encodes all many body interactions. However, this effective potential is *not exactly known* and needs to be approximated.

The most common approximation method for V_{eff} is the *Local Density Approximation (LDA)*, which approximates $V_{eff}(\mathbf{x})$ to depend only on the local value of $n(\mathbf{x})$ at the point x instead of the actual dependence on the density at all sites together. It can be improved by including a dependence on the gradient of $n(\mathbf{x})$ in so called generalized gradient approximations (GGA).

In DFT one needs to solve the Kohn-Sham equations[1]

$$\begin{aligned} \hat{H}(\{n\},\mathbf{R})\,\psi_{\mathbf{k}}(\mathbf{x}) &= \varepsilon_{\mathbf{k}}\,\psi_{\mathbf{k}}(\mathbf{x}) \\ n(\mathbf{x}) &= \sum_{\mathbf{k}\leq\mathbf{k}_{Fermi}} |\psi_{\mathbf{k}}(\mathbf{x})|^2 \end{aligned} \tag{2.7}$$

where the effective Hamiltonian H depends on V_{eff}. These equations contain a self-consistency condition between $n(\mathbf{x})$ and V_{eff}, which can be solved by iterating $n \to V_{eff}(n) \to n \to \ldots$ until convergence.

Such calculations are commonly called as coming from *"first principles"*, which refers to the exactness of the theorems by Hohenberg and Kohn. However, since the effective potential V_{eff} needs to be approximated, e.g. by LDA, the results are *not* exact. Because of the *local* approximation in LDA, they neglect part of the non-local many-body effects, for example spatial correlations, and tend to work badly with strongly correlated materials and also with phenomena like van der Waals forces.

Often one is interested in the equilibrium geometry of the nuclei. Then one needs to minimize the total energy of nuclei and electrons

$$\min_{\{\mathbf{R}_j\}} E_{tot}(\{\mathbf{R}_j\}),$$

which is again an optimization problem and obviously a challenging task as it involves the self-consistent solution of the electronic eigenvalue problem for each configuration $\{\mathbf{R}_j\}$.

[1] There are several software packages for DFT. Standard packages in widespread use include VASP and Wien2k, both developed in Vienna.

2.2 Quadratic Problems: Steepest Descent and Conjugate Gradient

A *quadratic form* is a scalar, quadratic function of a vector and is of the form:

$$\boxed{f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \boldsymbol{A}\mathbf{x} - \mathbf{b}^T\mathbf{x} + c.} \tag{2.8}$$

[See also Eq. (2.1).] Here, $\boldsymbol{A}$ is an $N \times N$ matrix, $\mathbf{x}$ and $\mathbf{b}$ are vectors $\in \mathbb{R}^N$, and c is a scalar constant. *We restrict the discussion to real-valued problems.*

Theorem: If the matrix $\boldsymbol{A}$ is symmetric (i.e. $\boldsymbol{A}^T = \boldsymbol{A}$) and positive definite (i.e. $\mathbf{x}^T\boldsymbol{A}\mathbf{x} > 0$ for any non-zero vector $\mathbf{x} \in \mathbb{R}^N$) the quadratic form (2.8) is *minimized* by:

$$\boxed{\boldsymbol{A}\mathbf{x} = \mathbf{b}.} \tag{2.9}$$

To prove this, we compute the gradient of $f(\mathbf{x})$:

$$\begin{aligned}
\nabla f(\mathbf{x}) &= \frac{1}{2}\nabla\left(\mathbf{x}^T\boldsymbol{A}\mathbf{x}\right) - \nabla\left(\mathbf{b}^T\mathbf{x}\right) \\
&= \frac{1}{2}\boldsymbol{A}^T\mathbf{x} + \frac{1}{2}\boldsymbol{A}\mathbf{x} - \mathbf{b}
\end{aligned}$$

$$\boxed{\nabla f(\mathbf{x}) \;=\; \boldsymbol{A}\mathbf{x} - \mathbf{b}} \tag{2.10}$$

Setting the gradient to zero produces (2.9), therefore $\boldsymbol{A}\mathbf{x} = \mathbf{b}$ correponds to an extremum of $f(\mathbf{x})$.

When $\boldsymbol{A}$ is not symmetric, i.e. $\boldsymbol{A}^T \neq \boldsymbol{A}$, then we see from this derivation that one will find a solution to the system $\tilde{\boldsymbol{A}}\mathbf{x} = \mathbf{b}$ with $\tilde{\boldsymbol{A}} = \frac{1}{2}\left(\boldsymbol{A}^T + \boldsymbol{A}\right)$ which is a symmetric matrix.

When the matrix $\boldsymbol{A}$ is also positive definite, then $\boldsymbol{A}\mathbf{x} = \mathbf{b}$ corresponds to a *minimum* of $f(\mathbf{x})$. To see this, let us assume that $\mathbf{x}$ is a point $\in \mathbb{R}^N$ which satisfies $\boldsymbol{A}\mathbf{x} = \mathbf{b}$. Let $\mathbf{e}$ be an "error" term, i.e. a deviation from the solution vector $\mathbf{x}$.

$$\begin{aligned}
f(\mathbf{x}+\mathbf{e}) &= \frac{1}{2}(\mathbf{x}+\mathbf{e})^T\boldsymbol{A}(\mathbf{x}+\mathbf{e}) - \mathbf{b}^T(\mathbf{x}+\mathbf{e}) + c \\
&= \frac{1}{2}\mathbf{x}^T\boldsymbol{A}\mathbf{x} + \frac{1}{2}\mathbf{e}^T\boldsymbol{A}\mathbf{e} + \underbrace{\frac{1}{2}(\mathbf{e}^T\mathbf{b} \;+\; \mathbf{b}^T\mathbf{e})}_{\mathbf{b}^T\mathbf{e}} - \mathbf{b}^T\mathbf{x} - \mathbf{b}^T\mathbf{e} + c \\
&= \frac{1}{2}\mathbf{x}^T\boldsymbol{A}\mathbf{x} - \mathbf{b}^T\mathbf{x} + c + \frac{1}{2}\mathbf{e}^T\boldsymbol{A}\mathbf{e} \\
&= f(\mathbf{x}) + \frac{1}{2}\mathbf{e}^T\boldsymbol{A}\mathbf{e}.
\end{aligned}$$

In the second line we used the symmetry of $\boldsymbol{A}$ to write $\mathbf{x}^T\boldsymbol{A}\mathbf{e} = \mathbf{b}^T\mathbf{x}$, and also $\mathbf{b}^T\mathbf{x} = \mathbf{x}^T\mathbf{b}$. If the matrix $\boldsymbol{A}$ is positive definite, then the last term is positive for all $\mathbf{0} \neq \mathbf{e} \in \mathbb{R}^N$ and therefore $\mathbf{x}$ minimizes $f(\mathbf{x})$.

Eq. (2.9) can be solved by a number of methods. A "direct" solution can be obtained, for example, by Gaussian elimination, the Gauss-Newton method, Cholesky-decomposition, or a similar method. These methods are related to computing the minimum directly via the inverse of $\boldsymbol{A}$:

$$\nabla f(\mathbf{x}) = \boldsymbol{A}\mathbf{x} - \mathbf{b} = 0 \quad \Rightarrow \quad \mathbf{x} = \boldsymbol{A}^{-1}\mathbf{b} \; . \tag{2.11}$$

They require $O(N^3)$ operations, which can, however, be prohibitive for large matrices.

In the present chapter, we will examine methods which start from some initial guess $\mathbf{x}_0$ and iteratively improve it, $\mathbf{x}_n \to \mathbf{x}_{n+1}$, moving towards the minimum of $f(\mathbf{x})$ by using the **gradient** ∇f at the current $\mathbf{x}_n$. Let us estimate the computational effort for such methods. The number of degrees of freedom, i.e. the number of elements of $\boldsymbol{A}$ and $\mathbf{b}$, is of order N^2. If one of these elements is modified, the solution $\mathbf{x}$ will change. The gradient $\nabla f(\mathbf{x})$ has N components of information. *A good method based on gradients should therefore ideally need at most N iterations.* The computational effort to compute the gradient is $\mathcal{O}(N^2)$; to compute it N times, the effort is $\mathcal{O}(N^3)$.

At first glance, the two approaches seem to be similar as far as CPU time is concerned. This is not true in practice, though, since with iterative methods based on gradients, there is usually no need to really perform all N iterations. In most large problems *a much smaller number of iterations suffices* to achieve some desired accuracy. Other properties of such methods:

- Gradient methods are especially suited for sparse matrices $\boldsymbol{A}$. The effort to compute the gradient is proportional to the number m of non-zero elements (typically $\mathcal{O}(mN)$, with $m << N$); i.e. the entire approach is of order $\mathcal{O}(mN^2)$ instead of $\mathcal{O}(N^3)$.
- They can do with any amount of memory. There are three possibilities
  - keep the entire matrix in fast memory
  - read it from hard disk in portions, which can be kept in memory
  - generate the matrix elements from scratch as they are needed
- They are well suited for parallelization or vectorization.

2.3 Steepest Descent: not a good method

The most elementary gradient-based algorithm is the method of steepest descent. At each step we move in the direction opposite to the gradient of $f(\mathbf{x}_n)$, i.e. downhill, until we reach the local minimum in that direction.

This method turns out to fail easily. It can, however, be modified to become the very useful method of Conjugate Gradients, discussed in the following section. Both Steepest Descent and Conjugate Gradient are methods to minimize quadratic functions (i.e. solve linear equations). They can be generalized to more general functions, as we shall see.
The outline of Steepest Descent is as follows

Algorithm 2 Steepest descent (draft)

```
Choose an initial vector $\mathbf{x}_0$
for $n = 0$ to $n_{\max}$ do
    calculate the gradient $\mathbf{g}_n = \nabla f(\mathbf{x})\big|_{\mathbf{x}_n}$
    new search direction $\mathbf{r}_n := -\mathbf{g}_n \;=\; \mathbf{b} - \boldsymbol{A}\mathbf{x}_n$
    set $\mathbf{x}_{n+1}$ to the minimum of $f(\mathbf{x})$ in direction $\mathbf{r}_n$
    if converged then EXIT
end for
```

The quantity $\mathbf{r}_n = \mathbf{b} - \boldsymbol{A}\mathbf{x}_n$ is called the *"residual"*. Its modulus $|\mathbf{r}|$ can be used to judge how well x_n has converged to the solution $\mathbf{x}$.

When we take a step n, we choose the direction in which $f(\mathbf{x}_n)$ decreases most quickly, and this is the direction opposite $\nabla f(\mathbf{x}_n) = \boldsymbol{A}\mathbf{x}_n - \mathbf{b}$. Suppose we start at the point $\mathbf{x}_0$. Our first step is along the direction of steepest descent. Thus, we will choose a point

$$\mathbf{x}_1 = \mathbf{x}_0 + \lambda_0 \mathbf{r}_0, \quad \mathbf{r}_0 = \mathbf{b} - \boldsymbol{A}\mathbf{x}_0 = -\nabla f(\mathbf{x}_0). \tag{2.12}$$

We choose λ_0 to minimize $f(\mathbf{x})$ along the line $\mathbf{x}_0 + \lambda_0 \mathbf{r}_0$ (*"line minimum"*). The miminum is reached when the *directional derivative* $df(\mathbf{x})/d\lambda_0$ is equal to zero. By the chain rule

$$0 \overset{!}{=} \frac{d}{d\lambda} f(\mathbf{x}_1) = \nabla f(\mathbf{x}_1)^T \frac{d}{d\lambda}\mathbf{x}_1 \overset{(2.12)}{=} \underbrace{\nabla f(\mathbf{x}_1)^T}_{-\mathbf{r}_1^T} \mathbf{r}_0 \;=\; -\mathbf{r}_1^T \mathbf{r}_0 \,.$$

We find that λ_0 should be chosen such that $\mathbf{r}_0$ (the *residual*) and $\nabla f(\mathbf{x}_1)$ are orthogonal. Since we always go to the respective line minimum, successive gradients are perpendicular. This is illustrated in Fig. 2.1 which shows two successive search directions (gradients) along with the contours of $f(\mathbf{x})$. If the gradients were not orthogonal, we could still reduce $f(\mathbf{x})$ by moving further in the direction of the old gradient.

For step n the strategy reads

$$\mathbf{r}_n \;=\; -\nabla f(\mathbf{x}_n) \;=\; \mathbf{b} - \boldsymbol{A}\mathbf{x}_n \tag{2.13a}$$

$$\mathbf{x}_{n+1} \;=\; \mathbf{x}_n + \lambda_n \mathbf{r}_n \tag{2.13b}$$

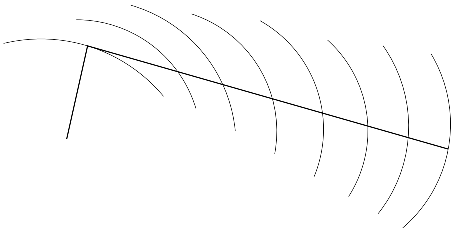


Figure 2.1: The search path follows the inverse gradient of $f(\mathbf{x}_n)$ until the line minimum is reached. There the gradient is orthogonal to the search direction. Curved lines are contour lines (i.e. constant f).

A simple recursion relation can be derived for successive search directions:

$$\begin{aligned} \mathbf{r}_{n+1} &= -\nabla f(\mathbf{x}_{n+1}) = \mathbf{b} - \boldsymbol{A}\mathbf{x}_{n+1} = \underbrace{\mathbf{b} - \boldsymbol{A}\mathbf{x}_n}_{\mathbf{r}_n} - \lambda_n \boldsymbol{A}\,\mathbf{r}_n \\ \Rightarrow\ \mathbf{r}_{n+1} &= \mathbf{r}_n - \lambda_n \boldsymbol{A}\,\mathbf{r}_n. \end{aligned} \tag{2.14}$$

We use the orthogonality of search directions to calculate λ_n:

$$0 \ = \ \mathbf{r}_{n+1}^T \mathbf{r}_n = \mathbf{r}_n^T \mathbf{r}_n - \lambda_n \boldsymbol{A}\,\mathbf{r}_n^T \mathbf{r}_n \tag{2.15}$$

$$\Rightarrow\ \lambda_n = \frac{\mathbf{r}_n^T \mathbf{r}_n}{\mathbf{r}_n^T \boldsymbol{A}\mathbf{r}_n}. \tag{2.16}$$

In summary, the Steepest Descent algorithm is given by Algorithm 3.

Algorithm 3 Steepest descent

```
Choose an initial vector $\mathbf{x}_0$
$\mathbf{r}_0 = \mathbf{b} - \boldsymbol{A}\mathbf{x}_0$
for $n = 0$ to $n_{\max}$ do
    $\lambda_n = \frac{\mathbf{r}_n^T \mathbf{r}_n}{\mathbf{r}_n^T \boldsymbol{A}\mathbf{r}_n}$
    $\mathbf{x}_{n+1} = \mathbf{x}_n + \lambda_n \mathbf{r}_n$
    if converged then EXIT
    $\mathbf{r}_{n+1} = \mathbf{r}_n - \lambda_n \boldsymbol{A}\mathbf{r}_n$
end for
```

We illustrate Steepest Descent by the following simple example in $N = 2$ dimensions.

$$A = \begin{pmatrix} 0.001 & 0 \\ 0 & 0.01 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 0.001 \\ 0.002 \end{pmatrix}, \quad c = 0.0007. \tag{2.17}$$

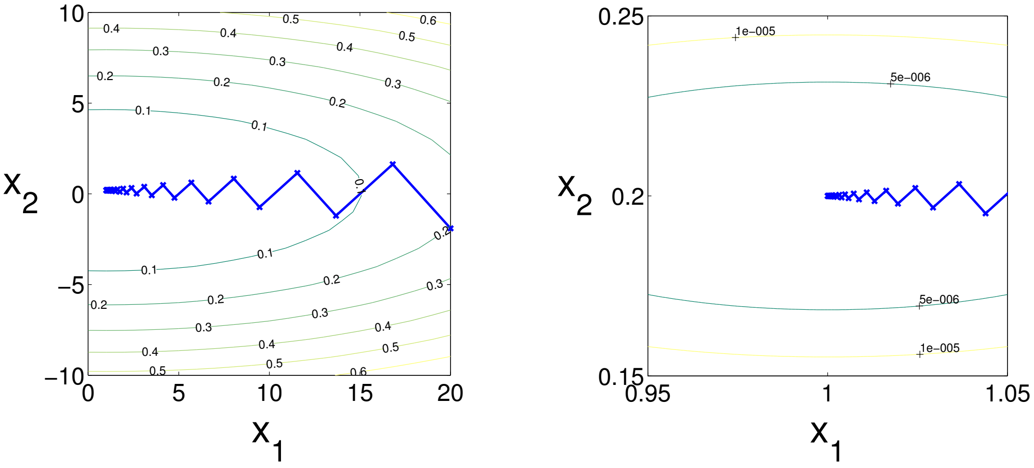


Figure 2.2: Steepest descent trajectory with initial point $(20, -1.9)$, for different resolutions. Curved lines are contour lines (i.e. constant f).

The initial starting point is chosen to be $(20, -1.9)$. This turns out to be an unfavorable example for steepest descent, because the eigenvalues of $\boldsymbol{A}$ are very different from each other.

The trajectories are plotted in Fig. 2.2 on different scales. The coordinates of the minimum are $\xi = (1, 0.2)$. A 6 digit accuracy takes 76 iterations. We see the oscillatory trajectory on all scales (panels) in figure 2.2. This is in strong contradiction to our general consideration that a good gradient based method ought to yield the exact solution in $\mathcal{O}(N)$ steps. The shortcoming is the orthogonality of the successive search directions. If the first gradient forms an angle of 45 degrees with the x_1-axis, then so do all following directions since they are orthogonal to each other. *Due to the asymmetric eigenvalues*, the contour lines are also strongly asymmetric and the line minimum is always close to the $(x_2 = 0)$-axis. Therefore each step cannot get far ahead on the x_1-axis towards the true minimum.

In the most favorable case, namely for spherical contour lines, i.e. equal eigenvalues for both principle axes of the matrix $\boldsymbol{A}$, the iteration would reach the exact solution within one step.

The relevant parameter determining how slowly the solution converges is the ratio of the largest and the smallest eigenvalues of the matrix $\boldsymbol{A}$, which is called the *condition number* of $\boldsymbol{A}$.

In order to overcome the problem of orthogonality of locked search directions, more general directions are required which account for any non-spherical metric. This is achieved in the method of *conjugate gradients*.

2.4 Conjugate Gradient Method

Conjugate Directions

Steepest Descent often finds itself taking steps in the same direction as earlier steps. The idea is now to find a suitable set of N orthogonal *search directions* $\mathbf{d}_0$, $\mathbf{d}_1$, $\ldots$, $\mathbf{d}_{N-1}$ (they will be ”A-orthogonal”). Together they form a *basis* of $\mathbb{R}^N$, so that the solution vector $\mathbf{x}$ can be expanded as

$$\mathbf{x} \;=\; \mathbf{x}_0 \;+\; \sum_{i=0}^{N-1} \lambda_i \, \mathbf{d}_i \;. \tag{2.18}$$

The goal is to take exactly one step in each search direction, with the right coefficient λ_i. After N steps we will be done. To accomplish this with low computational effort will require a special set of directions, and a new scalar product redefining "orthogonality".

For the first step, we set $\mathbf{x}_1 = \mathbf{x}_0 + \lambda_0 \mathbf{d}_0$. At step $n+1$ we choose a new point at the line minimum

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \lambda_n \mathbf{d}_n \tag{2.19}$$

etc., until $\mathbf{x}_N = \mathbf{x}$ after N steps.

Let us collect a few simple relations which follow directly from (2.18) and (2.19). It is useful to introduce the deviation (”error vector”) from the exact solution $\mathbf{x}$ (which contains the steps yet to be done)

$$\begin{aligned} \mathbf{e}_{n+1} \;&:=\; \mathbf{x}_{n+1} - \mathbf{x} && \text{(2.20a)}\\ &=\; \mathbf{e}_n \;+\; \lambda_n \mathbf{d}_n && \text{(2.20b)}\\ &=\; -\sum_{i=n+1}^{N-1} \lambda_i \, \mathbf{d}_i \;. && \text{(2.20c)} \end{aligned}$$

Similar relations hold for the residuals $\mathbf{r}_n$, namely

$$\begin{aligned} \mathbf{r}_{n+1} \;&:=\; -\nabla f(\mathbf{x}_{n+1}) \;=\; \mathbf{b} - \boldsymbol{A}\mathbf{x}_{n+1} && \text{(2.21a)}\\ &=\; \underbrace{\mathbf{b} - \boldsymbol{A}\mathbf{x}}_{=0} \;-\; \boldsymbol{A}\mathbf{e}_{n+1} && \text{(2.21b)}\\ &=\; \boldsymbol{A} \sum_{i=n+1}^{N-1} \lambda_i \, \mathbf{d}_i \;, && \text{(2.21c)} \end{aligned}$$

and also the recursion

$$\mathbf{r}_{n+1} = \mathbf{r}_n \;-\; \boldsymbol{A}\, \lambda_n \mathbf{d}_n \;. \tag{2.21d}$$

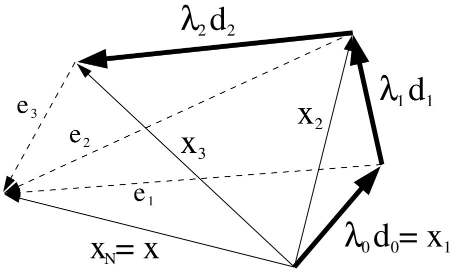


Figure 2.3: Sketch of the expansion of the solution $\mathbf{x}$ in directions $\mathbf{d}_i$ (with $x_0 = 0$). This figure is a projection from a higher dimensional space; therefore there are more than two "orthogonal" directions d_i.

The expansion is sketched in Fig.2.3.

Let us now see what would happen if we would demand that the directions d_n obey the usual 90 degree orthogonality $\mathbf{d}_i^T\mathbf{d}_j = 0$ for $i \neq j$. We want the search direction $\mathbf{d}_n$ to be orthogonal to all other search directions, including all future directions making up the error vector $\mathbf{e}_{n+1}$:

$$0 \overset{?}{=} \mathbf{d}_n^T\mathbf{e}_{n+1} \overset{(2.20b)}{=} \mathbf{d}_n^T\mathbf{e}_n + \mathbf{d}_n^T\lambda_n\mathbf{d}_n.$$

This would imply

$$\lambda_n \overset{?}{=} -\frac{\mathbf{d}_n^T\mathbf{e}_n}{\mathbf{d}_n^T\mathbf{d}_n} .$$

However, this equation is not useful, because this λ_n cannot be calculated without knowing the $\mathbf{e}_n$; but if we knew $\mathbf{e}_n$, then the problem would already be solved. This solution for λ_n would also mean that we totally ignore the line minima during search (e.g. when we try to reach the solution in Fig.2.1 with 2 vectors that are at 90 degrees to each other.)

Line minimum

The successful strategy is instead to still *follow each search direction* $\mathbf{d_i}$ *until the line minimum* ist reached. Thus

$$\begin{aligned} 0 &\overset{!}{=} \frac{d}{d\lambda}f(\mathbf{x}_{n+1}) = \nabla f(\mathbf{x}_{n+1})^T\frac{d}{d\lambda}\mathbf{x}_{n+1} \\ &= -\mathbf{r}_{n+1}^T\mathbf{d}_n \qquad (2.22) \\ &\overset{(2.21d)}{=} -\mathbf{d}_n^T\mathbf{r}_n + \mathbf{d}_n^T\boldsymbol{A}\mathbf{d}_n\,\lambda_n \end{aligned}$$

and we get

$$\lambda_n = \frac{\mathbf{d}_n^T\mathbf{r}_n}{\mathbf{d}_n^T\boldsymbol{A}\mathbf{d}_n}. \qquad (2.23)$$

This equation for the line minimum is valid for any set of search directions. It contains the current search direction $\mathbf{d_n}$ and the current residuum $\mathbf{r}_n$, which are both known.

Orthogonality

Eq. (2.22) tells us again that at the line minimum, the current search direction is orthogonal to the residuum. We rewrite this equation:

$$0 \;=\; -\mathbf{d}_n^T\mathbf{r}_{n+1} \;\overset{(2.21c)}{=}\; -\mathbf{d}_n^T\boldsymbol{A}\sum_{i=n+1}^{N-1}\lambda_i\,\mathbf{d}_i \;\overset{(2.20c)}{=}\; \mathbf{d}_n^T\boldsymbol{A}\,\mathbf{e}_{n+1}\;. \tag{2.24}$$

Our goal is that all search directions are "orthogonal". As argued above, this means that $\mathbf{d}_n$ should be orthogonal to $\mathbf{e}_{n+1}$. This is consistent with (2.24), if we **use a new scalar product**

$$(\mathbf{u},\mathbf{v})_{\boldsymbol{A}} \;:=\; \mathbf{u}^T\,\boldsymbol{A}\,\mathbf{v}\;. \tag{2.25}$$

to define orthogonality.

We now *demand* that all search directions are mutually *"A-orthogonal"*

$$\boxed{\mathbf{d}_i^T\boldsymbol{A}\mathbf{d}_j = 0 \quad (i \neq j)} \tag{2.26}$$

We will construct such a set of *"conjugate"* directions $\mathbf{d}_i$. Since $\mathbf{u}^T\,\boldsymbol{A}\,\mathbf{v}$ is a scalar product, these vectors form an orthogonal *basis* of $\mathbb{R}^N$, in which the solution vector $\mathbf{x}$ can be expanded as in (2.18).

We are therefore also guaranteed that the solution will take *at most N steps* (up to effects of rounding errors).

From (2.21c) and (2.26) we can deduce

$$\mathbf{d}_m^T\mathbf{r}_n \;=\; \mathbf{d}_m^T\,\boldsymbol{A}\sum_{i=n}^{N-1}\lambda_i\,\mathbf{d}_i \;=\; 0 \;\text{ for } m < n\;, \tag{2.27}$$

meaning that the residual $\mathbf{r}_n$ is orthogonal in the usual sense (90 degrees) to all old search directions.

Gram-Schmidt Conjugation

We still need a set of N $\boldsymbol{A}$-orthogonal search directions $\{\mathbf{d}_i\}$.There is a simple (but inefficient) way to generate them iteratively: The *conjugate Gram-Schmidt process*.

Let $\{\mathbf{u}_i\}$ with $i = 0,\ldots,N-1$ be a set of N linearly independent vectors, for instance unit vectors in the coordinate directions. Suppose that the search directions $\mathbf{d}_k$ for $k < i$ are already mutually $\boldsymbol{A}$-orthogonal. To construct the next direction $\mathbf{d}_i$, take $\mathbf{u}_i$ and subtract out all components that are not $\boldsymbol{A}$-orthogonal to the previous $\mathbf{d}$-vectors, as is demonstrated in Fig. 2.4. Thus, we set $\mathbf{d}_0 = \mathbf{u}_0$ and for $i > 0$ we choose

$$\mathbf{d}_i = \mathbf{u}_i + \sum_{k=0}^{i-1}\beta_{ik}\mathbf{d}_k, \tag{2.28}$$

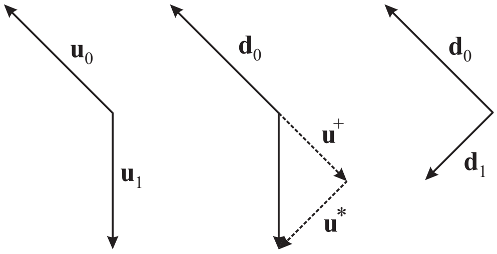


Figure 2.4: Gram-Schmidt conjugation of two vectors. Begin with two linearly independent vectors $\mathbf{u}_0$ and $\mathbf{u}_1$. Set $\mathbf{d}_0 = \mathbf{u}_0$. The vector $\mathbf{u}_1$ is composed of two components: $\mathbf{u}^*$ which is $\boldsymbol{A}$-orthogonal (in this figure: 90 degrees, in general: A-orthogonal) to $\mathbf{d}_0$, and $\mathbf{u}^+$ which is parallel to $\mathbf{d}_0$. We subtract $\mathbf{u}^+$, so that only the $\boldsymbol{A}$-orthogonal portion remains: $\mathbf{d}_1 = \mathbf{u}^* = \mathbf{u}_1 - \mathbf{u}^+ =: \mathbf{u}_1 + \beta_{10}\mathbf{d}_0$.

with the β_{ik} defined for $k < i$. To find their values we impose the $\boldsymbol{A}$-orthogonality of the new direction $\mathbf{d}_i$ with respect to the previous ones:

$$\begin{aligned} 0 \overset{i>j}{=} \mathbf{d}_i^T \boldsymbol{A}\mathbf{d}_j &= \mathbf{u}_i^T \boldsymbol{A}\mathbf{d}_j + \sum_{k=0}^{i-1} \beta_{ik}\mathbf{d}_k^T \boldsymbol{A}\mathbf{d}_j \\ &= \mathbf{u}_i^T \boldsymbol{A}\mathbf{d}_j + \beta_{ij}\mathbf{d}_j^T \boldsymbol{A}\mathbf{d}_j \; . \qquad (2.29) \\ \beta_{ij} &= -\frac{\mathbf{u}_i^T \boldsymbol{A}\mathbf{d}_j}{\mathbf{d}_j^T \boldsymbol{A}\mathbf{d}_j} . \qquad (2.30) \end{aligned}$$

Note that in the first line, the sum reduces to the term $k = j$ because of mutual $\boldsymbol{A}$-orthogonality of the previous search vectors $\mathbf{d}_k$, $k < i$.

Equation (2.30) provides the necessary coefficients for (2.28). However, there is a difficulty in using this method, namely that all the old search vectors must be kept and processed to construct the new search vector.

Construction from Gradients

This method is an efficient version of Gram-Schmidt. We will not need to keep the old search vectors. The new ansatz here is to choose a specific set of directions $\mathbf{u}_i$, namely

$$\mathbf{u}_i := \mathbf{r}_i \; . \qquad (2.31)$$

First we now use the fact that the residual is orthogonal to all previous search directions, (2.27). Together with (2.28) we get, for $i < j$ and for as yet general $\mathbf{u}_i$:

$$\begin{aligned} 0 \overset{i \le j}{=} \mathbf{d}_i^T \mathbf{r}_j &\overset{(2.28)}{=} \mathbf{u}_i^T \mathbf{r}_j + \underbrace{\sum_{k=0}^{i-1} \beta_{ik}\mathbf{d}_k^T \mathbf{r}_j}_{=0 \ (2.27)} \\ 0 &= \mathbf{u}_i^T \mathbf{r}_j \; . \qquad (2.32a) \end{aligned}$$

In the same way one gets

$$\mathbf{d}_i^T\mathbf{r}_i = \mathbf{u}_i^T\mathbf{r}_i. \tag{2.32b}$$

With our particular choice $\mathbf{u}_i = \mathbf{r}_i$, (2.32a) becomes

$$\mathbf{r}_i^T\mathbf{r}_j = 0, \quad i \neq j\ . \tag{2.33}$$

We see that all residue vectors $\mathbf{r}_i$ will actually be orthogonal to each other.

We can now compute the Gram-Schmidt coefficients (2.30). The recursion (2.21d) implies

$$\mathbf{r}_i^T\mathbf{r}_{j+1} \ = \ \mathbf{r}_i^T\mathbf{r}_j - \lambda_j\mathbf{r}_i^T\boldsymbol{A}\mathbf{d}_j\ .$$

The last term corresponds to the nominator in (2.30). Because of the orthogonality (2.33), it simplifies to

$$\mathbf{r}_i^T\boldsymbol{A}\mathbf{d}_j \ = \ \begin{cases} \frac{1}{\lambda_i}\mathbf{r}_i^T\mathbf{r}_i, & i = j \\ -\frac{1}{\lambda_{i-1}}\mathbf{r}_i^T\mathbf{r}_i, & i = j+1 \\ 0, & \text{otherwise,} \end{cases}$$

Thus we obtain all the coefficients needed in (2.28):

$$\beta_{ij} = \begin{cases} \frac{1}{\lambda_{i-1}}\,\frac{\mathbf{r}_i^T\mathbf{r}_i}{\mathbf{d}_{i-1}^T\boldsymbol{A}\mathbf{d}_{i-1}}, & i = j+1 \\ 0, & i > j+1. \end{cases}$$

Most of the β_{ij} terms have now become zero. It is no longer necessary to store old search vectors to ensure $\boldsymbol{A}$-orthogonality. We now denote, for simplification, $\beta_i := \beta_{i,i-1}$ and plug in λ_{i-1} from (2.23) to get the final form of the coefficients:

$$\begin{aligned} \beta_i \ &\overset{(2.23)}{=} \ \frac{\mathbf{r}_i^T\mathbf{r}_i}{\mathbf{d}_{i-1}^T\mathbf{r}_{i-1}} \\ &\overset{(2.32b)}{=} \ \frac{\mathbf{r}_i^T\mathbf{r}_i}{\mathbf{r}_{i-1}^T\mathbf{r}_{i-1}}. \end{aligned}$$

Putting all our results together we obtain the *Conjugate Gradient algorithm*, which is presented in symbolic form as algorithm 4.

Comments

Note that the first step of the Conjugate Gradient method is the same as for steepest descent. Convergence is again estimated by the norm of $\mathbf{r}_n$.

The computationally most expensive part of the algorithm is one Matrix-vector multiplication $\boldsymbol{A}\mathbf{d}_n$ per step. It can be implemented efficiently, especially for sparse matrices. All other parts are vector-vector multiplications. Because of rounding errors, orthogonality can get lost during the iteration.

Algorithm 4 Conjugate Gradient method for quadratic functions

```
Choose a suitable initial vector $\mathbf{x}_0$
Set $\mathbf{d}_0 := \mathbf{r}_0 = -\nabla f\big|_{\mathbf{x}_0} = \mathbf{b} - \boldsymbol{A}\mathbf{x}_0$
for $n = 0$ to $N$ do
    $\mathbf{a}_n = \boldsymbol{A}\mathbf{d}_n$
    $\lambda_n = (\mathbf{r}_n^T \mathbf{r}_n)/(\mathbf{d}_n^T \mathbf{a}_n)$
    $\mathbf{x}_{n+1} = \mathbf{x}_n + \lambda_n \mathbf{d}_n$
    $\mathbf{r}_{n+1} = \mathbf{r}_n - \lambda_n \mathbf{a}_n$
    $\beta_{n+1} = (\mathbf{r}_{n+1}^T \mathbf{r}_{n+1})/(\mathbf{r}_n^T \mathbf{r}_n)$
    $\mathbf{d}_{n+1} = \mathbf{r}_{n+1} + \beta_{n+1} \mathbf{d}_n$
    if converged then EXIT
end for
```

It can therefore be advantageous to occasionally calculate the residue directly as $\mathbf{r}_{n+1} = \mathbf{b} - \boldsymbol{A}\mathbf{x}_{n+1}$, which involves a second matrix multiplication.

In each step, the Conjugate Gradient algorithm invokes one multiplication with $\boldsymbol{A}$. The solution is thus in fact constructed in the space spanned by the vectors $\{\mathbf{r}_0, \boldsymbol{A}\mathbf{r}_0, \boldsymbol{A}^2\mathbf{r}_0, \boldsymbol{A}^3\mathbf{r}_0, \dots\}$, which is called a *Krylov space*. There are a many other related methods also acting in Krylov space. One of the most important ones is the closely related *Lanczos algorithm* for computing low lying eigenvalues of a big (sparse) matrix. It was developed before CG. Other methods exist for nonsymmetric matrices.

The numerical stability and convergence of CG is again governed by the *condition number* of the matrix $\boldsymbol{A}$. It can be improved greatly by a class of transformations called *Preconditioning*. Instead of solving $\boldsymbol{A}\mathbf{x} = \mathbf{b}$, one solves the equivalent equation $\boldsymbol{M}^{-1}\boldsymbol{A}\mathbf{x} = \boldsymbol{M}^{-1}\mathbf{b}$. The ideal case would be the exact solution $\boldsymbol{M}^{-1} = \boldsymbol{A}^{-1}$, giving the identity matrix $\boldsymbol{M}^{-1}\boldsymbol{A}$ with condition number unity. Much simpler transformations can already improve convergence greatly, e.g. just taking $\boldsymbol{M}$ to be the diagonal of $\boldsymbol{A}$.

2.5 Conjugate Gradient for General Functions

We are now prepared to apply the conjugate gradient idea to general functions $f(\mathbf{x})$. To this end we expand $f(\mathbf{x})$ in each iteration about the actual reference point $\mathbf{x}_n$ of the n^{th} step, to second order

$$f(\mathbf{x}) \simeq f(\mathbf{x}_n) + \mathbf{b}_n^T(\mathbf{x} - \mathbf{x}_n) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_n)^T \boldsymbol{A}_n (\mathbf{x} - \mathbf{x}_n) \tag{2.34a}$$

$$\mathbf{b}_n = \nabla f(\mathbf{x})\big|_{\mathbf{x}_n} \tag{2.34b}$$

$$(\boldsymbol{A}_n)_{ll'} = \frac{\partial^2}{\partial x_l \partial x_{l'}} f(\mathbf{x})\big|_{\mathbf{x}_n}, \tag{2.34c}$$

where the matrix $\boldsymbol{A}_n$ is called the Hessian. Since (2.34) is a quadratic form it can easily be cast into the form (2.8), on which CG (Conjugate Gradient) for quadratic problems was based, with one significant modification though: The matrix (Hessian) $\boldsymbol{A}_n$ changes from iteration to iteration as does the vector $\mathbf{b}_n$. All equations in algorithm 4 are still valid with the only modification that the matrix $\boldsymbol{A}$ corresponds to the Hessian $\boldsymbol{A}_n$ of the Taylor expansion about the reference point $\mathbf{x}_n$. The iteration scheme 4 remains valid, with the modification that $\boldsymbol{A}$ has to be replaced in each iteration by the respective Hessian $\boldsymbol{A}_n$. This implies that we have to be able to compute the Hessian. Is is available analytically only in rare cases, and the numeric computation is inefficient. Fortunately, $\boldsymbol{A}_n$ enters only in conjunction with the iteration scheme for the gradients and in the expression for λ_n. These steps can be modified. First of all, we replace the update rule for $\mathbf{r}_{n+1}$ by the definition

$$\mathbf{r}_{n+1} = -\nabla f(\mathbf{x})\big|_{\mathbf{x}_{n+1}}.$$

Here we merely require the knowledge of the gradient instead of the Hessian. Secondly, the parameter λ_n is obtained more directly via

$$\min_{\lambda} \; f(\mathbf{x}_n + \lambda \mathbf{d}_n) \;\Rightarrow \lambda_n,$$

which is approximately solved numerically. Equations (2.26), (2.27), and (2.33) are, however, no longer valid for $|i - j| > 1$, since the matrix $\boldsymbol{A}$ changes from iteration to iteration. If the conjugacy relation (2.26) was still valid then the arguments given for quadratic problems would still hold and convergence would be reached within at most N steps. This is, of course, not the case for arbitrary non-quadratic functions.

The corresponding algorithm is presented below.

Algorithm 5 Conjugate Gradient method for general functions

```
Choose a suitable initial vector $\mathbf{x}_0$
$\mathbf{d}_0 = \mathbf{r}_0 = -\nabla f\big|_{\mathbf{x}_0}$
**for** $n = 0$ to $n_{\max}$ **do**
    $\min_\lambda \; f(\mathbf{x}_n + \lambda \mathbf{r}_n) \quad \Rightarrow \lambda_n$
    $\mathbf{x}_{n+1} = \mathbf{x}_n + \lambda_n \mathbf{d}_n$
    $\mathbf{r}_{n+1} = -\nabla f\big|_{\mathbf{x}_{n+1}}$
    $\beta_{n+1} = (\mathbf{r}_{n+1}^T \mathbf{r}_{n+1})/(\mathbf{r}_n^T \mathbf{r}_n)$
    $\mathbf{d}_{n+1} = \mathbf{r}_{n+1} + \beta_{n+1} \mathbf{d}_n$
    **if** converged **then** EXIT
**end for**
```

2.6 Stochastic Optimization

Nonlinear problems often have more than one minimum. Gradient methods like Conjugate Gradient (CG) only yield a *local* minimum in the vicinity of the initial point. If there are not too many minima, one can start CG at different initial points (chosen at random) to obtain the global minimum. However, there are many problems not accessible to this procedure. If there are many minima, then CG with random start points is very inefficient.

Combinatorial problems. Traveling Salesman.

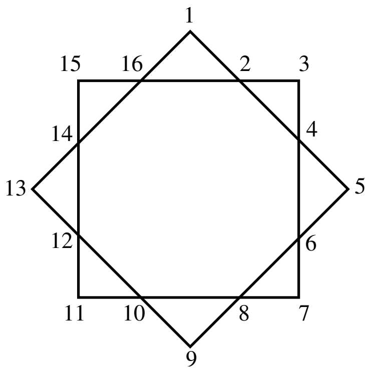
Some examples that are not suited for gradient-based methods are the following combinatorial problems:

- The *Traveling Salesman* problem (TSP). There are N cities. A salesman has to visit each city once, and the length of the trip has to be minimized. The function to be minimized is of the form

  $$f(\{i\}) = \sum_{\nu=1}^{N} \left|\mathbf{x}_{i_{\nu+1}} - \mathbf{x}_{i_\nu}\right|, \quad (i_{N+1} := i_1)$$

  with $\left|\mathbf{x}_{i_{\nu+1}} - \mathbf{x}_{i_\nu}\right|$ the distance between two consecutive cities within the list $\{i\}$. The locations $\mathbf{x}_i$ of the cities are fixed. The total distance travelled depends on the *order* in which the cities are visited. The notation $\{i\}$ stands for a list of indices in this order. The number of possible sequences is $N!$. For large N, they cannot be enumerated directly.

  More realistic applications contain additional terms and constraints in the cost function.



- Redistribute the integer numbers from $1-16$ in a way that the sums along all edges have the same value. There are $16!$ possible index arrangements.
- Time tables at schools, etc.

Note that in these combinatorial examples, there is a *discrete* set of possibilities. Therefore the function to be minimized does not have any derivative in the space of possibilities, and methods like Conjugate Gradient cannot be applied.

Complexity classes

Combinatorial problems (and related others like search problems) can be extremely hard computationally. There is a large area of computer science dealing with just the *classification* of such problems. One of the "simplest" complexity classes is called "**P**" the class of all problems which can be solved in *polynomial time*, i.e. the number of steps to solve the problem grows with the "size" (like the number N of cities in the traveling salesman problem) "only" polynomially. The Traveling Salesman probably does not belong to this class, which means that there are cases which require an *exponential* number of steps.

A more difficult fundamental class is called **NP**. It is the class of problems for which the solution, once specified, can be *verified* in polynomial time. This definition says nothing about how long it may take to find the solution, which might be exponentially long. (Note that NP stands for *non-deterministic polynomial* because such problems could be solved on a non-deterministic Turing machine (which can branch at every step) in polynomial time. It does *not* stand for "non-polynomial".)

The most difficult problems in NP are called **NP-complete**, namely those to which every other problem in NP can be transformed in polynomial time. All NP-complete problems are equivalent in this sense. It is known that the Traveling Salesman is in fact an *NP-complete* problem, as are many other problems in graph-theory, networks, sorting, partitioning, data compression, etc..

It is very likely (but not proven) that NP is larger than P, i.e. that there are indeed problems which can not be solved in polynomial time. If NP where the same as P, then there would be a polynomial time algorithm for *all* the myriad important problems known to be NP-complete.

There are other problems which are known to be outside P, and others which are not solvable even on a quantum computer (called **QMA**). In fact, there are so many proposed complexity classes that one speaks of the *complexity zoo*. However, for almost all of these classes, their supposed relationship is not mathematically *proven* ! One does not even know for sure whether there are problems that a quantum computer can solve faster (in the sense of polynomial versus non-polynomial) than a classical one.

It is also important to note that the classification is in terms of *worst case* problems of a given type. A "typical" or an "average" instance of a problem class may be much easier to solve. For example, the Traveling Salesman problem can sometimes be solved for thousands of cities. The situation becomes still better when one allows approximations to the optimum.

On the other hand, problems with "polynomial time" solutions may in practice involve such high powers of N that they become intractable - which may already be the case for N^3 when N is very large.

There is a great variety of optimization methods for such difficult problems. We will introduce some of the most fundamental ones.

No method is perfect

No method provides a guarantee to find the global optimum of an arbitrary function, except by exhaustive search (or analytic solution). In order to gain some confidence in any solution, it is therefore always advisable to perform several independent optimizations, with different starting points and different random number sequences.

It is important to note also that in many cases one wants to find a good **robust solution** insensitive to small parameter fluctuations rather than the true global optimum of the cost function.

2.6.1 Classical Simulated Annealing

Literature:

- S. KIRKPATRICK, C.D. GELLAT, JR., and M.P. VECCHI, *Simulated Annealing*, Science **220**, 671 (1983).

The most well known stochastic optimization algorithm is Simulated Annealing. Its approach is borrowed from physics, specifically from thermodynamics. The method models the way that liquids freeze and crystallize during an annealing process. At high temperatures, the particles can move freely. Differences of potential energies of various configurations are overcome by the particles due to their high kinetic energy. When the liquid is cooled down slowly, an ordering process sets in, thermal mobility is lost and some optimal configuration (e.g. an ideal crystal) may be achieved. This is a configuration where the cost function (free energy) attains its absolute minimum. However, if the liquid is cooled down too quickly, no ideal crystal is obtained and the systems ends in some local minimum of the energy (meta-stable state). This corresponds to a poly crystalline or amorphous state. We try to find the global minimum by simulating the slow cooling process on the computer. To this end we introduce an *artificial temperature* T and an artificial (!) *probability distribution* $p_E(\mathbf{x}|T)$, where x now is some point in the parameter space of the problem. The cost function to be minimized is denoted by $f(\mathbf{x})$ and can depend on a set of variables x that is either discrete or continuous. In Classical Simulated Annealing (CSA), we *choose*

$$p_E(\mathbf{x}|T) = \frac{1}{Z} e^{-f(\mathbf{x})/T}, \tag{2.35}$$

which corresponds to a Boltzmann distribution. Thus we treat f like an *energy* in statistical mechanics. Here Z is a normalization factor which corresponds to the partition function of the canonical ensemble, $Z = \sum_{\mathbf{x}} e^{-f(\mathbf{x})/T}$.

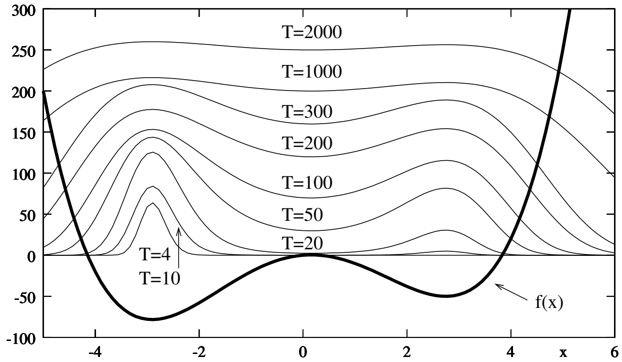


Figure 2.5: Example of a cost function $f(x)$ with two minima (thick solid line) and the Boltzmann weight $p_E(x|T)$ (thin solid lines, without scale) for various temperatures T.

We can also define the expectation value of $f(\mathbf{x})$ at a given temperature T by

$$\langle f\rangle_T = \sum\!\!\!\!\!\!\!\int_{\mathbf{x}} p_E(\mathbf{x}|T)\, f(\mathbf{x}). \tag{2.36}$$

In order to find the minimum, we will employ a Monte Carlo simulation with the weight function (2.35). As an example, figure 2.5 shows the cost function (thick solid line)

$$f(x) = x^4 - 16x^2 + 5x\ . \tag{2.37}$$

Figure 2.5 also depicts the Boltzmann weight (thin solid lines) for various temperatures. At temperatures much higher than the potential barrier, $T \gg 50$, the Boltzmann distribution is rather flat. As soon as the temperature is lower than the potential barrier, e.g. at $T = 20$, the weight outside the double-well is almost zero and two separate peaks develop. If the temperature is even lower than the difference of the two minima, e.g. at $T = 10$, the weight in the local minimum becomes negligible and the probability is concentrated merely around the global minimum.

It is the purpose of the distribution p_E to assign a high probability to states x where $f(\mathbf{x})$ is small, i.e. states x close to the global minimum. A *"random walker"* will spend a long time near the optimum state. The important difference with respect to gradient based methods like CG is the possibility for the walker to also go *uphill* and thus leave a local minimum. The probability to do so depends on temperature. The minimum of $f(\mathbf{x})$ as well as the corresponding state x are found by cooling.

An implementation of the Classical Simulated Annealing (CSA) method requires the following three parts

1) Move in configuration space, 2) Probability Distribution and Acceptance of states, and 3) A cooling scheme

ad 1: Move in configuration space In the vicinity of the current point $\mathbf{x}_n$ (n is the step index) we create a trial point $\mathbf{x}_t$ at random.

For *continuous* problems, the probability distribution of the trial point $\mathbf{x}_t$ (i.e. the proposal probability) is typically chosen to be a Gaussian

$$q_x(\mathbf{x}_t - \mathbf{x}_n) \propto \prod_i e^{-(\mathbf{x}_t - \mathbf{x}_n)_i^2/(2\sigma^2)} \tag{2.38}$$

of a certain variance σ around the current point $\mathbf{x}_n$. If the curvature of $f(\mathbf{x})$ in various directions differs significantly, it is expedient to use a p_x which employs different variances in different directions, or, more generally, the covariance matrix $\boldsymbol{C}$ of $f(\mathbf{x})$ instead of the variance σ:

$$q_x(\mathbf{x}_t - \mathbf{x}_n) \propto \exp\left\{\frac{1}{2}(\mathbf{x}_t - \mathbf{x}_n)\boldsymbol{C}^{-1}(\mathbf{x}_t - \mathbf{x}_n)\right\}.$$

As an example for a *discrete* problem, we look a the Traveling Salesman. The tour is represented by a list of indices $\{i_1, i_2, \ldots, i_N\}$ indicating the order in which the cities are visited. One possibility is the so-called *lin-2-opt* move. It consists in reversing a part of the trajectory:

$$\{i_1 \ldots i_{\nu-1}, \underline{i_\nu, i_{\nu+1} \ldots i_{\mu-1}, i_\mu}, i_{\mu+1} \ldots i_L\} \longrightarrow \{i_1 \ldots i_{\nu-1}, \underline{i_\mu, i_{\mu-1} \ldots i_{\nu+1}, i_\nu}, i_{\mu+1} \ldots i_L\}. \tag{2.39}$$

The advantage of such a move is that the change in cost function can be small and is computed quickly. Here it is

$$f(\{i'\}) - f(\{i\}) = \overline{\mathbf{x}_{i_{\nu-1}}\mathbf{x}_{i_\mu}} + \overline{\mathbf{x}_{i_\nu}\mathbf{x}_{i_{\mu+1}}} - \overline{\mathbf{x}_{i_{\nu-1}}\mathbf{x}_{i_\nu}} - \overline{\mathbf{x}_{i_\mu}\mathbf{x}_{i_{\mu+1}}},$$

with $\overline{\mathbf{x}_{i_\nu}\mathbf{x}_{i_\mu}} = \left|\mathbf{x}_{i_\nu} - \mathbf{x}_{i_\mu}\right|$ the distance between the cities ν and μ in configuration $\{i\}$.

ad 2: Probability Distribution and Acceptance of states We are free to choose any procedure which will lead us to the minimum, while providing the possibility to also move uphill. In Classical Simulated Annealing one chooses the Boltzmann Distribution (2.35). A common choice for the acceptance step is to use the *Metropolis probability*

$$p^{acc} = \min\left\{1, \frac{p_E(\mathbf{x}_t|T)}{p_E(\mathbf{x}_n|T)}\right\} \;=\; \min\left(1, e^{-\beta\,\Delta f}\right)\,, \tag{2.40}$$

which happens to satisfy detailed balance. We thus generate a Markov chain with which we can calculate expectation values *at a given temperature* T

$$\langle f(\mathbf{x})\rangle_T \;=\; \lim_{n\to\infty}\frac{1}{n}\sum_{i=1}^{n} f(\mathbf{x}_i) \tag{2.41}$$

by averaging over the Markov chain at that T.

Real problems often display several different (length) *scales in parameter space*. Consider as an example the traveling salesman. Towns will typically come in several agglomerations. The distance from one town to another strongly depends on whether the town is in the same agglomeration or not. Thus we have two different length scales:

- Distance between towns of the same agglomeration.
- Distance between different agglomerations.

The existence of different scales is reflected by the way the "energy" decreases when we lower the temperature. At extremely high temperatures, agglomerations do not play a role. The salesman travels at random. In the cooling procedure, the order of the agglomerations to be visited will be decided on first. Only then the trip inside each agglomeration is fixed.

In order to learn more about the behaviour of the system, we use the analogy with statistical physics. We denote the cost function $f(\{i\})$ explicitely as an *energy* $E(\{i\}) \equiv f(\{i\})$. During the simulation for one specific temperature T we calculate

$$\langle E\rangle \simeq \frac{1}{n}\sum_{\{i\}} E(\{i\}), \quad \text{and} \quad \langle E^2\rangle \simeq \frac{1}{n}\sum_{\{i\}} E^2(\{i\}),$$

with the variance

$$\langle \Delta E^2\rangle \ := \ \langle E^2\rangle - \langle E\rangle^2.$$

Here, $\sum_{\{i\}}$ indicates the sum over configurations simulated at the current temperature. This allows us to introduce a *specific heat* C_H defined as

$$C_H \ := \ \frac{\partial\langle E\rangle}{\partial T} = \frac{\langle \Delta E^2\rangle}{T^2}. \tag{2.42}$$

The last equality follows from (2.35) and (2.36). The specific heat is large when the energy changes quickly as a function of temperature (see fig. 2.6). In a statistical physics system this is typically associated with a phase transition. We get a first cooling rule: Away from "phase transitions" we can cool down quickly, since the energy (and thus the Boltzmann weight) varies slowly. However, we have to be careful in the vicinity of such "phase transitions", and the specific heat C_H with its rapid variation around phase transitions is a good indicator of critical regions.

ad 3: Cooling Strategy An important step is the choice of the initial temperature T_0. At this temperature it should be possible to cover the best part of the configuration space and it is a *rough* rule of thumb that at this temperature at least 80% of the configurations should be accepted. This can easily be achieved by choosing some value for T_0 and performing n steps. When more than 80% of the steps are accepted, then T_0 may be a good choice, otherwise we double T_0 and try again.

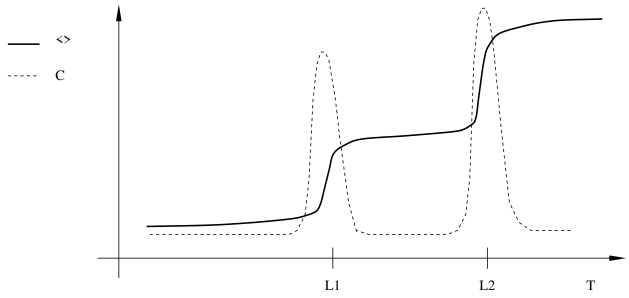


Figure 2.6: Cost function (solid line) and "specific heat" (dotted line) of the traveling salesman problem with two length scales L_1 and L_2

To proceed in the simulation, we let the walker do a n steps at every fixed temperature T_k.

One simple choice for the temperature sequence is to lower it according to a simple formula, e.g.

$$T_k = \frac{T_0}{k^q} \tag{2.43}$$

with some exponent q. Sometimes even a very fast exponential cooling like $T_k = T_0\, q^k$ is used. A fixed cooling sequence has the drawback of not taking into account any "phase transitions", where the simulation could get stuck more easily.

One can devise a more adaptive cooling strategy which takes into account the rate of change of the function f, i.e. the "specific heat". We need to make sure that the walker can move *back and forth* in energy space. Therefore the energies in the canonical distribution of configurations at temperature T_k should overlap those in the canonical distribution at the next temperature T_{k+1}. The widths of these distributions is related to the specific heat and one can show that the step in temperature should be bounded

$$\frac{T_k - T_{k+1}}{T_k} < \frac{\delta}{\sqrt{C_H}} \tag{2.44}$$

where δ is a small number related to the desired acceptance rate.

During the annealing process it is expedient to choose the trial points in such a way that an acceptance rate of about 50% is achieved (in both directions). We can for example measure the acceptance rate during $N = 100$ steps. If it is below 40% we decrease the size of the moves, if it is above 60% we increase the size of the moves, otherwise the proposal distribution is acceptable. (Note that changing the proposal distribution during the update violates detailed balance, but does not disturb the search for the global minimum).

Algorithm 6 Classical Simulated Annealing (CSA)

```
Choose a suitable initial vector $\mathbf{x}_0$
$T_0 = T_0^E$
for $j = 0$ to $j_{\max}$ do
  for $l = 0$ to $n$ do
    generate trial state $\mathbf{x}_t$ according to $p_x(\mathbf{x}_t - \mathbf{x}_n)$
    compute $q = \min(1, p_E(\mathbf{x}_t)/p_E(\mathbf{x}_n))$
    if $q = 1$; $\mathbf{x}_{n+1} = x_t$; else
      random number $r \in [0, 1)$
      if $q > r$; $\mathbf{x}_{n+1} = \mathbf{x}_t$; else $\mathbf{x}_{n+1} = \mathbf{x}_n$;
  end for
  determine $T_{j+1}$
  if converged then EXIT
end for
```

2.6.2 Fast Simulated Annealing

One shortcoming of the CSA method is that due to the exponential character of the Boltzmann distribution, only *short moves*, or rather small modifications, are allowed. Therefore it takes a long time to escape from a local minimum.

Instead of the Gaussian (2.38), *Fast Simulated Annealing* (FSA) uses the CAUCHY function as a proposal probability distribution. The D-dimensional Cauchy distribution is given by

$$p_x(\Delta\mathbf{x}|T) = \frac{T}{[(\Delta\mathbf{x})^2 + T^2]^{\frac{D+1}{2}}}, \tag{2.45}$$

where D is the dimension of the vector $\mathbf{x}$ of parameters. Due to its long ranging tails (namely an unbounded variance), the Cauchy distribution has the advantage to allow occasionally larger changes in configuration space, while the short range moves are still Gaussian distributed with a variance $\sigma^2 = 2\,T^2/(D+1)$.

Formally, it has been shown that under some assumptions on the function to be minimized, the Gaussian proposal (2.38) may need a temperature schedule $T(t) \sim \frac{1}{\ln(t+1)}$, (which means an exponential number of steps !), whereas under the same assumptions the Cauchy distribution needs only $T(t) \sim \frac{1}{t+1}$.

In figure 2.7 b), CSA and FSA are compared based on the double well potential (2.37) discussed before. The temperature entering the proposal distribution is adjusted every 100 steps to ensure 50% acceptance rate. For each annealing step n, the lowest energy (value of the cost function) is stored in $E(n)$. The entire annealing procedure, covering $n_{\max} = 10000$ steps is repeated 1000 times and average $\langle E(n)\rangle$ is computed for each value of n separately. We see that FSA is indeed superior.

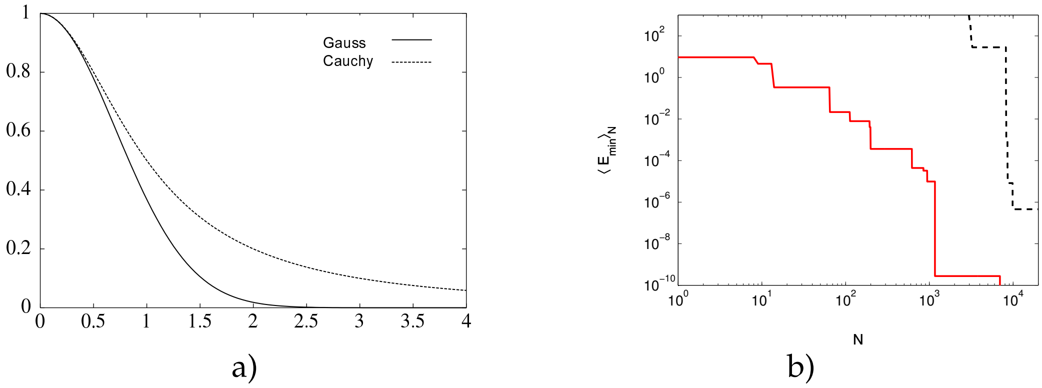


Figure 2.7: a) Comparison between Gaussian and Cauchy proposal distributions. b) Average minimum energy $\langle E_{\min} \rangle_N$reached in step N for Fast Simulated Annealing (FSA, solid red curve) and Classical Simulated Annealing (CSA, dashed black curve).

2.6.3 Other variants

Alternative Acceptance methods

Note that for optimization purposes one does not need detailed balance! (It does however help in order to map the region of attraction of the optimum, i.e. its stability.) The proposals mentioned here do *not* satisfy detailed balance nor stationarity. Thus they do not lead to a simultation of the canonical distribution of a statistical system. Nevertheless they are well suited for finding the minimum of f.

- *Threshold Acceptance:* A new configuration $\mathbf{x}_t$ is generated. If the cost function satisfies $f(\mathbf{x}_t) < f(\mathbf{x}_n) + T$, with T some tolerance level (threshold), then $\mathbf{x}_t$ is accepted. This allows rather effectively to leave local minima. During the iteration the threshold is continuously reduced.

- *Deluge Algorithms:* Accept new configurations $\mathbf{x}_t$ only if $f(\mathbf{x}_t) > T$ with T the acceptance level. T is continuously increased during the iterations; the landscape is 'flooded' until only the summits of the mountains, and finally only the summit of the biggest mountain is above the water level.

Multicanoncial ensemble

This method, describe in more detail in section 1.7.3, moves through parameter space efficiently by using an auxiliary cost function $\tilde{f}(\mathbf{x})$ which is almost constant. It is constructed from the actual cost function $f(\mathbf{x})$ by mapping and then "filling up" its local minima. Trapping can be avoided in this way, so that the global minimum of $f(\mathbf{x})$ can be found better. This method is quite successful for, e.g., protein folding.

Tempering

A severe drawback of Simulated Annealing is that the temperature is always lowered. Once the walker is stuck in a local minimum, it will stay there. In *Tempering* (section 1.7.3) the temperature is allowed to fluctuate randomly like any other parameter, between a very low and a very high bound. The temperature steps are not prescribed from outside, but instead decided with Metropolis decisions. When the temperature is high, a local minimum can be left. Tempering is therefore somewhat like an automatic multiple run of Simulated Annealing with many starting configurations and many temperature sequences.

2.7 Genetic algorithms

A genetic algorithm takes its approach from nature. It simulates the natural evolution of a *population* of individuals (candidate solutions) from generation to generation, with steps of inheritance, mutation, and selection. There are many varieties of such algorithms.

Each individual solution is characterized by a ″genome″, representing one point in the parameter space of the cost function (″*fitness*″) to be optimized. The genome is traditionally represented by a set of bits, but it can also be comprised of real or integer numbers, or more complicated objects. The cost function needs to be evaluated very often, and an efficient implementation is therefore important.

Initialization.

The optimization starts with an initial *population* of typically hundreds or thousands of individuals. They can be chosen at random, or according to initial guesses, in areas of parameter space where solutions are likely to be found. For each individual, the cost function is evaluated.

Iteration.

The following steps are then iterated:

- Selection. A part of the population is selected. This selection is based on fitness. However, it is not appropriate to select just the best solutions. One needs to include some randomness in the selection in order to insure genetic variety in the population. Otherwise the method will quickly tend towards a local optimum instead of the global one.

- Reproduction. Pairs of solutions are selected. The genomes are combined to generate one or more new solution, for example by a *crossover*, where up to some bit position, the bits from the first solution are taken, and then those from the second one.

- *Mutation.* Some information in the new solution is randomly mutated, often with carefully chosen probabilities and mutation steps.
- *Evaluation and Selection.* For each new solution, the cost function is evaluated. The new generation is then selected based on the cost function, either completely from the new generation, or including the previous one. It will on average have higher fitness than the previous generation.

Termination.

The above iteration is ended when a suitable condition has been reached. Common choices are

- A good enough solution is found.
- The quality of solutions has reached a plateau and does not improve any more.
- A maximum cost (number of iterations, computer time) has been reached.

Genetic algorithms can often find *good* solutions quickly, even when the cost function has a difficult shape.

Optimizing the method.

Similar to other methods, these algorithms have a *tendency towards a local optimum instead of the global one*. Of course this depends on the shape of the cost function. One needs to maintain sufficient genetic diversity to counteract such tendencies, for example by imposing a penalty for populations that are too homogeneous.

Genetic algorithms tend to be efficient at finding the neighborhood of an optimum solution, but less efficient at finding the optimum itself. It can be helpful to *combine* genetic evolution of a population with steps in which an optimum is approached more directly.

In general, the details of the algorithm, such as selection processes and mutations, as well as data representations should be *tuned* to the class of problems to be solved. For example, when the genome is specified by integer numbers, with the usual power-of-two representation in bits, then mutations of one or a few bits are likely to cause a catastrophic change of the cost function. It is much better to use *Gray coding*. In this representation of the integer numbers, sequential integers differ by only 1 bit (!). Mutations of one or a few bits will then lead to nearby integer numbers.

Applications.

Genetic algorithms are particularly good for complicated combinatorial problems. Common applications include the following

- Time tables and scheduling (including the Travelling Salesman problem).
- Design of networks (Electronic circuits, computer networks, locations of mobile-phone transmitters, water distribution, . . .)
- Molecular structure optimization, protein folding.
- Data fitting.
- Preparation of neural networks.

Related Methods.

- *Ant colony optimization* uses many agents (ants) to find local optima in parameter space and to move on from there.
- *Particle swarm optimization* simulates the way that e.g. a swarm of fish or a flock of birds communicates. Many individuals are simulated which fly through parameter space. In each step, the information on good solutions that an individual finds is communicated to its neighbors up to some distance. Each individual adjusts its course and speed to this information. Often the overall best solution found so far is communicated to each individual immediately in order to improve convergence.
- *Simulated Annealing* can be seen as a limiting case of a genetic algorithms, with a population of only one individual.

Optimization of average fitness.

In some situations it is appropriate to formulate the problem such that one has to optimize the *average* fitness of a population instead of the maximum fitness. An example is again the set of locations of mobile-phone transmitters. Instead of viewing the complete set as an "individual", and simulating a populations of such sets, one can declare each transmitter to be an individual, characterized by its location. An examples of algorithms which optimize the average fitness are *Bacteriological algorithms*, which take their clues from populations of bacteria.

Chapter 3

Molecular Dynamics

Literature

- D.FRENKEL, B. SMIT, *Understanding Molecular Simulation*, Academic Press, 2001
- J.M. THIJSSEN, *Computational Physics*, Cambridge University Press, 2007
- M. GRIEBEL ET AL,, *Numerische Simulationen in der Moleküldynamik*, Springer Verlag, 2004

An important field of research is to describe macroscopically observable properties of matter on the basis of microscopic kinematics and dynamics of molecules. However, the simultaneous motion of a large number of interacting bodies cannot be described analytically. One approach is to make simplifiying assumptions which allow a fast solution, like in thermodynamic calculations. Then it is hard to estimate the influence of those simplifications on the solutions. In complex situations, direct numerical simulations are a better alternative. There are essentially two approaches to determine physical quantities over a restricted set of states, namely Monte Carlo (MC), which samples from a static equilibrium ensemble and which we have treated before, and Molecular Dynamics (MD), which can also be applied in nonequilibrium situations.

3.1 Overview

3.1.1 Equations of motion

The general idea of Molecular Dynamics is to let a system evolve in time by *integrating the equations of motion* of many individual interacting particles together. It is widely used to study classical systems. With some approximations, quantum systems can be studied as well. (See also chapter 2.1).

One is for example interested in the motion of atoms in molecules and solids. Due to their much lighter mass, the electron dynamics is usually much faster than that of the nuclei and it is in most cases a good approximation to treat the dynamics of the nuclei classically (Born-Oppenheimer approximation), i.e. by Newton's equation of motion

$$\ddot{x}_{i\alpha}(t) \;=\; \frac{1}{m_i}\,F_{i\alpha}[\mathbf{r}(t),t] \;=:\; f_{i\alpha}[\mathbf{r}(t),t], \tag{3.1}$$

where $x_{i\alpha}$ is the α-th coordinate of particle i. The vector $\mathbf{r}$ stands for the collection of coordinates $\{x_{i\alpha}\}$. Similarly, we introduce the vector $\mathbf{f}$ for the set $\{f_{i\alpha}\}$, and get the equation of motion:

$$\ddot{\mathbf{r}}(t) = \mathbf{f}[\mathbf{r}(t),t]. \tag{3.2}$$

On an atomic scale, the *forces* acting on the nuclei originate from the direct Coulomb interaction between the nuclei plus the indirect contribution stemming from electrons. The latter are often approximated by parameterized potentials (usually two-particle potentials, e.g. Lennard-Jones), which are often determined empirically for a specific situation.

More precision and generality can be achieved by quantum mechanical calculations for the electrons, but they are much more expensive. They usually involve Density Functional Theory (DFT, see also chapter 2.1). It can be combined with diagonalization of a reduced electronic basis set at each step of the Molecular Dynamics calculation. Alternatively, in the so-called Car-Parinello method, the evolution of the electronic degrees of freedom is instead calculated more cheaply by using the Kohn-Sham energy density from DFT.

"Molecular Dynamics" is also applicable to classical dynamics on macroscopic scales, with appropriate forces and time scales, up to the simulation of the motion of whole galaxies, subject only to gravitation.

Often, one wants to determine results on a large spatial scale, which however depend on microscopic details. Then it is necessary to perform some sort of *coarse-graining* of the microscopic variables in order to treat them on a larger scale; and the new variables may in turn also have to be coarse-grained, etc., in order to get to a macroscopic scale.

MD is widely used for studying many-particle systems. It is a simulation of the system as it develops over a stretch of time. The dynamics coincide (more or less) with the actual evolution of the simulated physical system. This is a big advantage over Monte Carlo calculations, which are designed to sample from an equilibrium distribution and do not usually incorporate actual dynamics, since the evolution from one Monte Carlo configuration to the next is designed for efficient sampling, but often has nothing to do with the real time evolution of a system. Molecular dynamics calculations resemble actual experiments on a system, including preparation, time evolution, and measurements.

3.1.2 Equilibrium and Nonequilibrium

MD is most often used to extract *equilibrium* properties of a system. Since Newton's equations of motion are solved, the *energy is constant* during a simulation. The system follows trajectories of constant energy in phase space.[1] This is the *microcanonical ensemble*. After an initial equilibration, the average of measured quantities over a long time should sample the whole microcanonical ensemble (ergodic hypothesis).

Although the microcanonical ensemble is simulated, one usually defines an *effective temperature*, using the average kinetic energy per degree of freedom,

$$\left\langle \frac{1}{2} m_i \, v_{i\alpha}^2 \right\rangle \quad =: \quad \frac{1}{2} k_B T_{eff} \; .$$

In a system of N particles, one averages over the total number of degrees of freedom N_f, where $N_f \simeq 3N$ in 3 dimensions, to define an instantaneous temperature at time t:

$$T_{eff}(t) \quad := \quad \frac{1}{k_B N_f} \sum_{i=1}^{N} m_i \, \mathbf{v}_i^2 \; . \tag{3.3}$$

The relative fluctuations of this temperature will be of order $1/\sqrt{N_f}$. One can also sample the actual canonical ensemble of fixed temperature, as discussed later.

Molecular Dynamics can also be used to simulate *Nonequilibrium properties*, for example the time evolution starting at a specific initial state. One prominent example is the Weather Forecast (which also includes nonparticle equations of motion like fluid dynamics). We shall touch on some aspects of nonequilibrium systems in subsequent chapters.

3.1.3 Boundary conditions

For finite systems, *boundary conditions* are important. Most equilibrium MD simulations are performed for periodic boundary conditions (pbc), which means that the finite system is surrounded by identical systems with exactly the same configuration in phase-space. Forces act across the boundary of neighboring replicas. The angular momentum is not conserved when pbc are imposed. Another common situation are open boundary conditions (obc). The reason for pbc is that otherwise too many particles are at the surface of the system. For example, in a simple cubic crystal of 1000 atoms, almost 50% of the atoms are in the outer layer, and for 10^6 atoms there are still 6% on the surface.

[1] One needs to carefully distinguish these "system trajectories" in the total phase space of N particles from the individual "particle trajectories".

3.1.4 Time scales

The time integration for many particles together cannot be done exactly. It has to be performed approximately in many small time steps of some size Δt, such that the result becomes formally exact when the time step size goes to zero.

However, for any Δt, errors will accumulate and the calculation will always become too inaccurate after some time. As a matter of fact, the trajectory of each individual particle will deviate rather quickly from its true course. One would therefore want to make Δt very small. Note that with proper integration schemes, *ensemble averages* over many particles fortunately remain useful much longer than individual particle trajectories.

On the other hand, the maximum time t which can be reached with some computational effort, i.e. with some number n of iterations, is $t = n\Delta t$. In order to reach a physically interesting large time t, one wants a large value of Δt. If one wants to to time-averaging, then the maximum time needs to be much larger than physical correlation times. We note that there is a *tradeoff* between accuracy and maximum time. In any case, Δt needs to be small enough that the relevant errors (difficult to estimate) at the maximum time are still acceptable.

The actual time scales will very greatly from system to system, depending on whether one simulates, e.g., argon atoms or galaxies. For liquid argon, which can be described reliably by simple Lennard-Jones pair forces, the typical time step used in the numerical integration of the equations of motion is about 10^{-14} seconds, which means that with 10^6 time steps, a total simulation time of about 10^{-8} seconds can be covered. For galaxies, the time steps may be many years.

We will see that *during* a time step, the *force* acting on a particle is assumed to be *constant*. This can be taken to be a criterion for choosing a suitable length τ for a time step in the given physical situation. When some particles are very close together and forces change quickly, it can be advantageous to subdivide a time step just for these particles in order to keep a larger time step overall.

The number of particles which can be simulated will be limited. This corresponds to a finite system size which will also become noticeable after some time. One should expect to observe differences to the true evolution when the particles have travelled on more than half the linear system size. In practice, noticeable effects tend to show up considerably later.

The applicability of MD is limited by the times and system sizes which can be reached, and also by the physical approximations in the equations of motion (e.g. lack of quantum mechanics or approximate treatment).

3.2 MD at constant energy

If the forces acting on the particles depend on the their mutual relative position, then energy and total momentum are physically conserved. By design, particle number and volume are usually conserved as well. Time averages then correspond to the microcanonical or (NVE) ensemble. The rough structure of the MD algorithm is

- Initialize, then loop:
- Calculate forces
- Integrate equations of motion for a small time step Δt.
- For equilibrium calculations, let the system settle for some equilibration time,
- Then perform measurements.
- As a diagnostic tool, check that the energy does stay constant to some desired precision.

The following discussion will be worded mostly for an equilibrium simulation of a gas or a liquid of particles.

3.2.1 Initialization

The number N of particles and the finite size volume need to be specified. As an example, we will use periodic boundary conditions with a box of size L^3.

For equilibrium calculations, an effective "temperature" is usually of greater interest than the total energy and might therefore be specified as an input parameter. (Actual simulations in the canonical ensemble will be discussed later). The particles are assigned initial positions and velocities. Typically, the positions are chosen on a regular grid or at random, while the velocities $\mathbf{v}_i$ are generated according to the Boltzmann distribution

$$p(v_\alpha) \propto e^{-mv_\alpha^2/(2k_BT)}.$$

A vanishing total momentum is achieved by subtracting the mean momentum from all particle momenta.

3.2.2 Force calculation

This is usually the most time-consuming part of an MD-simulation.

As an example, we will consider the Lennard-Jones pair potential

$$u^{ij}(r) \;=\; 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] . \tag{3.4}$$

between particles i and j, with r the distance between them. All particles shall have the same mass m.

From now on, we express everything as dimensionless quantities in terms of *reduced units*, taking σ as the unit of length, ε as the unit of energy, and m as the unit of mass. (Then the unit of time is $\sigma\sqrt{m/\varepsilon}$ and the unit of temperature is ε/k_B.) Note that different original problems and scales can correspond to the same set of reduced parameters. In reduced units, the Lennard-Jones potential becomes

$$u^{ij}(r) \;=\; 4\left[\left(\frac{1}{r}\right)^{12} - \left(\frac{1}{r}\right)^{6}\right] . \tag{3.5}$$

Every particle interacts with every other particle. Within the box, there are therefore $O(N^2)$ interactions. With pbc, there are also the interactions with the infinite number of replicas („periodic images") of every particle.

Cut-off

For an interaction which decays quickly with distance like the Lennard-Jones potential, it is convenient to *cut off* the force[2] at some radius r_c. When r_c is small enough, $r_c \leq \frac{L}{2}$, then a particle i interacts with at most one copy of particle j (see figure 3.1)

The cut-off neglects interactions beyond r_c. Their total contribution to the energy can be estimated approximately as

$$u_{tail} = \frac{N\rho}{2}\int_{r_c}^{\infty} dr\; 4\pi r^2\,\left(u(r) - u(r_c)\right) . \tag{3.6}$$

Let us calculate the force on particle i. Assume that we use p.b.c. and cut off the forces at some radius $r_c \leq \frac{L}{2}$. We now loop over all other particles $j \neq i$ and calculate the smallest distance between i and j on the periodic lattice (e.g. in direction x: $r_x = \min(|x_i - x_j|,\; L - |x_i - x_j|)$. When

[2] A sharp cut-off of the *potential* would cause an infinite force at distance r_c, unless all energies are shifted to obtain zero potential at r_c first.

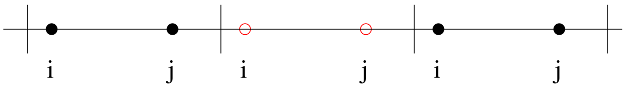


Figure 3.1: Periodic boundary conditions in 1d, with two particles i and j.

$r_c \leq \frac{L}{2}$, there will be at most one instance of each particle j (in the original box or in the periodic copies) with a distance smaller than the cut-off radius r_c. For these particles, we need to calculate the force on particle i. In x-direction it is

$$f_x(r) = -\frac{\partial u(r)}{\partial r_x} = -\frac{\partial r}{\partial r_x}\frac{\partial u(r)}{\partial r} = -\frac{r_x}{r}\frac{\partial u(r)}{\partial r}\,, \tag{3.7}$$

which for our Lennard Jones system becomes

$$f_x(r) = \frac{48 r_x}{r^2}\left(\frac{1}{r^{12}} - \frac{1}{2r^6}\right)\,. \tag{3.8}$$

Improved scaling for finite range interactions

We need to repeat the force calculation for all N particles, i.e. $N(N-1)/2$ pairs. The computational effort in a direct approach therefore scales like N^2, whereas the time integration step will only scale like $O(N)$. When there are many particles the force calculation would be a prohibitive bottleneck. Fortunately, we can get much better scaling, even like $O(N)$, also for force calculations.

In case of finite range interactions (or finite cut-off), the key is clever bookkeeping. A simple and very good approach are so-called *cell lists.*Let r_c be the maximum range of the force (e.g., the cut-off). We divide our box of size L^3, $L \gg r_c$, into cells of size r_c^3 or slightly larger. Initially, we go through all particles and we make lists of the particles contained in every cell, one list for each cell. This is an $O(N)$ operation. After each time step (or when a particle moves), the lists are updated. Each particle i in a cell interacts only with particles j in the same cell or in immediately neighboring cells. The number of these particles is not of order N: when the system becomes larger at finite density, then the number of cells grows, but the number of neighbors to consider remains the same on average. If M is the average number of particles per cell, the force calculation will now scale like $O(NM)$ instead of $O(N^2)$.

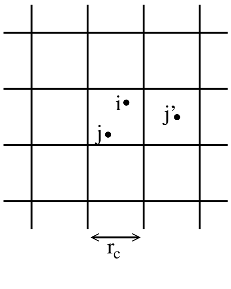


Figure 3.2: Cells: with cutoff r_c, particle i interacts only with particles in the same cell or in directly neighbouring cells.

Improved scaling for infinite range interactions

Coulomb and gravitational interactions are important cases. They are long range, and often cannot be cut off without large undesirable effects. Then the force calculations would scale like N^2.

Fortunately, there are again methods to reduce the scaling, to about $O(N \log N)$. The two main strategies are so-called *Tree-algorithms* and *Particle-Mesh* methods. The description below will be worded for gravitational forces. The goal is to calculate, for each particle i, the total force acting on this particle

$$\vec{f}_i = \sum_{j \neq i} \vec{g}(\vec{r}_i - \vec{r}_j)\, m(\vec{r}_j)\,, \tag{3.9}$$

where $\vec{g}(\vec{r})$ contains the $1/r^2$ dependence.

In **Particle-Mesh methods**, space is divided into a finite number of grid (=mesh) points. Each particle is assigned to one grid point, or its mass (or charge) is distributed over several neighbouring grid points. Note that (3.9) is a convolution. On a grid, it can be evaluated in Fourier space in $O(N \log N)$ time by using the Fast Fourier Transform and the convolution theorem.[3] When particles can be very close together, this method would need a very finely spaced grid, thus extremely many grid points for the whole system. It can be improved considerably, by separately calculating the forces from particles which are in some close neighborhood of particle i (the "particle-particle" part of the method), and using the grid and FFT only for particles with are further away, so that for them the grid approximation is acceptable. The distinction between "near" and "far" need not be abrupt, since one can always write

$$f(r) \;=\; f(r)h(r) + f(r)(1-h(r)) \;\equiv\; f_{near}(r) + f_{far}(r))$$

with any function $h(r)$ that separates "near" from "far". Without FFT, this becomes the (slower) old method of "Ewald-sums". The desired $O(N \log N)$ scaling is achieved with FFT. Corresponding methods are called "particle-particle particle-mesh" (= PPPM = P3M), or particle-mesh Ewald (PME) method, etc. They are often applied to calculating Coulomb forces in crystals. Then periodic boundary conditions can be helpful to reduce finite size effects. Their use requires that the system is charge-neutral, because otherwise the forces from the infinitely many periodic images would diverge. In cases without periodic boundary conditions, one needs to use zero-padding in the Fourier Transforms in order to avoid artifacts.

[3] Alternatively, instead of using (3.9), one can first calculate the potential $\Phi(\vec{r})$ and then obtain the force by differentiation in space (by finite differences) or by multiplikation with $\vec{k}$ in Fourier-space.

In **Tree methods**, space is divided into a *hierarchical grid*, organized like a tree, as illustrated for two space dimensions in figure 3.3. The tree root represents the whole system. It has four children (eight in 3 dimensions), representing a subdivision of space into four squares. If a subdivision contains one or more particles, it is again subdivided, until at the end, each final node corresponds to zero or one particle. Therefore, in dense regions of space, the tree extends further. Each node stores the center of mass and the total mass of all the nodes below it.

We can now calculate the force on a particle i: In the *Barnes-Hut* tree method, the tree is traversed starting from the root. Each node represents a region with spatial size d, at some distance r from particle i. If d/r is smaller than some threshold, then this region is relatively far away and its effect on particle i is approximated by the center of mass of that region, stored in the node. If d/r is larger than the threshold, then more detail is necessary, thus the children of the node need to be visited, etc. Tree methods are advantageous when the distribution of particles is very inhomogeneous, for example in astrophysics. As for almost all methods, many variants exist.

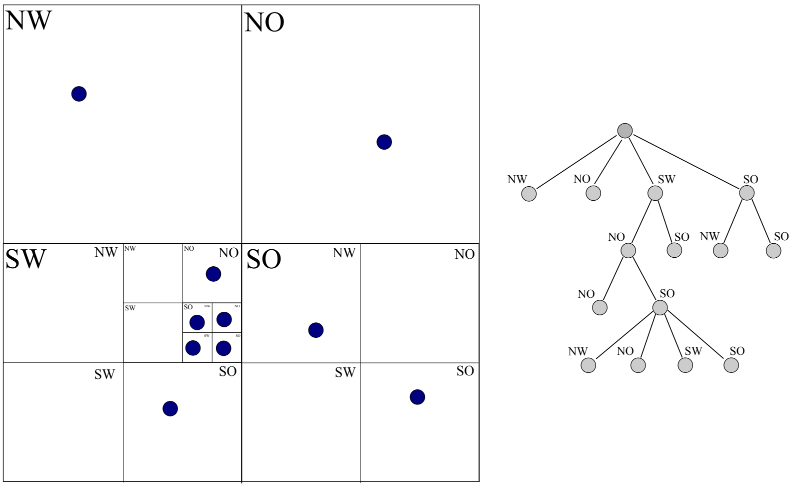


Figure 3.3: Barnes-Hut tree method, two-dimensional example. *Left:* space is subdivided into a hierarchical grid. *Right:* tree structure corresponding to the grid on the left. *(From beltoforion.de/article.php?a=barnes-hut-galaxiensimulation,* 10 May 2019.)

3.2.3 Time integration: Verlet and Leap-frog algorithm

We will integrate the equations of motion in discrete time steps of length τ. We will concentrate on one of the most widely used algorithms, which is both simple and reliable, the Verlet algorithm. We start with a Taylor expansion of the coordinates of a particle:

$$\begin{aligned} \mathbf{r}(\tau) &= \mathbf{r}(0) + \tau\,\dot{\mathbf{r}}(0) + \frac{\tau^2}{2}\ddot{\mathbf{r}}(0) + \frac{\tau^3}{6}\dddot{\mathbf{r}}(0) + \mathcal{O}(\tau^4) \\ \mathbf{r}(-\tau) &= \mathbf{r}(0) - \tau\,\dot{\mathbf{r}}(0) + \frac{\tau^2}{2}\ddot{\mathbf{r}}(0) - \frac{\tau^3}{6}\dddot{\mathbf{r}}(0) + \mathcal{O}(\tau^4) \\ \Rightarrow\quad \mathbf{r}(\tau) + \mathbf{r}(-\tau) &= 2\mathbf{r}(0) + \tau^2\,\mathbf{f}[\mathbf{r}(0), 0] + \mathcal{O}(\tau^4). \end{aligned}$$

The Verlet algorithm cuts away the $O(\tau^4)$ part. Thus

$$\mathbf{r}(\tau) = 2\mathbf{r}(0) - \mathbf{r}(-\tau) + \tau^2\,\mathbf{f}[\mathbf{r}(0), 0] \tag{3.10}$$

at each time step. At the very first step, one would need $\mathbf{r}(-\tau)$, which is not known. Instead one employs the initial condition with a simpler discretization

$$\mathbf{r}(\tau) = \mathbf{r}(0) + \tau\,\mathbf{v}(0) + \frac{\tau^2}{2}\,\mathbf{f}[\mathbf{r}(0), 0]. \tag{3.11}$$

for the first time step. The corresponding discretization error is $\mathcal{O}(\tau^3)$, which occurs just once.

VERLET ALGORITHM

$$\mathbf{r}(t+\tau) = 2\mathbf{r}(t) - \mathbf{r}(t-\tau) + \tau^2\,\mathbf{f}[\mathbf{r}(t), t], \qquad t = \tau, 2\tau, \ldots$$

where Equation (3.11) can be used to obtain $\mathbf{r}(\tau)$ initially. The error in a single time step is $O(\tau^4)$. However, one can see after some calculation that the error after n steps is $n(n+1)/2$ larger. (It becomes quadratic in n because Verlet contains a discretized second derivative of r.) Therefore the overall error for the position of a single particle in a fixed background after a time $T = n\tau$ is $O(\tau^2)$. We shall see below that the overall errors for the interacting many-body system actually tend to grow much faster.

The Verlet method does not use velocities, except at the first step. In order to calculate velocities, e.g. for obtaining the kinetic energy, one can use

$$\mathbf{r}(t+\tau) - r(t-\tau) \;=\; 2\tau\,\mathbf{v}(t) \;+\; O(\tau^3)\,. \tag{3.12}$$

There is an alternative formulation of the Verlet algorithm which is more robust against rounding errors. From

$$\frac{\mathbf{v}(\tau/2) - \mathbf{v}(-\tau/2)}{\tau} = \dot{\mathbf{v}}(0) + \mathcal{O}(\tau^2)$$

we obtain

$$\mathbf{v}(\tau/2) = \mathbf{v}(-\tau/2) + \tau \; \mathbf{f}[\mathbf{r}(0), 0] + \mathcal{O}(\tau^3).$$

We use, likewise, for the positions

$$\frac{\mathbf{r}(\tau/2 + \tau/2) - \mathbf{r}(\tau/2 - \tau/2)}{\tau} = \dot{\mathbf{r}}(\tau/2) + \mathcal{O}(\tau^2)$$

which leads to

$$\mathbf{r}(\tau) = \mathbf{r}(0) + \tau \; \mathbf{v}(\tau/2) + \mathcal{O}(\tau^3).$$

This approach is called the leap-frog[4] algorithm due to the way r and v interchange in time. Space-coordinates are computed at $0, \tau, 2\tau, \ldots$ and velocities are calculated at intermediate times $\tau/2, 3\tau/2, \ldots$.

LEAP-FROG ALGORITHM

$$\begin{aligned} \mathbf{r}(t+\tau) &= \mathbf{r}(t) + \tau \cdot \mathbf{v}(t+\tau/2) && t = 0, \tau, \ldots \\ \mathbf{v}(t+\tau/2) &= \mathbf{v}(t-\tau/2) + \tau \cdot \mathbf{f}[\mathbf{r}(t), t] && t = \tau, 2\tau, \ldots \\ \mathbf{v}(\tau/2) &= \mathbf{v}(0) + \frac{\tau}{2} \cdot \mathbf{f}[\mathbf{r}(0), 0]. && t = 0 \end{aligned}$$

Let us consider one more modification. We use

$$\begin{aligned} \mathbf{r}(t+\tau) &= \mathbf{r}(t) + \tau \cdot \mathbf{v}(t) + \frac{\tau^2}{2} \ddot{\mathbf{r}}(t) + \mathcal{O}(\tau^3) \\ &= \mathbf{r}(t) + \tau \cdot \mathbf{v}(t) + \frac{\tau^2}{2} \mathbf{f}[\mathbf{r}(t), t] + \mathcal{O}(\tau^3) \\ \mathbf{v}(t+\tau) &= \mathbf{v}(t) + \tau \ddot{\mathbf{r}}(t+\tau/2) + \mathcal{O}(\tau^3). \end{aligned}$$

This approach would require the knowledge of $\ddot{\mathbf{r}}(t+\tau/2)$, which can be approximated by

$$\frac{\ddot{\mathbf{r}}(t+\tau) + \ddot{\mathbf{r}}(t)}{2} = \ddot{\mathbf{r}}(t+\tau/2) + \mathcal{O}(\tau^2) \; .$$

We obtain the

[4] German: Bockspringen

VELOCITY-VERLET ALGORITHM

$$\begin{aligned}\mathbf{r}(t+\tau) &= \mathbf{r}(t) + \tau\cdot\mathbf{v}(t) + \frac{\tau^2}{2}\mathbf{f}[\mathbf{r}(t),t] \\ \mathbf{v}(t+\tau) &= \mathbf{v}(t) + \tau\frac{1}{2}\left\{\mathbf{f}[\mathbf{r}(t),t] + \mathbf{f}[\mathbf{r}(t+\tau),t+\tau]\right\}.\end{aligned}$$

This form is equivalent to the other two schemes as long as all computational steps are performed with infinite accuracy, but it is less susceptible to numerical errors.

3.2.4 Stability

Several properties are important for a good time integration scheme. They are all satisfied by the Verlet-algorithm and its variants.

First of all, energy needs to be conserved, both short-term and long-term. This is a very difficult requirement for MD methods. In fact, there are a number of schemes (like so-called „predictor-corrector" schemes) which use higher order expansions in τ and thus seem to allow bigger time steps τ. However, their long-term performance is worse than that of the Verlet algorithm !

One criterion that such methods often do not satisfy is the *time reversal symmetry* of Newton's equations of motion.

Another one is that of *preservation of volume in phase space*. All trajectories of a system that correspond to a particular energy E are contained in a specific volume in phase space. If we let Newton's equations of motion act on all points in this volume, we end up with exactly the same volume. A method which does not conserve phase space volume may eventually spread over all of phase space, and then cannot conserve energy.

Symplectic integrators perform time steps which correspond to a canonical transformation in phase space. They do conserve phase space volume. One can construct such symplectic time evolution methods with a fairly recent approach that has been adapted from Quantum Monte Carlo, namely the use of time evolution operators looking like $\exp(\tau H)$, together with subdivisions of the Hamilton function (e.g. into kinetic and potential energy) and approximations like the Baker-Hausdorff equation ("Trotter-Suzuki-approximation"). The Verlet algorithm can be written in such a way. One can show that symplectic integrators follow the exact time evolution of a slightly different (but very complicated) Hamilton function, which differs from the desired Hamiltonian only by terms of order τ or smaller.

There is, however, a another big stability concern. The trajectories of a system goverened by Newton's equations of motion depend sensitively on the initial conditions, even with exact time evolution.

When $\vec{x}(0)$ is perturbed by a small amount $\delta \cdot \vec{e}$, where $\vec{e}$ is a unit vector, then $x(t)$ will typically change by $\exp(t\lambda(\vec{e}))$. There are as many exponents $\lambda(\vec{e})$ as there are directions in phase space. For a Hamiltonian system, one exponent will always be zero (corresponding to $\vec{e} \parallel \vec{v}(0)$), and the others will occur in pairs $\pm\lambda$ (corresponding to conservation of phase space volume). The largest positive λ is usually called *the* Lyapunov exponent.

Thus two system trajectories which are initially very close will diverge from each other as time progresses ! This is the so-called *Lyapunov-instability*. Therefore even tiny integration errors and numerical errors in MD will lead the simulated trajectory to diverge from the true one.

For the calculation of, e.g., trajectories of satellites in space this can be devastating. Fortunately, one is usually interested in averages, over particles and over time, which behave much better.

3.2.5 Measurements

The equilibrium thermodynamic (ensemble) average of any quantity is obtained via averaging over time. For an observable O the expectation value reads

$$\langle O \rangle = \lim_{T\to\infty} \frac{1}{T} \int_0^T dt\, O(t).$$

For instance, the inner energy $U = \langle E \rangle$. We have already mentioned the kinetic energy, which defines the so-called instanteneous temperature. One can also measure the pressure. For pairwise additive interactions one can write

$$P \;=\; \rho k_B T \;+\; \frac{1}{dV} \left\langle \sum_{i<j} \mathbf{r}_{ij}\, \mathbf{f}(\mathbf{r}_{ij}) \right\rangle \;, \tag{3.13}$$

where d is the dimensionality of the system and V is the volume. This is actually the relation for a canonical ensemble, but is often employed also at constant E.

Other important quantities are correlation functions like the so-called radial distribution function $g(r)$, which is proportional to the density-density correlation function. Care has to be taken to evaluate these correlations efficiently, e.g. by FFT, in order to avoid $O(N^2)$ scaling.

3.3 Example: Harmonic Oscillator

Here we consider the harmonic oscillator since it allows to *asses the stability of the algorithm analytically*. The equation of motion for the 1D harmonic oscillator is

$$\ddot{x} = -\omega^2\, x.$$

For $t > 0$ the Verlet algorithm reads

$$x(t+\tau) - 2x(t) + x(t-\tau) = -\tau^2\omega^2\, x(t), \tag{3.14}$$

For $t = 0$ we need to specify the initial conditions. We have $\dot{v} = f$ and, therefore, $v(\tau) \simeq v(0) + \tau f[x(0)]$. We take the initial condition $v(0) = 0$ and the initial Verlet time step then is

$$x(\tau) = x(0) - \frac{\tau^2}{2}\omega^2 x(0). \tag{3.15}$$

Eqs. (3.14) can be expressed in matrix form if we use the assignment $x_n = x(n\,\tau)$ for $n \in \mathbb{N}_0$:

$$\underbrace{\begin{pmatrix} -2 & 2 & 0 & & & \\ & 1 & -2 & 1 & & \\ & & 1 & -2 & 1 & \\ & & & 1 & -2 & \ddots \\ & & & & \ddots & \ddots \end{pmatrix}}_{=:\boldsymbol{M}} \mathbf{x} = -\tau^2\omega^2\mathbf{x}.$$

(Zero matrix elements are supressed.) Thus, the eigenvalue equation $\boldsymbol{M}\mathbf{x} = \lambda\mathbf{x}$ has to be solved for the given eigenvalue $\lambda = -\tau^2\omega^2$. We will now solve this equation analytically. Since $x(t) = e^{i\omega t}$ is the exact solution for the harmonic oscillator, we try the ansatz

$$x_n = e^{i\alpha\, n}. \tag{3.16}$$

The general condition (3.14) for $\tau > 0$ reads $e^{i\alpha(n+1)} - 2e^{i\alpha n} + e^{i\alpha(n-1)} = -(\tau\omega)^2\, e^{i\alpha n}$ or rather

$$e^{i\alpha} - 2 + e^{-i\alpha} = -(\tau\omega)^2.$$

Hence, α is given by

$$1 - (\tau\omega)^2/2 = \cos(\alpha). \tag{3.17}$$

This implicit equation for α has real solutions only for

$$-1 \leq (\tau\omega)^2/2 - 1 \leq 1$$

or rather

$$0 \leq \tau\omega \leq 2.$$

In other words, the discretization has to obey *at least* $\tau \leq \frac{2}{\omega}$, otherwise α becomes complex and the solution (3.16) obtains exponentially increasing components, as we will see later. However, the time evolution becomes imprecise already much earlier (see below). Equation (3.17) always has two roots, namely $\pm\alpha$. Hence, the general solution obeying the initial condition reads

$$x(n\tau) = Ae^{i\alpha n} + Be^{-i\alpha n} = a\cos(\alpha n + \varphi),$$

with parameters which are fixed by the initial condition $x(t = 0) = x(0) = a\cos(\varphi)$. We still have to satisfy $v(0) = 0$, i.e. eq. (3.15).

$$\begin{aligned} a\cos(\alpha+\varphi) &\overset{!}{=} a\cos(\varphi) - \frac{(\tau\omega)^2}{2} a\cos(\varphi) \\ &= a\cos(\varphi)\left[1 - \frac{(\tau\omega)^2}{2}\right]. \end{aligned}$$

Because of Eq. (3.17), the last line is also equal to $a\cos(\varphi)\cos(\alpha)$. Thus

$$\begin{aligned} \cos(\alpha+\varphi) &= \cos(\varphi)\cos(\alpha) \\ \cos(\alpha)\cos(\varphi) - \sin(\alpha)\sin(\varphi) &= \cos(\varphi)\cos(\alpha) \\ \sin(\varphi) &= 0. \end{aligned}$$

The solution therefore reads

$$x_n = x_0\cos(\alpha n). \tag{3.18}$$

(In the original continuum representation this would read $x(t) = x(0)\cos(\frac{\alpha}{\tau}t)$, which would correspond to the exact solution if $\frac{\alpha}{\tau} = \omega$.) For $\tau\omega \ll 1$, Eq. (3.17) yields

$$\begin{aligned} 1 - (\tau\omega)^2/2 &\simeq 1 - \frac{\alpha^2}{2} \; + \; O(\alpha^4) \\ \alpha &\simeq \tau\omega. \end{aligned}$$

Thus the numerical solution indeed approaches the correct solution for $\tau \to 0$, with an error of $O(\alpha^4) = O((\tau\omega)^4)$. Obviously, there is a tradeoff between the accuracy and the total time that can be simulated by a fixed number of iterations.

Figures 3.4 and 3.5 show the time dependence of $x(t)$ for different parameters $\tau\omega$. We see that the simulation is stable over many periods, if $\tau\omega \ll 1$, while it goes off course at larger τ. As anticipated, the trajectory diverges for $\tau\omega > 2$.

We now estimate the number of periods until the discretization error becomes significant. The result of the Verlet algorithm oscillates with frequency α/τ. According to Eq. (3.18) the position at time $t = n\tau$ is given by $x^v(t) = x_0\cos(\alpha n)$. A characteristic time t^* at which the result has become completely wrong is when $x^v(t^*) = -x(t^*),$ i.e. when the phase error is π:

$$\begin{aligned} |\omega t^* - \alpha t^*/\tau| &= \pi \\ t^* &= \frac{\pi}{|\omega - \alpha/\tau|}. \end{aligned}$$

The number N^* of periods $T = 2\pi/\omega$ corresponding to t^* is:

$$N^* = \frac{t^*}{T} = \frac{\omega\tau}{2\,|\omega\tau - \alpha|}.$$

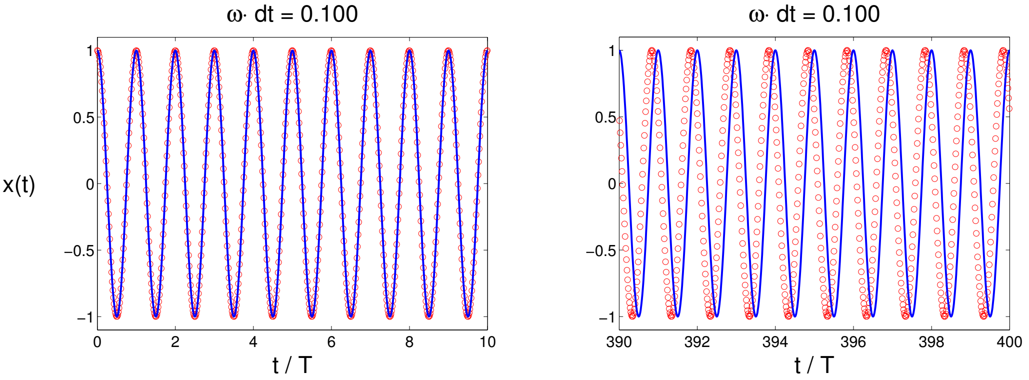


Figure 3.4: Simulation of the harmonic oscillator with $\omega\tau = 0.1$. The exact solution is represented by the blue line, and the MD results by red circles. According to Eq. (3.19) it takes $N^* = 1200$ cycles for the Verlet algorithm to go completely off course.

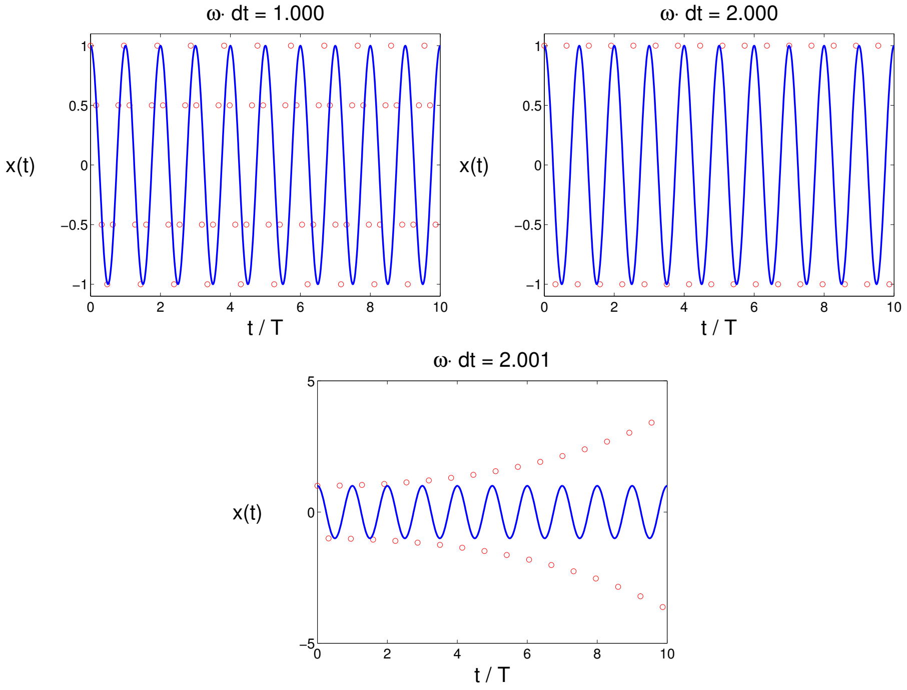


Figure 3.5: Top left: Simulation of the harmonic oscillator with the foolish choice $\omega\tau = 1$. The result of formula (3.19), $N^* = 12$, is corroborated by the simulation. Top right and bottom: Simulations for $\omega\tau = 2$ and $\omega\tau = 2.001$. The phase becomes complex for $\omega\tau > 2$ leading to an exponential increase of the amplitude. ("dt" in the figure is τ in the main text).

Together with an expansion of Eq. (3.17) to order α^4, with $\cos\alpha \simeq 1-\frac{\alpha^2}{2}+\frac{\alpha^4}{24}$ this leads after a few lines to

$$|\omega\tau - \alpha| \simeq \frac{(\omega\tau)^3}{24},$$

and hence

$$N^* = \simeq \frac{12}{(\omega\tau)^2}. \tag{3.19}$$

Alternatively, if a certain number N^* of stable periods is required, the condition for the discretization reads

$$\omega\tau \ll 2\sqrt{\frac{3}{N^*}}\,,$$

or

$$\frac{T}{\tau} \gg \pi\sqrt{\frac{N^*}{3}},$$

which gives the number of partitions of one period.
For example, $N^* = 1000$ corresponds to $T/\tau = 57$ partitions.

3.4 Generalizations

Canonical ensemble

The "effective temperature" can be adjusted in an MD simulation by rescaling all velocities during equilibration. However, this is still the microcanonical ensemble of constant energy

In order to obtain a true canonical ensemble, one can employ the "Anderson thermostat": The system is coupled to a heat bath. The coupling works through occasional impulsive forces that act on randomly selected particles. It can be considered to be a Monte Carlo move which transports the system from one energy to another. The collisions with the heat bath are designed such that all constant energy shells are visited according to their Boltzmann weight.

Hybrid Monte Carlo

Molecular dynamics, even a not-so-good scheme, can be employed to greatly improve the performance of some *Monte Carlo* calculations. In this method, the Molecular Dynamics evolution is used to provide a new *proposal configuration* for the next Monte Carlo step. In order to provide the MC configuration with dynamics, artificial momenta are introduced, and chosen randomly according to some distribution. The only requirement on the MD simulation is time reversal symmetry. By way of a fairly long time evolution, the system can move far in phase space, while keeping its total energy (actual energy plus small artificial kinetic energy) approximately constant, so that the new configuration will likely be accepted in the Monte Carlo step. Changes of energy are achieved by changing the articificial momenta. This method has for example been in very successful use for simulations of QCD (Quantum Chromodynamics, the theory of quarks and gluons).

Nonequilibrium simulations

Molecular dynamics is also able to to look at the time evolution of a system starting from a special initial situation and/or with external non-conservative driving forces. Then there is no time averaging, so that great care has to be taken to integrate the equations of motion as precisely as possible.

Agent based simulations

The general approach of a simulation of invidual ″particles″, interacting and following specific rules in their time evolution has been carried over into many other fields, where one usually speaks of ″agent based simulations″, including e.g. financial markets, evolution, or the flow of dense crowds of people at large events.

Beyond Molecular Dynamics

When Molecular Dynamics is not feasible, e.g. because the necessary integration times cannot be reached with sufficiently small time steps, then one has to resort to coarser pictures. They can be based on the more traditional formulation of solid bodies or of fluids as continuous media obeying differential equations like e.g. the Navier-Stokes equation. For solid matter, these are often treated by the *Finite Element Method (FEM)*. There, space is partitioned into finite regions (the „elements″), for example triangles on surfaces. The differential equation to be solved is rephrased as a variational problem. The solutions are approximated by a linear combination of a *finite* set of functions. The coefficients of this linear combination are then the solution of a (linear or nonlinear) minimization problem, for which techniques like conjugate gradient can be used.

For liquids the difficulties are greater. Depending on the precise problem, *Computational Fluid Dynamics (CFD)* uses a large variety of discretizations of space and of integration methods, often with phenomenological additional interactions, for example to describe turbulence. MD, on the other hand, can describe turbulence directly, on a microscopic scale, although currently only with unrealistic parameters like extremely large forces, in order to make it fit within the space and time available.

Chapter 4

Finite Element Method

Literature: (tiny selection from an enormous number of publications)

- K.J. Bathe, *Finite Element procedures*, 2nd edition, Pearson 2014 (1043 pages, comprehensive).
  Available as pdf at www.mit.edu.

- Aurélien Larcher, Niyazi Cem Degirmenci, *Lecture Notes: The Finite Element Method*, 2013.
  Concise and readable mathematical treatment, 40pp. Pdf available as course literature at www.kth.se/social/course/SF2561.

- Multiphysics cyclopedia, *The Finite Element Method (FEM)*.
  https://www.comsol.de/multiphysics/finite-element-method.
  The present chapter including figures is mostly based on this exposition.

The Finite Element Method (FEM) is an efficient approach to discretizing and numerically solving partial differential equations (PDEs) for some quantity $u(\mathbf{x}, t)$ (static or time dependent), which may describe quantities like the temperature distribution of a complex body, or mechanical shifts, electric currents, fluid flow, etc. . The method and its many variations have been developed since about the 1960s, with first beginnings much earlier. It is very widely applied especially in engineering, and many software-packages are available. The present chapter provides a brief introduction.

Discretization and basis functions

A basic ingredient of the Finite Element Method is the approximation of the desired solution $u(\mathbf{x}, t)$ of the PDEs as a linear combination of a carefully chosen finite set of *basis functions* $\psi_i(\mathbf{x})$ (also called *shape functions*).

$$u(\mathbf{x}, t) \;\approx\; u_h(\mathbf{x}, t) \;:=\; \sum_i u_i(\mathbf{x}, t)\; \psi_i(\mathbf{x})\;. \tag{4.1}$$

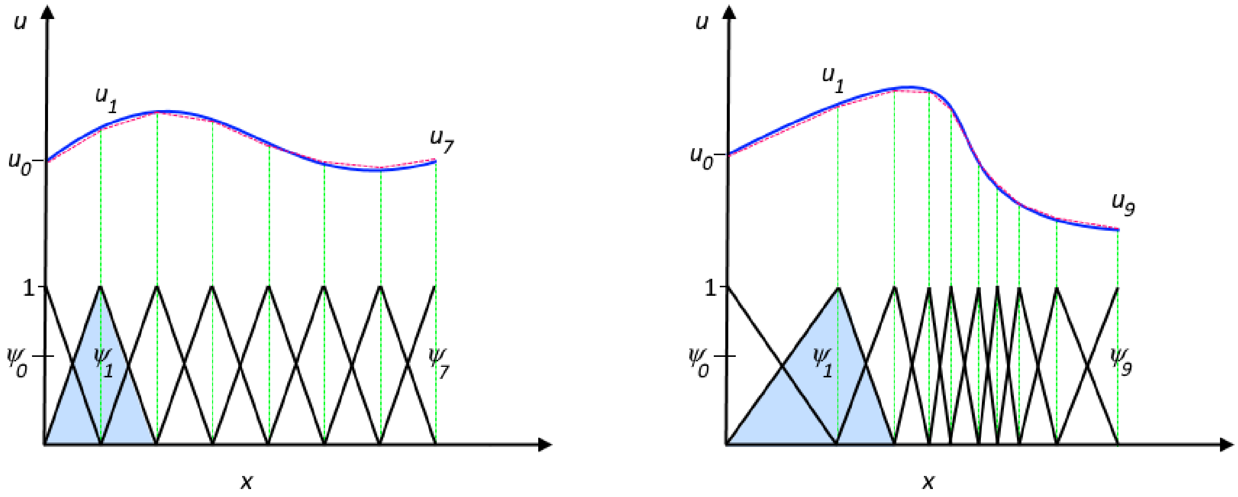


Figure 4.1: Example for a set of local linear basis functions in one spatial dimension, and the corresponding approximation (thin red lines) of a function $u(x)$ (blue line) as a linear combination of these basis functions. *Left:* equally spaced nodes x_i. *Right:* adaptive discretization.

An example is provided in Fig. 4.1. The finite number of coefficients makes the problem *variational*. FEM is related to the Ritz variational approach.

In order to obtain suitable basis functions, space is divided into small ”elements”, for example line segments in 1d, triangles in 2d, or tetraedra in 3d. The corner points of these elements are called ”nodes”. The basis functions are usually chosen to have support only on a single or a few spatial elements, i.e. to be very **local in space**, and each even to be zero at the position of all nodes except one.

Examples with linear basis functions on a triangular grid are provided in Fig. 4.2.

Differential equation: example

As an example we will look at the temperature $T(\mathbf{x}, t)$ in a solid. The relevant equation is

$$\rho\, C_p \,\frac{\partial T}{\partial t} \;+\; \nabla \mathbf{q} \;=\; g(T, \mathbf{x}, t)\;, \tag{4.2}$$

where ρ is the density, C_p the heat capacity, $\mathbf{q}(\mathbf{x}, t)$ is the heat flux, and $g(T, t, \mathbf{x})$ is a heat source, which might vary with temperature and time.

The heat flux itself depends on temperature as $\mathbf{q} = -k\nabla T$, where k is the thermal conductivity, so that the differential equation for $T(\mathbf{x}, t)$ reads

$$\rho\, C_p \,\frac{\partial T}{\partial t} \;+\; \nabla(-k\nabla T) \;=\; g(T, \mathbf{x}, t)\;. \tag{4.3}$$

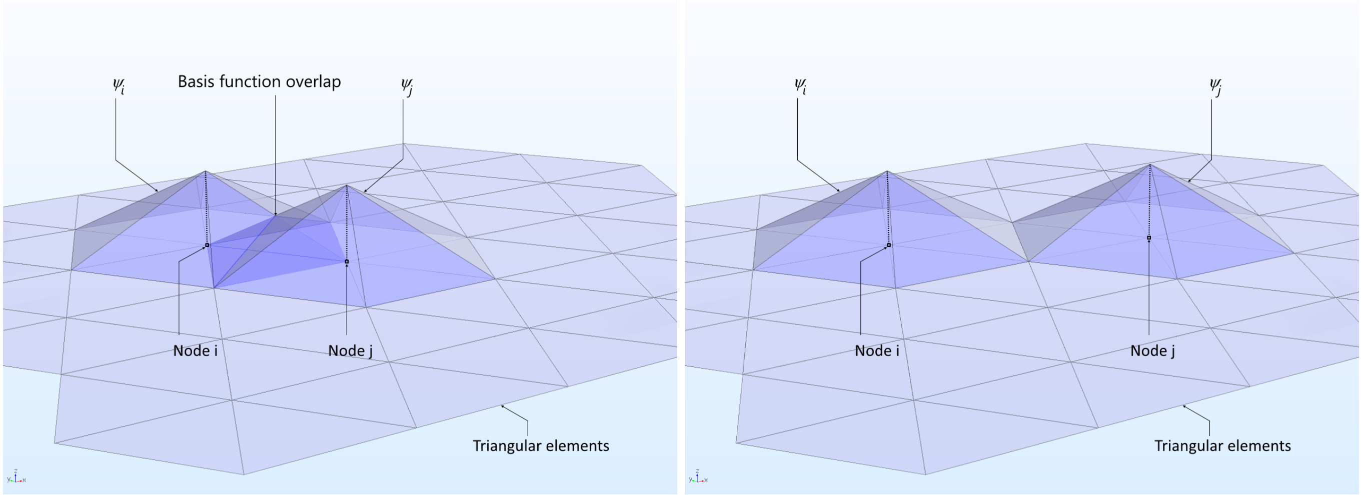


Figure 4.2: Example for a set of local linear basis functions on a 2 dimensional triangular grid. There is one basis function for each node. *Left:* Basis functions for nearest neighbor nodes have a non-zero overlap. *Right:* Basis functions for nodes at farther distances do not overlap.

The temperature is further determined by the *initial condition* $T(\mathbf{x}, t_0)$ and by *boundary conditions*.

We will now specifically look at a *steady state*, i.e. a *time-independent* situation. Then $\frac{\partial T}{\partial t} = 0$ and (4.3) becomes

$$\nabla(-k\nabla T) \;=\; g(T, \mathbf{x}) \quad \text{in } \Omega, \tag{4.4}$$

where Ω is the spatial domain of the body which we want to describe.

Typical boundary conditions are:
(1) the temperature on some boundary $\partial\Omega_1$:

$$T(\mathbf{x}) = T_0(\mathbf{x}) \;\text{ on } \partial\Omega_1, \tag{4.5}$$

(2) the heat flux normal to some surface $\partial\Omega_2$ might be determined by an ambient temperature T_{amb}, or it might be zero on some surface $\partial\Omega_3$:

$$(-k\nabla T)\cdot\mathbf{n} = h(T - T_{amb}) \;\text{ on } \partial\Omega_2, \tag{4.6}$$

$$(-k\nabla T)\cdot\mathbf{n} = 0 \;\text{ on } \partial\Omega_3. \tag{4.7}$$

Here $\mathbf{n}$ is a unit vector normal to the respective boundary.

Integral "weak form" of the differential equations

The original physical problem is in continuous space and the differential equations have to be satisfied at every point $\mathbf{x}$. When the solution is approximated by basis functions on discrete finite elements, derivatives become problematic. For example, the first derivative of $u(x)$ in Fig. 4.1 is discontinuous, so that the second derivative is defined only in the sense of a distribution. Such problems can be avoided by multiplying the PDE by a *test-function* $\varphi(\mathbf{x})$ and *integrating* over space. Eq. (4.4) becomes

$$\int_\Omega \nabla \cdot (-k\nabla T)\, \varphi \, dV \;=\; \int_\Omega g(T)\, \varphi \, dV \tag{4.8}$$

This relation has to hold *for any test function* from a suitable Hilbert space. The solution $T(\mathbf{x})$ is also assumed to belong to such a Hilbert space. This is the so-called "weak formulation" of the PDEs, which needs to hold only in an integral sense instead of a every point $\mathbf{x}$. When the solution is differentiable enough, then together with the boundary conditions, eq (4.9) can be shown to be equivalent to the original PDE (4.4) (see e.g. Larcher et al.).

Galerkin approach

In the Galerkin approach, the test functions are taken from the same Hilbert space as the solution function. Eq (4.8) can be integrated by parts (Green's first identity) to obtain

$$\int_\Omega (k\nabla T) \;\cdot\; \nabla\varphi \, dV \;+\; \int_{\partial\Omega} (-k\nabla T) \;\cdot\; \mathbf{n}\, \varphi \, dS \;=\; \int_\Omega g(T)\, \varphi \, dV \;. \tag{4.9}$$

We now approximate the solution as a linear combination of a finite number of basis functions ψ_i like in eq. (4.1:

$$T(\mathbf{x}) \;\approx\; T_h(\mathbf{x}) \;:=\; \sum_i T_i(\mathbf{x})\, \psi_i(\mathbf{x}) \;. \tag{4.10}$$

These basis functions span our Hilbert space. Eq. (4.9 has to hold for every test function, which in the Galerkin approach are taken from the same Hilbert space. Thus it is sufficient if eq. (4.9 holds for every basis function ψ_j as a test function. We then obtain the discretized form of eq. (4.9:

$$\sum_i T_i \int_\Omega (k\nabla\psi_i)\cdot\nabla\psi_j \, dV \;+\; \sum_i \int_{\partial\Omega} (-kT_i\nabla\psi_i)\cdot\mathbf{n}\,\psi_j \, dS \;=\; \int_\Omega g(\sum_i T_i\psi_i)\, \psi_j \, dV \;. \tag{4.11}$$

(Note that on the right hand side, $\sum_i T_i\psi_i$ is not a factor, but the argument T of $g(T)$. If there are N basis functions ψ_j, then eq. 4.11 is a **system of N equations for N unknowns** T_i, which is now suitable for numerical treatment.

Often, the "source term" $g(T)$ is a linear function of T. Then eq. 4.11 is linear in T_i and can be written as a set of linear equations (!):

$$\boxed{\mathbf{A}\,\mathbf{T}_h \;=\; \mathbf{b}\,,} \tag{4.12}$$

for which many numerical approaches are available, for example Gauss elimination, Jacobi, Gauss-Seidel, or Conjugate Gradient methods. Here,

$\mathbf{T}_h$ is the vector of unknown coefficients T_i. The matrix $\mathbf{A}$ is called *system matrix* or *stiffness matrix* (referring to early applications in structural mechanics). In FEM, it is usually not symmetric.

When the source function is nonlinear, then one can write a similar equation, but with the vector b now a nonlinear function of the unknown coefficients T_i, and more expensive non-linear numerical solvers like e.g. the Newton method need to be employed.

It is very beneficial to use basis functions ψ_i that are very local in space, like in figure 4.2. Then the integrals in eq. 4.11 are non-zero only for (e.g.) basis functions on nearest-neighbor nodes (!). Therefore the matrix $\mathbf{A}$ becomes **sparse** (namely almost diagonal) and thus much easier to solve.

Time dependence

In time-dependent cases, the time derivative in (4.3) needs to be included. In discretized form, we then get, similar to (4.11),

$$\begin{aligned} \rho\, C_p \sum_i \frac{\partial T_i}{\partial t} \int_\Omega \psi_i \psi_j \, dV \;+\; \sum_i T_i \int_\Omega (k\nabla\psi_i)\cdot\nabla\psi_j \, dV \qquad (4.13) \\ +\; \sum_i \int_{\partial\Omega} (-kT_i\nabla\psi_i)\cdot\mathbf{n}\,\psi_j \, dS \;=\; \int_\Omega g(\sum_i T\psi_i)\,\psi_j\, dV\;, \end{aligned}$$

where now the coefficients T_i are time-dependent, while the basis functions in this equation are still taken to be time-independent.

One could generalize FEM to 3+1 dimensional elements, which is however computationally very demanding. Instead, time is treated separately, with two different approaches, both involving finite time steps Δt. One can approximate the time derivative as

$$\frac{\partial T_{i,t}}{\partial t} \;\approx\; \frac{T_{i,t+\Delta t} \;-\; T_{i,t}}{\Delta t}. \qquad (4.14)$$

Let us assume that we already know the coefficients $T_{i,t}$ at time t and want to obtain $T_{i,t+\Delta t}$.

The first approach is similar to Molecular Dynamics. Equation (4.13) is written for the coefficients $T_{i,t}$ at time t. Then $T_{i,t+\Delta t}$ only occurs in the time derivative and can be directly computed. This is called an *explicit* time integration scheme and requires very small time steps Δt for stability and accuracy.

In the second approach, Equation (4.13) is instead written for the coefficients $T_{i,t+\Delta t}$ at time $t + \Delta t$. Now the known coefficients $T_{i,t}$ only appear in the time derivative and we get a linear system of equations for $T_{i,t+\Delta t}$, which needs to be solved at every time step. This *"implicit method"* allows for larger time steps, but each step is more expensive.

In practice, FEM routines switch between these approaches automatically, and also employ higher order polynomial expressions for the time derivative instead of the simple difference equation (4.13).

Error estimates

One way to estimate the reliability of an FEM solver for a specific problem is the *"method of manufactured solutions"*, in which a related problem with exactly known solution is treated by the solver. The output can then be compared to the known exact solution.
Example: suppose we have a solver for the Poisson equation

$$\nabla^2 u(\mathbf{x}) = g(\mathbf{x}) \ \text{ in } \Omega \tag{4.15}$$

with boundary conditions

$$u(\mathbf{x}) = 0 \ \text{ on } \partial\Omega. \tag{4.16}$$

We now take a freely chosen function $\bar{v}(\mathbf{x})$ (for example the output of the solver for our actual problem) for which $\bar{v}(\mathbf{x}) = 0$ on $\partial\Omega$, take $g(\mathbf{x}) = \nabla^2\bar{v}(\mathbf{x})$, and compute an approximate solution $\bar{v}_h(\mathbf{x})$ to (4.15) with our solver. We know that the exact solution to this problem is $\bar{v}(\mathbf{x})$ and we can now examine the difference to $\bar{v}_h(\mathbf{x})$. Note that when we take $\bar{v}(\mathbf{x})$ to be a previous output from our solver and and use the same discretization and basis functions for the error estimate, then this method does not provide information about any features lost by the discretization.

Formally exact error bounds for specific quantities can sometimes be obtained a posteriori (see Larcher et al.). However, they involve bounds on matrix elements of the inverse of the adjoint of the matrix $\mathbf{A}$, which are very difficult to obtain reliably in a numerical calculations, so that an actual bound may become too large to be useful.

Other error estimates may be obtained from the (potentially misleading) convergence of results when using smaller and smaller elements.

Basis function refinements and mesh adaptations

The basis functions need not be linear functions on their elements like in Figs. 4.1 and 4.2. In figure 4.3, first and second order Lagrange elements are shown, on square spatial elements. Higher order basis functions have the advantage of better approximating the true solution $u(\mathbf{x})$ on a given discrete mesh. On the other hand, there are more higher order basis functions, so that the computational effort increases. Alternatively, one can therefore use simple basis functions on a finer mesh.

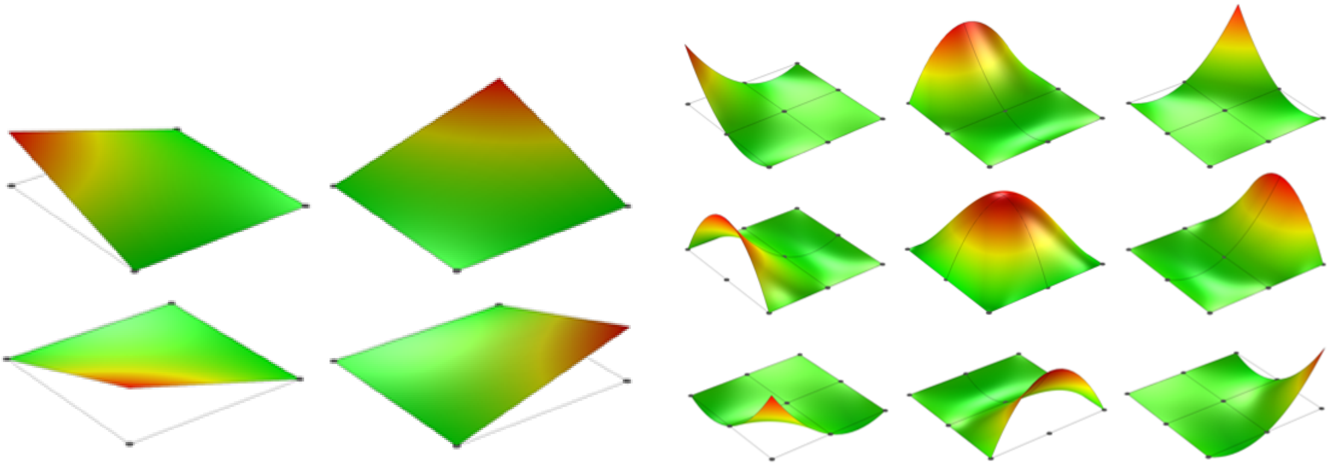


Figure 4.3: The shape functions (basis functions) for a first-order (*left*) and second order (*right*) square Lagrange element. The corresponding 1d functions would be Lagrange polynomials.

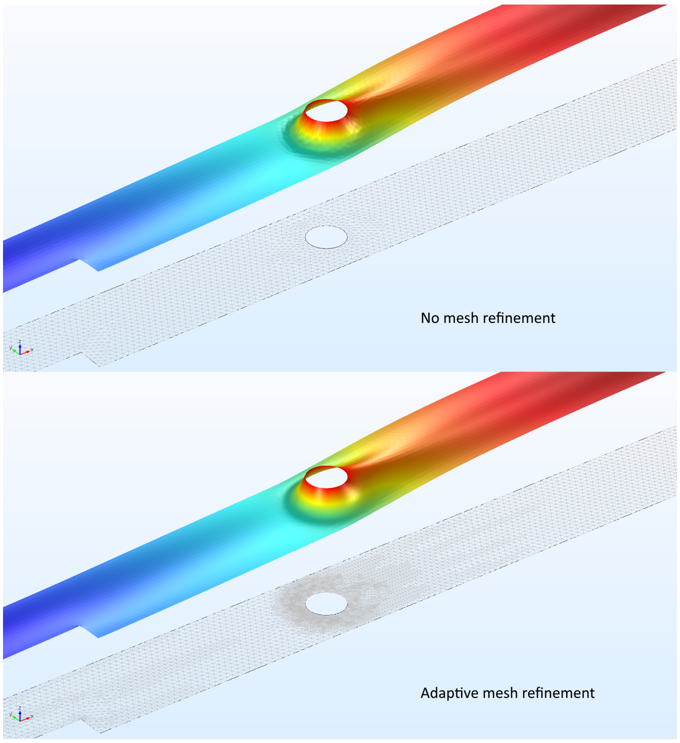


Figure 4.4: Temperature around a heated cylinder subject to a flow, computed without (top) and with (bottom) mesh refinement.

The spatial discretization elements can and should of course be adapted to the problem. One can for example use spatial elements with curved boundaries to better represent a given body. The adaptions can also be done iteratively according to a first approximate solution, e.g. by using a finer grid where more precision is needed, for example because of large gradients. An example is shown in figure 4.4.

Appendix A

Errors (Uncertainties)

A.1 Error specification

Stating a numerical result without providing an error estimate is meaningless.

Unfortunately, missing or completely erroneous error specifications[1] are a remarkably common and important problem, even and especially in everyday life. One should note that an implicit error specification is already and often unintentionally passed along via the number of digits specified in a result.

Some examples:

A statement like ″the national debt of Austria next year will increase by 13,427,853,974.48 Euros″, the kind of which is sometimes found in newspapers, would imply that this sum is known to a precision better than one Euro. Here it is obvious that this cannot be true. But what is the actual precision ? Is it on the one million Euro level ? Or more like one billion or even more ? In the latter (probably more realistic) case, only the first or the first two digits of that long number have any meaning.

″The distance between Vienna and Graz is 145.5827345 km″. The last digit corresponds to a tenth of a millimeter. Can this be true ? Distance between which points ? The following year, the distance is quoted 2.3 centimeters larger. Are Vienna and Graz splitting apart ?

Such numbers with many digits often result from blindly reading off and quoting a number from some display or from a computer screen. They can lead to a lot of confusion.

Obviously, these numbers need some realistic error estimate in order to be useful. An efficient way to *specify* errors on numbers with numerous digits is an expression like ″145.5827(12)″, where the digits in brackets are

[1] A better name would be ″uncertainty″ of a result, but ″error″ has become standard.

the error of the same number of preceeding digits, meaning here that there is an error of ″12″ on the digits ″27″. One still needs to add information on whether this a one-standard-deviation error or whether it corresponds for example to two standard deviations (both common).

A.2 Polls and histograms: simple error estimates

Another unfortunate everyday example is in the specification of the results of polls, e.g. for political parties. One week it may be that party A is reported at ″32.18%″ and next week at ″31.45%" and a big discussion begins on why the numbers have dropped. But is the difference significant ? What is the error of those 4-digit specifications ? Results similar to polls commonly occur in measurements or in numerical calculations, in the form of histograms which are to approximate some probability distribution.

Let us try to obtain an easy to use simple error estimate for such numbers. A more precise discussion can be found in the following section on Bayesian analysis.

We examine a histogram with a total number $N = \sum_i N_i$ of entries, where N_i is the number of entries in bin number i. We first want to obtain an error estimate for N_i. With one specific value for i, we can treat this as a Bernoulli problem (binomial distribution): how many events belong to bin number i and how many do not ? The variance for N_i for the binomial distribution is

$$\sigma^2_{N_i} \;=\; N\,p_i\,(1-p_i)\;, \tag{A.1}$$

where p_i is the probability for bin i, of the underlying probability distribution from which the events in the histogram were drawn. We can make a simple estimate

$$p_i \;\approx\; \frac{N_i}{N} \tag{A.2}$$

from the data, with normalization $\sum_i p_i = 1$. The Bayesian analysis in the next section will show us that this estimate is good when $N_i \gg 1$ and not so good for small values of N_i. With this estimate, we get

$$\boxed{\sigma_{N_i} \;\approx\; \sqrt{N_i\,(1-\frac{N_i}{N})}} \tag{A.3}$$

for a single standard deviation. As long as N_i does not come close to N, this can even be approximated in the *very* simple form

$$\underline{\sigma_{N_i} \;\approx\; \sqrt{N_i}} \tag{A.4}$$

which can be a take-home message for rough estimates. (Even when $\frac{N_i}{N}$ is as large as $\frac{1}{2}$, this simple form only introduces an error of a factor of $\sqrt{2}$).

Translated to p_i, this becomes

$$\sigma_{p_i} \;\approx\; \frac{1}{N}\sigma_{N_i} \;\approx\; \frac{1}{\sqrt{N}}\,\sqrt{p_i\,(1-p_i)} \;\approx\; \frac{1}{\sqrt{N}}\,\sqrt{\frac{N_i}{N}\,(1-\frac{N_i}{N})}\,, \qquad \text{(A.5)}$$

in which we see the very important $\frac{1}{\sqrt{N}}$ dependence, related to the central limit theorem.[2]

Let us look at examples: a typical poll has between 400 and 1000 participants. If the "32.18%" mentioned above came from a poll with about N=1000 participants, then what is the approximate error ? Eq. (A.3) gives $\sigma_{N_i} \approx \sqrt{322(1-0.322)} \simeq 15$, not too far from the simple estimate $\sqrt{322} \simeq 18$. Thus the one-sigma (!) error in p_i is about $\frac{15}{1000} = 1.5\%$! Therefore the difference to the second poll result of "31.45%" is not at all significant !

We learn that the results of the polls *really* need to be specified together with an error estimate.

A.3 Histograms: Bayesian analysis

Let us now try to do a better analysis of histogram data, using the Bayesian approach. We shall see that the results for large N_i are very similar, but for small N_i we will obtain better and more sensible results both for p_i and for the error estimate.[3]

The simple analysis in the preceding section used the approximation $p_i \approx \frac{N_i}{N}$. This is a *frequentist* approach. When $N_i = 0$, then the estimate $p_i = 0$ results, as well as the clearly wrong estimate of $\sigma_{p_i} = 0$: there is no data in bin i, yet the error-estimate has the value zero ?

We look again at the situation where we have a set of N numbers, classified into n_b bins i of width b_i, plus an additional bin "r" for any remainder of the data, outside of the range of the histogram, i.e., we actually classify the data into $n_b + 1$ classes. Bin number i contains N_i numbers.

The numbers are assumed to be drawn independently from an underlying probability distribution with probabilities p_i for bin i and p_r for the remainder. We want to obtain an estimate for p_i from the numbers N_i, with normalization $\sum_{i=1}^{n_b} + p_r = 1$, as well as an error for this estimate.

Let us already look at the result of the Bayesian analysis, shown in Fig. A.1 for the case of an example. Here the bin widths $b_i = b$ are chosen constant for simplicity. The black line corresponds to the underlying real distribution p_i, usually not known, for which we want to obtain an estimate. In the example, this distribution has two peaks and is zero in between, *except* for two bins in the middle, where p_i is small but nonzero. The bar heights h_i are normalized with respect to integration over x, $h_i = \frac{p_i}{b_i}$.

[2] For histogram bars of width b_i we get $h_i = p_i/b_i$ and $\sigma_{h_i} = \sigma_{p_i}/b_i$, like in (A.17).

[3] Thanks go to Florian Maislinger for initiating this section and for contributing most of it, including the calculations and figures.

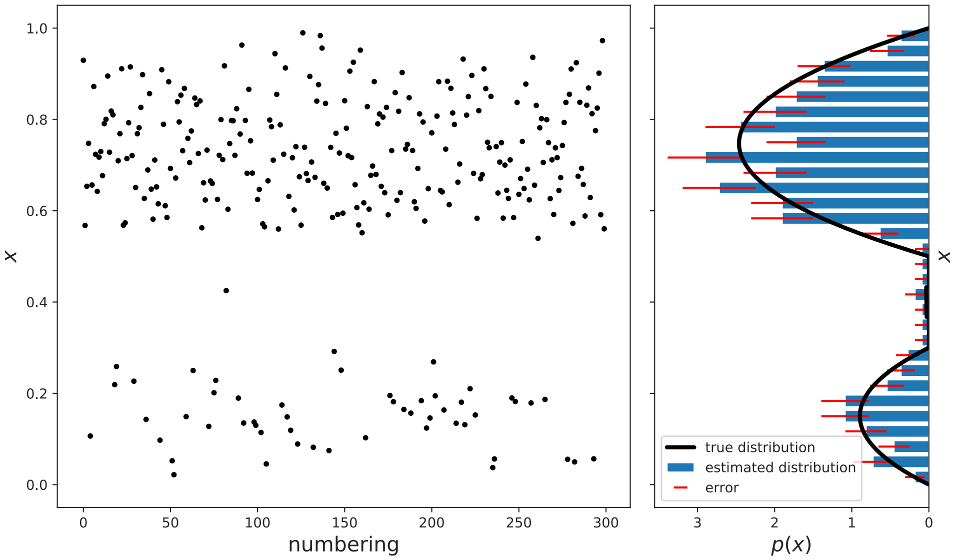


Figure A.1: Example data (left) and resulting bar plot of a normalized histogram h_i after the Bayesian analysis (right).

Fig. A.2 shows more details, with the same Bayesian results on the right hand side, in comparison to the result of the simple frequentist analysis on the left hand side. The two plots have different bar heights. The total number of data $N = 300$ here is fairly small, such that the differences become clear. The insets provide detail. One of the bins in the middle has $N_i = 0$, which results in the strange frequentist estimate of $p_i = 0$, with zero error. For the same bin, the Bayesian analysis gives a finite estimate for p_i and a finite error.

Let us start the analysis. For a given bin i, we can again regard the distribution as binomial:

$$p\left(N_i \mid p_i, N, \mathcal{B}\right) = \binom{N}{N_i} \, p_i^{N_i} \, \left(1 - p_i\right)^{N - N_i} \quad . \tag{A.6}$$

We are interested in the distribution for p_i so we need to invert the distribution above with Bayes theorem.

$$p\left(p_i \mid N_i, N, \mathcal{B}\right) = \frac{p\left(N_i \mid p_i, N, \mathcal{B}\right) \; p\left(p_i \mid N, \mathcal{B}\right)}{p\left(N_i \mid N, \mathcal{B}\right)} \tag{A.7}$$

We now need a prior, which expresses any previous knowledge. We will use

$$p\left(p_i \mid N, \mathcal{B}\right) \propto \left(1 - p_i\right)^{n_b - 1} \quad . \tag{A.8}$$

This is the marginalized distribution of the *flat* prior:

$$p\left(p_1, p_2, \ldots, p_{n_b}, p_r \mid N, \mathcal{B}\right) \propto \delta\left(1 - \textstyle\sum_{m=1}^{n_b} p_m - p_r\right) \quad , \tag{A.9}$$

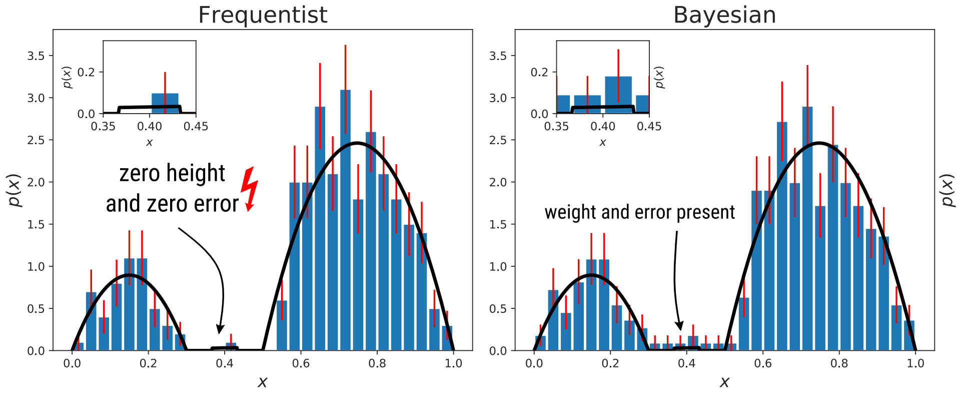


Figure A.2: Motivation for the Bayesian inference. If only a few numbers ($N = 300$) are generated, the basic frequentist method ($h_i = \frac{1}{b_i}\frac{N_i}{N}$) indicates that there is no density and zero error. Inset: Zoomed in around the center area where the probability distribution function has a small weight.

i.e. we assume that a priori each bin (including the ”rest” r) has equal probability. With (A.8), (A.7) becomes

$$p\left(p_i \mid N_i, N, \mathcal{B}\right) = \frac{1}{Z}\, p_i^{N_i}\ \left(1 - p_i\right)^{N - N_i + n_b - 1} \tag{A.10}$$

The normalization constant can be calculated using the integral

$$B(p, q)\ \ :=\ \ \int_0^1 x^{p-1}\ (1-x)^{q-1}\ \mathrm{d}x\ \ =\ \ \frac{\Gamma(p)\,\Gamma(q)}{\Gamma(p+q)}\ , \tag{A.11}$$

which is the so-called Beta-function. Here $\Gamma(n)$ is the Gamma-function. For integer arguments, $\Gamma(n+1) = n!$, and the integral simplifies to

$$\int_0^1 x^a\ (1-x)^b\ \mathrm{d}x\ \ =\ \ \frac{a!\ b!}{(a+b+1)!} \tag{A.12}$$

(which can also be obtained directly by iterating partial integratins) and our normalization becomes

$$\begin{aligned} 1 &= \int p\left(p_i \mid N_i, N, \mathcal{B}\right)\ \mathrm{d}p_i \\ &= \frac{1}{Z}\int_0^1 p_i^{N_i}\ \left(1-p_i\right)^{N-N_i+n_b-1}\ \mathrm{d}p_i \\ &= \frac{1}{Z}\,\frac{N_i!\ \left(N - N_i + n_b - 1\right)!}{\left(N+n_b\right)!} \\ \Longrightarrow\ \ p\left(p_i \mid N_i, N, \mathcal{B}\right) &= \frac{\left(N+n_b\right)!}{N_i!\ \left(N-N_i+n_b-1\right)!}\ p_i^{N_i}\ \left(1-p_i\right)^{N-N_i+n_b-1} \end{aligned} \tag{A.13}$$

Now we have everything to compute the first and second moment of the probability distribution and with this the expectation value and the standard deviation.

$$\begin{aligned}\langle p_i \rangle &= \int_0^1 p_i \, p\left(p_i \mid N_i, N, \mathcal{B}\right) \, \mathrm{d}p_i \\ &= \frac{(N+n_b)!}{N_i! \; (N-N_i+n_b-1)!} \, \frac{(N_i+1)! \; (N-N_i+n_b-1)!}{(N+n_b+1)!} \; ,\end{aligned}$$

thus

$$\boxed{\langle p_i \rangle \quad = \quad \frac{N_i+1}{N+n_b+1}} \tag{A.14}$$

and

$$\begin{aligned}\left\langle p_i^2 \right\rangle &= \int_0^1 p_i^2 \, p\left(p_i \mid N_i, N, \mathcal{B}\right) \, \mathrm{d}p_i \\ &= \frac{(N+n_b)!}{N_i! \; (N-N_i+n_b-1)!} \, \frac{(N_i+2)! \; (N-N_i+n_b-1)!}{(N+n_b+2)!} \\ &= \frac{N_i+2}{N+n_b+2} \, \frac{N_i+1}{N+n_b+1} \\ &= \frac{N_i+2}{N+n_b+2} \, \langle p_i \rangle\end{aligned}$$

The standard deviation $\sigma_{p_i} \equiv \sqrt{\left\langle p_i^2 \right\rangle - \langle p_i \rangle^2}$ can be simplified using (A.14) and becomes

$$\boxed{\sigma_{p_i} \quad \equiv \quad \sqrt{\left\langle p_i^2 \right\rangle - \langle p_i \rangle^2} \quad = \quad \sqrt{\frac{1}{N+n_b+2} \, \langle p_i \rangle \left(1 - \langle p_i \rangle\right)}} \tag{A.15}$$

We can convert the p_i to bar heights h_i, normalized such that the integral over x is unity: the probability to generate a number in bar i is the integral over the bar height h_i, thus

$$\langle p_i \rangle = \int_{x_i}^{x_i+b_i} h_i \, \mathrm{d}x = b_i \, h_i \tag{A.16}$$

With this we can compute the bar height and error for a normalized histogram:

$$h_i = \frac{1}{b_i} \, \langle p_i \rangle \; , \tag{A.17}$$

$$\sigma_{h_i} = \frac{1}{b_i} \, \sigma_{p_i} \; . \tag{A.18}$$

A.3.1 Limit of large numbers

Let us compare: the Bayesian result for $\langle p_i \rangle$ is $\frac{N_i+1}{N+n_b+1}$. For small numbers N_i and N, it can be quite different from the simple frequentist result $\frac{N_i}{N}$, but for large N_i and N it becomes similar.

Error estimates are quite different when N_i is small, as we already saw in the figures. We can rewrite $\sigma^2_{p_i}$ by plugging (A.14) into (A.15) :

$$\sigma^2_{p_i} = \frac{1}{N+n_b+2} \; \frac{N_i+1}{N+n_b+1} \left(1 - \frac{N_i+1}{N+n_b+1}\right) \; \approx \; \frac{1}{N} \, \frac{N_i}{N} \left(1 - \frac{N_i}{N}\right) \; . \tag{A.19}$$

The right hand side is the frequentist result, and the approximation becomes valid for large numbers, $N \gg n_b$ and $N_i \gg 1$. Fig. A.3 shows that indeed the results at large N and N_i are similar.

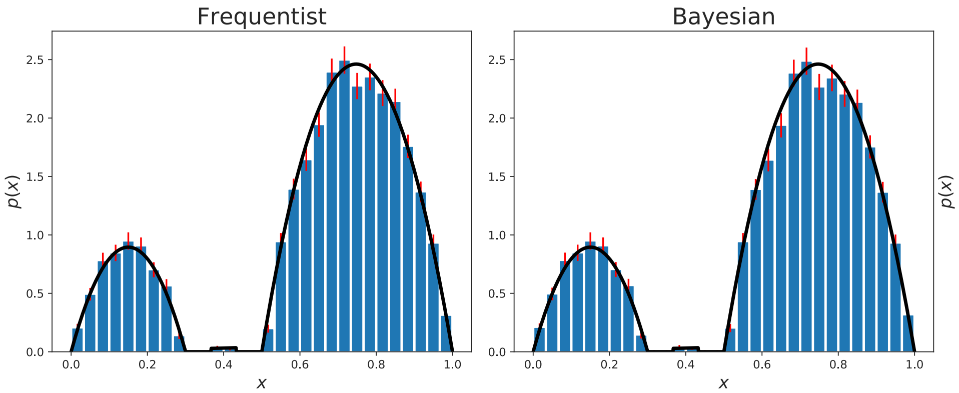


Figure A.3: If sufficiently many numbers (here $N = 5000$) are generated, both methods converge to the same result and yield an acceptable solution.

A.3.2 Conjugate prior

The Bayesian analysis for a binomial distribution which we have just performed can actually be written in a more general form. Eq. A.8 is a special case of the *conjugate prior* of the binomial distribution

$$p_{\alpha,\beta}(q) = \frac{q^{\alpha-1}(1-q)^{\beta-1}}{B(\alpha,\beta)} \; , \tag{A.20}$$

which is the so-called beta-distribution, with parameters α, β.
Bayes' theorem can be written as

$$p(p_i|N_i, N, \mathcal{B}) = \frac{p(N_i|p_i, N, \mathcal{B}) \; p(p_i|N, \mathcal{B})}{\int_0^1 dq \; p(N_i|q, N, \mathcal{B}) p(q|N, \mathcal{B})} . \tag{A.21}$$

When we now have N_i data in bin i (i.e. N_i successes and $N - N_i$ failures in the binomial process), and use the conjugate prior $p_{\alpha,\beta}(p_i)$, we get the posterior distribution

$$p(p_i|N_i, N, \mathcal{B}) = \frac{\binom{N}{N_i} p_i^{N_i}(1-p_i)^{N-N_i} \;\; p_i^{\alpha-1}(1-p_i)^{\beta-1} \;/\; B(\alpha,\beta)}{\int_0^1 dq \binom{N}{N_i} q^{N_i}(1-q)^{N-N_i} \;\; q^{\alpha-1}(1-q)^{\beta-1} \;/\; B(\alpha,\beta)} \tag{A.22}$$

$$= \frac{p_i^{N_i+\alpha-1}(1-p_i)^{N-N_i+\beta-1}}{B(N_i+1, N-N_i+1)} \tag{A.23}$$

$$= p_{\alpha+N_i,\beta+N-N_i}(p_i) \tag{A.24}$$

(For $\alpha = 1, \beta = 0$ the result is indeed the same as the previous result in (A.13).)

We see that we obtain *another beta-distribution*, now with parameters $\alpha + N_i$ and $\beta + N - N_i$, i.e., α increased by the number of successes and β by the number of failures of the binomial events. In this sense, the parameters α and β of the prior distribution can be interpreted as counts of previously performed measurements, thus making the term ″prior knowledge″ very explicit here, and allowing the new posterior distribution to be used as the prior distribution to take account of additional events.

A.4 Jackknife: General purpose error analysis with automatic error propagation

Data are often combined in complicated nonlinear ways to obtain results, including fits, and fits of fit-results, etc. It is then next to impossible to apply standard error propagation. One example are quantities expressed like $\langle A\rangle/\langle B\rangle$, where A and B correspond to measured quantities.

Another example is the autocorrelation function of a Markov Chain Monte Carlo calculation, for which it is very difficult to compute error bars. When the slope of the autocorrelation function is fitted, reliable errors on this slope are even more difficult to obtain, in part because the values of the autocorrelation function at different time distances are correlated among each other.

A surprisingly easy solution in many cases is provided by "resampling" techniques, in which partial samples of all data are taken several times and analyzed. We will concentrate on the Jackknife method.[4] A closely related but less useful procedure is called *bootstrap*.

To explain Jackknife we will use the analysis of a Monte Carlo time series as an example. We start by cutting the data (time series) into $N_B \gtrsim 30$ blocks. The number 30 is chosen so that the relative error of the calculated error will be reasonably small, namely of order $1/\sqrt{30}$. The blocks should ideally be uncorrelated. For Monte Carlo this means that they should be much larger than the autocorrelation times (which have to be determined separately). One can also combine Jackknife and Binning, i.e. start with small blocks and successively increase the block size until the error converges.

Error calculation

Analyze the data $N_B + 1$ times: We first perform an analysis of the full data set, arriving at a number which we call $R^{(0)}$ (This number can e.g. be the autocorrelation function at some time distance t, or it can e.g. be a fitted slope of the autocorrelation function).

Then we perform *another N_B analyses, each time leaving out one of the N_B blocks*, providing results $R^{(j)}$, $j = 1, \ldots, N_B$, with an average $R^{(av)}$. The fluctuation of these results tells us about the statistical error. However, the $R^{(j)}$ are highly correlated amongst each other, since they use almost the same data. The correct variance of the overall results is given by

$$\boxed{\sigma_R^2 = \frac{N_B - 1}{N_B} \sum_{j=1}^{N_B} \left(R^{(j)} - R^{(av)} \right)^2 \ .} \tag{A.25}$$

[4] Named for its versatility.

which is $(N_B - 1)^2$ times the usual estimator of the variance of a mean.

Bias

Usually , the result $R^(0)$ is corrected for a "bias", with the overall result

$$R \;=\; R^{(0)} \;-\; \text{Bias}, \quad \text{Bias} \;=\; (N_B - 1)(R^{av} - R^{(0)})\,. \tag{A.26}$$

However, this bias is usually much smaller than the statistical error of R and therefore unimportant. It can be used as a *diagnostic tool*: When "Bias" is of the same size as the statistical error or larger, then there is likely a *problem with the data* like an outlier, that is a strongly deviating data point.

Stability

A very big *advantage of the Jackknife approach* ist that all the analyses involve almost all the data. Therefore procedures like fits are very stable, providing a set of $N_B + 1$ stable results (e.g. for the slope of the autocorrelation function) and a reliable statistical error.

In contrast, in the more naive approach of analyzing small single bins one by one to compute a variance, quantities like fit-results may become meaningless.

We see that Jackknife is a very general and versatile error analysis with automatic error propagation, even through successive highly nonlinear data processing like fits.
In the related *bootstrap* method, random samples of N_B data points are taken many times and analyzed. This method cannot be used for time-series, since the sequential nature of events gets lost.