Computational Physics

Prof. Matthias Troyer

ETH Zürich, 2005/2006

Contents

Chapter 1

Introduction

1.1 General

For **physics students** the computational physics courses are recommended prerequisites for any computationally oriented semester thesis, proseminar, diploma thesis or doctoral thesis.

For **computational science and engineering (RW) students** the computational physics courses are part of the "Vertiefung" in theoretical physics.

1.1.1 Lecture Notes

All the lecture notes, source codes, applets and supplementary material can be found on our web page `http://www.itp.phys.ethz.ch/lectures/RGP/`.

1.1.2 Exercises

Programming Languages

Except when a specific programming language or tool is explicitly requested you are free to choose any programming language you like. Solutions will often be given either as C++ programs or Mathematica Notebooks.

If you do not have much programming experience we recommend to additionally attend the"Programmiertechniken" lecture on Wednesday.

Computer Access

The lecture rooms offer both Linux workstations, for which accounts can be requested with the computer support group of the physics department in the HPR building, as well as connections for your notebook computers. In addition you will need to sign up for accounts on the supercomputers.

1.1.3 Prerequisites

As a prerequisite for this course we expect knowledge of the following topics. Please contact us if you have any doubts or questions.

Computing

- Basic knowledge of UNIX
- At least one procedural programming language such as C, C++, Pascal, Modula or FORTRAN. C++ knowledge is preferred.
- Knowledge of a symbolic mathematics program such as Mathematica or Maple.
- Ability to produce graphical plots.

Numerical Analysis

- Numerical integration and differentiation
- Linear solvers and eigensolvers
- Root solvers and optimization
- Statistical analysis

Physics

- Classical mechanics
- Classical electrodynamics
- Classical statistical physics

1.1.4 References

1. J.M. Thijssen, *Computational Physics*, Cambridge University Press (1999) ISBN 0521575885

2. Nicholas J. Giordano, *Computational Physics*, Pearson Education (1996) ISBN 0133677230.

3. Harvey Gould and Jan Tobochnik, *An Introduction to Computer Simulation Methods*, 2nd edition, Addison Wesley (1996), ISBN 00201506041

4. Tao Pang, *An Introduction to Computational Physics*, Cambridge University Press (1997) ISBN 0521485924

5. D. Landau and K. Binder, *A Guide to Monte Carlo Simulations in Statistical Physics*, Cambridge University Press (2000), ISBN 0521653665

6. Wolfgang Kinzel und Georg Reents *Physik per Computer*, Spektrum Akademischer Verlag, ISBN 3827400201; english edition: *Physics by Computer*, Springer Verlag, ISBN 354062743X

7. Dietrich Stauffer and Ammon Aharony, *Introduction to percolation theory*, Taylor & Francis(1991) ISBN 0748402535

8. Various articles in the journal *Computers in Physics* and its successor journal *Computers in Science and Engineering*

1.2 Overview

1.2.1 What is computational physics?

Computational physics is a new way of doing physics research, next to experiment and theory. Traditionally, the experimentalist has performed measurements on real physical systems and the theoretical physicist has explained these measurements with his theories. While this approach to physics has been extremely successful, and we now know the basis equations governing most of nature, we face the problem that an exact solution of the basic theories is often possible only for very simplified models. Approximate analytical methods, employing e.g. mean-field approximations or perturbation theory extend the set of solvable problems, but the question of validity of these approximation remains – and many cases are known where these approximations even fail completely.

The development of fast digital computers over the past sixty years has provided us with the possibility to quantitatively solve many of these equations not only for simplified toy models, but for realistic applications. And, indeed, in fields such as fluid dynamics, electrical engineering or quantum chemistry, computer simulations have replaced not only traditional analytical calculations but also experiments. Modern aircraft (e.g. the new Boeing and Airbus planes) and ships (such as the Alinghi yachts) are designed on the computer and only very few scale models are built to test the calculations.

Besides these fields, which have moved from "physics" to "engineering", simulations are also of fundamental importance in basic physics research to:

- solve problems that cannot be solved analytically
- check the validity of approximations and effective theories
- quantitatively compare theories to experimental measurements
- visualize complex data sets
- control and perform experimental measurements

In this lecture we will focus on the first three applications, starting from simple classical one-body problems and finishing with quantum many body problems in the summer semester.

Already the first examples in the next chapter will show one big advantage of numerical simulations over analytical solutions. Adding friction or a third particle to the Kepler problem makes it unsolvable analytically, while a program written to solve the Kepler problem numerically can easily be extended to cover these cases and allows realistic modelling.

1.2.2 Topics

In this lecture we will focus on classical problems. Computational quantum mechanics will be taught in the summer semester.

- Physics:
  - Classical few-body problems
  - Classical many-body problems
  - Linear and non-linear wave equations
  - Other important partial differential equations
  - Monte Carlo integration
  - Percolation
  - Spin models
  - Phase transitions
  - Finite Size Scaling
  - Algorithms for N-body problems
  - Molecular Dynamics
- Computing:
  - Mathematica
  - Vector supercomputing
  - Shared memory parallel computing
  - Distributed memory parallel computing

1.3 Programming Languages

There have been many discussions and fights about the "perfect language" for numerical simulations. The simple answer is: it depends on the problem. Here we will give a short overview to help you choose the right tool.

1.3.1 Symbolic Algebra Programs

Mathematica and Maple have become very powerful tools and allow symbolic manipulation at a high level of abstraction. They are useful not only for exactly solvable problems but also provide powerful numerical tools for many simple programs. Choose Mathematica or Maple when you either want an exact solution or the problem is not too complex.

1.3.2 Interpreted Languages

Interpreted languages range from simple shell scripts and perl programs, most useful for data handling and simple data analysis to fully object-oriented programming languages such as Python. We will regularly use such tools in the exercises.

1.3.3 Compiled Procedural Languages

are substantially faster than the interpreted languages discussed above, but usually need to be programmed at a lower level of abstraction (e.g. manipulating numbers instead of matrices).

FORTRAN (FORmula TRANslator)

was the first scientific programming languages. The simplicity of FORTRAN 77 and earlier versions allows aggressive optimization and unsurpassed performance. The disadvantage is that complex data structures such as trees, lists or text strings, are hard to represent and manipulate in FORTRAN.

Newer versions of FORTRAN (FORTRAN 90/95, FORTRAN 2000) converge towards object oriented programming (discussed below) but at the cost of decreased performance. Unless you have to modify an existing FORTRAN program use one of the languages discussed below.

Other procedural languages: C, Pascal, Modula,...

simplify the programming of complex data structures but cannot be optimized as aggressively as FORTRAN 77. This can lead to performance drops by up to a factor of two! Of all the languages in this category C is the best choice today.

1.3.4 Object Oriented Languages

The class concept in object oriented languages allows programming at a higher level of abstraction. Not only do the programs get simpler and easier to read, they also become easier to debug. This is usually paid for by an "abstraction penalty", sometimes slowing programs down by more than a factor of ten if you are not careful.

Java

is very popular in web applications since a compiled Java program will run on any machine, though not at the optimal speed. Java is most useful in small graphics applets for simple physics problems.

C++

Two language features make C++ one of the best languages for scientific simulations: operator overloading and generic programming. Operator overloading allows to define mathematical operations such multiplication and addition not only for numbers but also for objects such as matrices, vectors or group elements. Generic programming, using template constructs in C++, allow to program at a high level of abstraction, without incurring the abstraction penalty of object oriented programming. We will often provide C++ programs as solutions for the exercises. If you are not familiar with the advanced features of C++ we recommend to attend the "Programmiertechniken" lecture on Wednesday.

1.3.5 Which programming language should I learn?

We recommend C++ for three reasons:

- object oriented programming allows to express codes at a high level of abstraction
- generic programming enables aggressive optimization, similar to FORTRAN
- C++-knowledge will help you find a job.

Chapter 2

The Classical Few-Body Problem

2.1 Solving Ordinary Differential Equations

2.1.1 The Euler method

The first set of problems we address are simple initial value problems of first order ordinary differential equations of the form

$$\frac{dy}{dt} = f(y,t) \tag{2.1}$$
$$y(t_0) = y_0 \tag{2.2}$$

where the initial value y_0 at the starting time t_0 as well as the time derivative $f(y,t)$ is given. This equations models, for example, simple physical problems such as radioactive decay

$$\frac{dN}{dt} = -\lambda N \tag{2.3}$$

where N is the number of particles and λ the decay constant, or the "coffee cooling problem"

$$\frac{dT}{dt} = -\gamma(T - T_{\text{room}}) \tag{2.4}$$

where T is the temperature of your cup of coffee, T_{room} the room temperature and γ the cooling rate.

For these simple problems an analytical solution can easily be found by rearranging the differential equation to

$$\frac{dT}{T - T_{\text{room}}} = -\gamma dt, \tag{2.5}$$

integrating both sides of this equation

$$\int_{T(0)}^{T(t)} \frac{dT}{T - T_{\text{room}}} = -\gamma \int_0^t dt, \tag{2.6}$$

evaluating the integral

$$\ln(T(t) - T_{\text{room}}) - \ln(T(0) - T_{\text{room}}) = -\gamma t \tag{2.7}$$

and solving this equation for $T(t)$

$$T(t) = T_{\text{room}} + (T(0) - T_{\text{room}}) \exp(-\gamma t). \tag{2.8}$$

While the two main steps, evaluating the integral (2.6) and solving the equation (2.7) could easily be done analytically in this simple case, this will not be the case in general.

Numerically, the value of y at a later time $t + \Delta t$ can easily be approximated by a Taylor expansion up to first order

$$y(t_0 + \Delta t) = y(t_0) + \Delta t \frac{dy}{dt} = y_0 + \Delta t f(y_0, t_0) + \mathrm{O}(\Delta\tau^2) \tag{2.9}$$

Iterating this equation and introducing the notation $t_n = t_0 + n\Delta t$ and $y_n = y(t_n)$ we obtain the Euler algorithm

$$y_{n+1} = y_n + \Delta t f(y_n, t_n) + \mathrm{O}(\Delta\tau^2) \tag{2.10}$$

In contrast to the analytical solution which was easy for the simple examples given above but can be impossible to perform on more complex differential equations, the Euler method retains its simplicity no matter which differential equation it is applied to.

2.1.2 Higher order methods

Order of integration methods

The truncation of the Taylor expansion after the first term in the Euler method introduces an error of order $\mathrm{O}(\Delta t^2)$ at each time step. In any simulation we need to control this error to ensure that the final result is not influenced by the finite time step. We have two options if we want to reduce the numerical error due to ths finite time step Δt. We can either choose a smaller value of Δt in the Euler method or choose a *higher order method*.

A method which introduces an error of order $\mathrm{O}(\Delta t^n)$ in a single time step is said to be *locally* of n-th order. Iterating a locally n-th order method over a fixed time interval T these truncation errors add up: we need to perform $T/\Delta t$ time steps and at each time step we pick up an error of order $\mathrm{O}(\Delta t^n)$. The total error over the time T is then:

$$\frac{T}{\Delta t}\mathrm{O}(\Delta t^n) = \mathrm{O}(\Delta t^{n-1}) \tag{2.11}$$

and the method is *globally* of $(n-1)$-th order.

The Euler method, which is of second order locally, is usually called a first order method since it is globally of first order.

Predictor-corrector methods

One straightforward idea to improve the Euler method is to evaluate the derivative dy/dt not only at the initial time t_n but also at the next time t_{n+1}:

$$y_{n+1} \approx y_n + \frac{\Delta t}{2} \left[f(y_n, t_n) + f(y_{n+1}, t_{n+1}) \right]. \tag{2.12}$$

This is however an *implicit* equation since y_{n+1} appears on both sides of the equation. Instead of solving this equation numerically, we first *predict* a rough estimate $\tilde{y}_{n+1}$ using the Euler method:

$$\tilde{y}_{n+1} = y_n + \Delta t f(y_n, t_n) \tag{2.13}$$

and then use this estimate to *correct* this Euler estimate in a second step by using $\tilde{y}_{n+1}$ instead of y_{n+1} in equation (2.12):

$$y_{n+1} \approx y_n + \frac{\Delta t}{2}\left[f(y_n, t_n) + f(\tilde{y}_{n+1}, t_{n+1})\right]. \tag{2.14}$$

This correction step can be repeated by using the new estimate for y_{n+1} instead of $\tilde{y}_{n+1}$ and this is iterated until the estimate converges.

Exercise: determine the order of this predictor-corrector method.

The Runge-Kutta methods

The Runge-Kutta methods are families of systematic higher order improvements over the Euler method. The key idea is to evaluate the derivative dy/dt not only at the end points t_n or t_{n+1} but also at intermediate points such as:

$$y_{n+1} = y_n + \Delta t f\left(t_n + \frac{\Delta t}{2}, y\left(t_n + \frac{\Delta t}{2}\right)\right) + \mathrm{O}(\Delta t^3). \tag{2.15}$$

The unknown solution $y(t_n + \Delta t/2)$ is again approximated by an Euler step, giving the second order Runge-Kutta algorithm:

$$\begin{aligned} k_1 &= \Delta t f(t_n, y_n) \\ k_2 &= \Delta t f(t_n + \Delta t/2, y_n + k_1/2) \\ y_{n+1} &= y_n + k_2 + \mathrm{O}(\Delta t^3) \end{aligned} \tag{2.16}$$

The general ansatz is

$$y_{n+1} = y_n + \sum_{i=1}^{N} \alpha_i k_i \tag{2.17}$$

where the approximations k_i are given by

$$k_i = \Delta t f(y_n + \sum_{j=1}^{N-1} \nu_{ij} k_i, t_n + \sum_{j=1}^{N-1} \nu_{ij} \Delta t) \tag{2.18}$$

and the parameters α_i and ν_{ij} are chosen to obtain an N-th order method. Note that this choice is usually not unique.

The most widely used Runge-Kutta algorithm is the fourth order method:

$$\begin{aligned} k_1 &= \Delta t f(t_n, y_n) \\ k_2 &= \Delta t f(t_n + \Delta t/2, y_n + k_1/2) \\ k_3 &= \Delta t f(t_n + \Delta t/2, y_n + k_2/2) \\ k_4 &= \Delta t f(t_n + \Delta t, y_n + k_3) \\ y_{n+1} &= y_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + \mathrm{O}(\Delta t^5) \end{aligned} \tag{2.19}$$

in which two estimates at the intermediate point $t_n + \Delta t/2$ are combined with one estimate at the starting point t_n and one estimate at the end point $t_n = t_0 + n\Delta t$.

Exercise: check the order of these two Runge-Kutta algorithms.

2.2 Integrating the classical equations of motion

The most common ordinary differential equation you will encounter are Newton's equation for the classical motion for N point particles:

$$\begin{aligned} m_i \frac{d\vec{v}_i}{dt} &= \vec{F}_i(t, \vec{x}_1, \ldots, \vec{x}_N, \vec{v}_1, \ldots, \vec{v}_N) & (2.20) \\ \frac{d\vec{x}_i}{dt} &= \vec{v}_i, & (2.21) \end{aligned}$$

where m_i, $\vec{v}_i$ and $\vec{x}_i$ are the mass, velocity and position of the i-th particle and $\vec{F}_i$ the force acting on this particle.

For simplicity in notation we will restrict ourselves to a single particle in one dimension, before discussing applications to the classical few-body and many-body problem. We again label the time steps $t_{n+1} = t_n + \Delta t$, and denote by x_n and v_n the approximate solutions for $x(t_n)$ and $v(t_n)$ respectively. The accelerations are given by $a_n = a(t_n, x_n, v_n) = F(t_n, x_n, v_n)/m$.

The simplest method is again the forward-Euler method

$$\begin{aligned} v_{n+1} &= v_n + a_n \Delta t \\ x_{n+1} &= x_n + v_n \Delta t. \end{aligned} \qquad (2.22)$$

which is however unstable for oscillating systems as can be seen in the Mathematica notebook on the web page. For a simple harmonic oscillator the errors will increase exponentially over time no matter how small the time step Δt is chosen and the forward-Euler method should thus be avoided!

For velocity-independent forces a surprisingly simple trick is sufficient to stabilize the Euler method. Using the backward difference $v_{n+1} \approx (x_{n+1} - x_n)/\Delta t$ instead of a forward difference $v_n \approx (x_{n+1} - x_n)/\Delta t$ we obtain the stable backward-Euler method:

$$\begin{aligned} v_{n+1} &= v_n + a_n \Delta t \\ x_{n+1} &= x_n + v_{n+1} \Delta t, \end{aligned} \qquad (2.23)$$

where the new velocity v_{n+1} is used in calculating the positions x_{n+1}.

A related stable algorithm is the mid-point method, using a central difference:

$$\begin{aligned} v_{n+1} &= v_n + a_n \Delta t \\ x_{n+1} &= x_n + \frac{1}{2}(v_n + v_{n+1}) \Delta t. \end{aligned} \qquad (2.24)$$

Equally simple, but surprisingly of second order is the leap-frog method, which is one of the commonly used methods. It evaluates positions and velocities at different times:

$$\begin{aligned} v_{n+1/2} &= v_{n-1/2} + a_n \Delta t \\ x_{n+1} &= x_n + v_{n+1/2}\Delta t. \end{aligned} \tag{2.25}$$

As this method is not self-starting the Euler method is used for the first half step:

$$v_{1/2} = v_0 + \frac{1}{2} a_0 \Delta t. \tag{2.26}$$

For velocity-dependent forces the second-order Euler-Richardson algorithm can be used:

$$\begin{aligned} a_{n+1/2} &= a\left(x_n + \frac{1}{2} v_n \Delta t,\ v_n + \frac{1}{2} a_n \Delta t,\ t_n + \frac{1}{2}\Delta t\right) \\ v_{n+1} &= v_n + a_{n+1/2}\Delta t \\ x_{n+1} &= x_n + v_n \Delta t + \frac{1}{2} a_{n+1/2} \Delta t^2. \end{aligned} \tag{2.27}$$

The most commonly used algorithm is the following form of the Verlet algorithm ("velocity Verlet"):

$$\begin{aligned} x_{n+1} &= x_n + v_n \Delta t + \frac{1}{2} a_n (\Delta t)^2 \\ v_{n+1} &= v_n + \frac{1}{2}(a_n + a_{n+1})\Delta t. \end{aligned} \tag{2.28}$$

It is third order in the positions and second order in the velocities.

2.3 Boundary value problems and "shooting"

So far we have considered only the initial value problem, where we specified both the initial position and velocity. Another type of problems is the boundary value problem where instead of two initial conditions we specify one initial and one final condition. Examples can be:

- We lanuch a rocket from the surface of the earth and want it to enter space (defined as an altitude of 100km) after one hour. Here the initial and final positions are specified and the question is to estimate the required power of the rocket engine.
- We fire a cannon ball from ETH Hnggerberg and want it to hit the tower of the university of Zürich. The initial and final positions as well as the initial speed of the cannon ball is specified. The question is to determine the angle of the cannon barrel.

Such boundary value problems are solved by the "shooting" method which should be familiar to Swiss students from their army days. In the second example we guess an angle for the cannon, fire a shot, and then iteratively adjust the angle until we hit our target.

More formally, let us again consider a simple one-dimensional example but instead of specifying the initial position x_0 and velocity v_0 we specify the initial position $x(0) = x_0$ and the final position after some time t as $x(t) = x_f$. To solve this problem we

1. guess an initial velocity $v_0 = \alpha$
2. define $x(t;\alpha)$ as the numerically integrated value of for the final position as a function of α
3. numerically solve the equation $x(t;\alpha) = x_f$

We thus have to combine one of the above integrators for the equations of motion with a numerical root solver.

2.4 Numerical root solvers

The purpose of a root solver is to find a solution (a root) to the equation

$$f(x) = 0, \tag{2.29}$$

or in general to a multi-dimensional equation

$$\vec{f}(\vec{x}) = 0. \tag{2.30}$$

Numerical root solvers should be well known from the numerics courses and we will just review three simple root solvers here. Keep in mind that in any serious calculation it is usually best to use a well optimized and tested library function over a hand-coded root solver.

2.4.1 The Newton and secant methods

The Newton method is one of best known root solvers, however it is not guaranteed to converge. The key idea is to start from a guess x_0, linearize the equation around that guess

$$f(x_0) + (x - x_0)f'(x_0) = 0 \tag{2.31}$$

and solve this linearized equation to obtain a better estimate x_1. Iterating this procedure we obtain the **Newton method**:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{2.32}$$

If the derivative f' is not known analytically, as is the case in our shooting problems, we can estimate it from the difference of the last two points:

$$f'(x_n) \approx \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}} \tag{2.33}$$

Substituting this into the Newton method (2.32) we obtain the **secant method**:

$$x_{n+1} = x_n - (x_n - x_{n-1})\frac{f(x_n)}{f(x_n) - f(x_{n-1})}. \tag{2.34}$$

The Newton method can easily be generalized to higher dimensional equations, by defining the matrix of derivatives

$$A_{ij}(\vec{x}) = \frac{\partial f_i(\vec{x})}{\partial x_j} \tag{2.35}$$

to obtain the **higher dimensional Newton method**

$$\vec{x}_{n+1} = \vec{x}_n - A^{-1}\vec{f}(\vec{x}) \tag{2.36}$$

If the derivatives $A_{ij}(\vec{x})$ are not known analytically they can be estimated through finite differences:

$$A_{ij}(\vec{x}) = \frac{f_i(\vec{x} + h_j\vec{e}_j) - f_i(\vec{x})}{h_j} \qquad \text{with} \qquad h_j \approx x_j\sqrt{\varepsilon} \tag{2.37}$$

where ε is the machine precision (about 10^{-16} for double precision floating point numbers on most machines).

2.4.2 The bisection method and regula falsi

Both the bisection method and the regula falsi require two starting values x_0 and x_1 surrounding the root, with $f(x_0) < 0$ and $f(x_1) > 0$ so that under the assumption of a continuous function f there exists at least one root between x_0 and x_1.

The **bisection method** performs the following iteration

1. define a mid-point $x_m = (x_0 + x_1)/2$.
2. if $\text{sign} f(x_m) = \text{sign} f(x_0)$ replace $x_0 \leftarrow x_m$ otherwise replace $x_1 \leftarrow x_m$

until a root is found.

The **regula falsi** works in a similar fashion:

1. estimate the function f by a straight line from x_0 to x_1 and calculate the root of this linearized function: $x_2 = (f(x_0)x_1 - f(x_1)x_0)/(f(x_1) - f(x_0)$
2. if $\text{sign} f(x_2) = \text{sign} f(x_0)$ replace $x_0 \leftarrow x_2$ otherwise replace $x_1 \leftarrow x_2$

In contrast to the Newton method, both of these two methods will always find a root.

2.4.3 Optimizing a function

These root solvers can also be used for finding an extremum (minimum or maximum) of a function $f(\vec{x})$, by looking a root of

$$\nabla f(\vec{x}) = 0. \tag{2.38}$$

While this is efficient for one-dimensional problems, but better algorithms exist.

In the following discussion we assume, without loss of generality, that we want to minimize a function. The simplest algorithm for a multi-dimensional optimization is

steepest descent, which always looks for a minimum along the direction of steepest gradient: starting from an initial guess $\vec{x}_n$ a one-dimensional minimization is applied to determine the value of λ which minimizes

$$f(\vec{x}_n + \lambda \nabla f(\vec{x}_n)) \tag{2.39}$$

and then the next guess $\vec{x}_{n+1}$ is determined as

$$\vec{x}_{n+1} = \vec{x}_n + \lambda \nabla f(\vec{x}_n) \tag{2.40}$$

While this method is simple it can be very inefficient if the "landscape" of the function f resembles a long and narrow valley: the one-dimensional minimization will mainly improve the estimate transverse to the valley but takes a long time to traverse down the valley to the minimum. A better method is the **conjugate gradient** algorithm which approximates the function locally by a paraboloid and uses the minimum of this paraboloid as the next guess. This algorithm can find the minimuim of a long and narrow parabolic valley in one iteration! For this and other, even better, algorithms we recommend the use of **library functions**.

One final word of warning is that all of these minimizers will only find a **local minimum**. Whether this local minimum is also the global minimum can never be decided by purely numerically. A necessary but never sufficient check is thus to start the minimization not only from one initial guess but to try many initial points and check for consistency in the minimum found.

2.5 Applications

In the last section of this chapter we will mention a few interesting problems that can be solved by the methods discussed above. This list is by no means complete and should just be a starting point to get you thinking about which other interesting problems you will be able to solve.

2.5.1 The one-body problem

The one-body problem was already discussed in some examples above and is well known from the introductory classical mechanics courses. Here are a few suggestions that go beyond the analytical calculations performed in the introductory mechanics classes:

Friction

Friction is very easy to add to the equations of motion by including a velocity-dependent term such as:

$$\frac{d\vec{v}}{dt} = \vec{F} - \gamma |\vec{v}|^2 \tag{2.41}$$

while this term usually makes the problem impossible to solve analytically you will see in the exercise that this poses no problem for the numerical simulation.

Another interesting extension of the problem is adding the effects of spin to a thrown ball. Spinning the ball causes the velocity of airflow differ on opposing sides. This in turn exerts leads to differing friction forces and the trajectory of the ball curves. Again the numerical simulation remains simple.

Relativistic equations of motion

It is equally simple to go from classical Newtonian equations of motion to Einsteins equation of motion in the special theory of relativity:

$$\frac{d\vec{p}}{dt} = \vec{F} \tag{2.42}$$

where the main change is that the momentum $\vec{p}$ is no longer simply $m\vec{v}$ but now

$$\vec{p} = \gamma m_0 \vec{v} \tag{2.43}$$

where m_0 is the mass at rest of the body,

$$\gamma = \sqrt{1 + \frac{|\vec{p}|^2}{m_0^2 c^2}} = \frac{1}{\sqrt{1 - \frac{|\vec{v}|^2}{c^2}}}, \tag{2.44}$$

and c the speed of light.

These equations of motion can again be discretized, for example in a forward-Euler fashion, either by using the momenta and positions:

$$\begin{aligned} \vec{x}_{n+1} &= \vec{x}_n + \frac{\vec{p}_n}{\gamma m_0}\Delta t & (2.45) \\ \vec{p}_{n+1} &= \vec{p}_n + \vec{F}_n \Delta t & (2.46) \end{aligned}$$

or using velocities and positions

$$\begin{aligned} \vec{x}_{n+1} &= \vec{x}_n + \vec{v}_n \Delta t & (2.47) \\ \vec{v}_{n+1} &= \vec{v}_n + \frac{\vec{F}_n}{\gamma m_0}\Delta t & (2.48) \end{aligned}$$

The only change in the program is a division by γ, but this small change has large consequences, one of which is that the velocity can never exceed the speed of light c.

2.5.2 The two-body (Kepler) problem

While the generalization of the integrators for equations of motion to more than one body is trivial, the two-body problem does not even require such a generalization in the case of forces that depend only on the relative distance of the two bodies, such as gravity. The equations of motion

$$\begin{aligned} m_1 \frac{d^2\vec{x}_1}{dt^2} &= \vec{F}(\vec{x}_2 - \vec{x}_1) & (2.49) \\ m_2 \frac{d^2\vec{x}_2}{dt^2} &= \vec{F}(\vec{x}_1 - \vec{x}_2) & (2.50) \end{aligned}$$

where $\vec{F}(\vec{x}_2 - \vec{x}_1) = -\vec{F}(\vec{x}_2 - \vec{x}_1)$ we can perform a transformation of coordinates to center of mass and relative motion. The important *relative* motion gives a single body problem:

$$m\frac{d^2\vec{x}}{dt^2} = \vec{F}(\vec{x}) = -\nabla V(|\vec{x}|), \tag{2.51}$$

where $\vec{x} = \vec{x}_2 - \vec{x}_1$ is the distance, $m = m_1m_2/(m_1 + m_2)$ the reduced mass, and V the potential

$$V(r) = -\frac{Gm}{r} \tag{2.52}$$

In the case of gravity the above problem is called the Kepler problem with a force

$$\vec{F}(\vec{x}) = -Gm\frac{\vec{x}}{|\vec{x}|^3} \tag{2.53}$$

and can be solved exactly, giving the famous solutions as either circles, ellipses, parabolas or hyperbolas.

Numerically we can easily reproduce these orbits but can again go further by adding terms that make an analytical solution impossible. One possibility is to consider a satellite in orbit around the earth and add **friction** due to the atmosphere. We can calculate how the satellite spirals down to earth and crashes.

Another extension is to consider effects of Einsteins **theory of general relativity**. In a lowest order expansion its effect on the Kepler problem is a modified potential:

$$V(r) = -\frac{Gm}{r}\left(1 + \frac{\vec{L}^2}{r^2}\right), \tag{2.54}$$

where $\vec{L} = m\vec{x} \times \vec{v}$ is the angular momentum and a constant of motion. When plotting the orbits including the extra $1/r^3$ term we can observe a rotation of the main axis of the elliptical orbit. The experimental observation of this effect on the orbit of Mercury was the first confirmation of Einsteins theory of general relativity.

2.5.3 The three-body problem

Next we go to three bodies and discuss a few interesting facts that can be checked by simulations.

Stability of the three-body problem

Stability, i.e. that a small perturbation of the initial condition leads only to a small change in orbits, is easy to prove for the Kepler problem. There are 12 degrees of freedom (6 positions and 6 velocities), but 11 integrals of motion:

- total momentum: 3 integrals of motion
- angular momentum: 3 integrals of motion
- center of mass: 3 integrals of motion

- Energy: 1 integral of motion
- Lenz vector: 1 integral of motion

There is thus only one degree of freedom, the initial position on the orbit, and stability can easily be shown.

In the three-body problem there are 18 degrees of freedom but only 10 integrals of motion (no Lenz vector), resulting in 8 degrees of freedom for the orbits. Even restricting the problem to planar motions in two dimensions does not help much: 12 degrees of freedom and 6 integrals of motion result in 6 degrees of freedom for the orbits.

Progress can be made only for the *restricted three-body problem*, where the mass of the third body $m_3 \to 0$ is assumed to be too small to influence the first two bodies which are assumed to be on circular orbits. This restricted three-body problem has four degrees of freedom for the third body and one integral of motion, the energy. For the resulting problem with three degrees of freedom for the third body the famous KAM (Kolmogorov-Arnold-Moser) theorem can be used to prove stability of moon-like orbits.

Lagrange points and Trojan asteroids

In addition to moon-like orbits, other (linearly) stable orbits are around two of the Lagrange points. We start with two bodies on circular orbits and go into a rotating reference frame at which these two bodies are at rest. There are then five positions, the five Lagrange points, at which a third body is also at rest. Three of these are colinear solutions and are unstable. The other two stable solutions form equilateral triangles.

Astronomical observations have indeed found a group of asteroids, the Trojan asteroids on the orbit of Jupiter, 60 degrees before and behind Jupiter. They form an equilateral triangle with the sun and Jupiter.

Numerical simulations can be performed to check how long bodies close to the perfect location remain in stable orbits.

Kirkwood gaps in the rings of Saturn

Going farther away from the sun we next consider the Kirkwood gaps in the rings of Saturn. Simulating a system consisting of Saturn, a moon of Saturn, and a very light ring particle we find that orbits where the ratio of the period of the ring particle to that of the moon are unstable, while irrational ratios are stable.

The moons of Uranus

Uranus is home to an even stranger phenomenon. The moons Janus and Epimetheus share the same orbit of 151472 km, separated by only 50km. Since this separation is less than the diameter of the moons (ca. 100-150km) one would expect that the moons would collide.

Since these moons still exist something else must happen and indeed a simulation clearly shows that the moons do not collide but instead switch orbits when they approach each other!

2.5.4 More than three bodies

Having seen these unusual phenomena for three bodies we can expect even stranger behavior for four or five bodies, and we encourage you to start exploring them with your programs.

Especially noteworthy is that for five bodies there are extremely unstable orbits that diverge in finite time: five bodies starting with the right initial positions and finite velocities can be infinitely far apart, and flying with infinite velocities after finite time! For more information see `http://www.ams.org/notices/199505/saari-2.pdf`

Chapter 3

Partial Differential Equations

In this chapter we will present algorithms for the solution of some simple but widely used partial differential equations (PDEs), and will discuss approaches for general partial differential equations. Since we cannot go into deep detail, interested students are referred to the lectures on numerical solutions of differential equations offered by the mathematics department.

3.1 Finite differences

As in the solution of ordinary differential equations the first step in the solution of a PDE is to discretize space and time and to replace differentials by differences, using the notation $x_n = n\Delta x$. We already saw that a first order differential $\partial f/\partial x$ can be approximated in first order by

$$\frac{\partial f}{\partial x} = \frac{f(x_{n+1}) - f(x_n)}{\Delta x} + \mathrm{O}(\Delta x) = \frac{f(x_n) - f(x_{n-1})}{\Delta x} + \mathrm{O}(\Delta x) \tag{3.1}$$

or to second order by the symmetric version

$$\frac{\partial f}{\partial x} = \frac{f(x_{n+1}) - f(x_{n-1})}{2\Delta x} + \mathrm{O}(\Delta x^2), \tag{3.2}$$

From these first order derivatives can get a second order derivative as

$$\frac{\partial^2 f}{\partial x^2} = \frac{f(x_{n+1}) + f(x_{n-1}) - 2f(x_n)}{\Delta x^2} + \mathrm{O}(\Delta x^2). \tag{3.3}$$

To derive a general approximation for an arbitrary derivative to any given order use the ansatz

$$\sum_{k=-l}^{l} a_k f(x_{n+k}), \tag{3.4}$$

insert the Taylor expansion

$$f(x_{n+k}) = f(x_n) + \Delta x f'(x_n) + \frac{\Delta x^2}{2} f''(x_n) + \frac{\Delta x^3}{6} f'''(x_n) + \frac{\Delta x^4}{4} f^{(4)}(x_n) + \ldots \tag{3.5}$$

and choose the values of a_k so that all terms but the desired derivative vanish.

As an example we give the fourth-order estimator for the second derivative

$$\frac{\partial^2 f}{\partial x^2} = \frac{-f(x_{n-2}) + 16f(x_{n-1}) - 30f(x_n) + 16f(x_{n+1}) - f(x_{n+2})}{12\Delta x^2} + \mathrm{O}(\Delta x^4). \tag{3.6}$$

and the second order estimator for the third derivative:

$$\frac{\partial^3 f}{\partial x^3} = \frac{-f(x_{n-2}) + 2f(x_{n-1}) - 2f(x_{n+1}) + f(x_{n+2})}{\Delta x^3} + \mathrm{O}(\Delta x^2). \tag{3.7}$$

Extensions to higher dimensions are straightforward, and these will be all the differential quotients we will need in this course.

3.2 Solution as a matrix problem

By replacing differentials by differences we convert the (non)-linear PDE to a system of (non)-linear equations. The first example to demonstrate this is determining an electrostatic or gravitational potential Φ given by the Poisson equation

$$\nabla^2\Phi(\vec{x}) = -4\pi\rho(\vec{x}), \tag{3.8}$$

where ρ is the charge or mass density respectively and units have been chosen such that the coupling constants are all unity.

Discretizing space we obtain the system of linear equations

$$\begin{aligned} \Phi(x_{n+1}, y_n, z_n) + \Phi(x_{n-1}, y_n, z_n) & \\ +\Phi(x_n, y_{n+1}, z_n) + \Phi(x_n, y_{n-1}, z_n) & \\ +\Phi(x_n, y_n, z_{n+1}) + \Phi(x_n, y_n, z_{n-1}) & \\ -6\Phi(x_n, y_n, z_n) & = -4\pi\rho(x_n, y_n, z_n)\Delta x^2, \end{aligned} \tag{3.9}$$

where the density $\rho(x_n, y_n, z_n)$ is defined to be the average density in the cube with linear extension Δx around the point $\rho(x_n, y_n, z_n)$.

The general method to solve a PDE is to formulate this linear system of equations as a matrix problems and then to apply a linear equation solver to solve the system of equations. For small linear problems Mathematica can be used, or the `dsysv` function of the LAPACK library.

For larger problems it is essential to realize that the matrices produced by the discretization of PDEs are usually very sparse, meaning that only $\mathrm{O}(N)$ of the N^2 matrix elements are nonzero. For these sparse systems of equations, optimized iterative numerical algorithms exist[1] and are implemented in numerical libraries such as in the ITL library.[2]

[1] R. Barret, M. Berry, T.F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods* (SIAM, 1993)

[2] J.G. Siek, A. Lumsdaine and Lie-Quan Lee, *Generic Programming for High Performance Numerical Linear Algebra* in *Proceedings of the SIAM Workshop on Object Oriented Methods for Inter-operable Scientific and Engineering Computing (OO'98)* (SIAM, 1998); the library is availavle on the web at: `http://www.osl.iu.edu/research/itl/`

This is the most general procedure and can be used in all cases, including boundary value problems and eigenvalue problems. The PDE eigenvalue problem maps to a matrix eigenvalue problem, and an eigensolver needs to be used instead of a linear solver. Again there exist efficient implementations[3] of iterative algorithms for sparse matrices.[4]

For non-linear problems iterative procedures can be used to linearize them, as we will discuss below.

Instead of this general and flexible but brute-force method, many common PDEs allow for optimized solvers that we will discuss below.

3.3 The relaxation method

For the Poisson equation a simple iterative method exists that can be obtained by rewriting above equation as

$$\begin{aligned}\Phi(x_n, y_n, z_n) &= \frac{1}{6}[\Phi(x_{n+1}, y_n, z_n) + \Phi(x_{n-1}, y_n, z_n) + \Phi(x_n, y_{n+1}, z_n) \\ &\quad +\Phi(x_n, y_{n-1}, z_n) + \Phi(x_n, y_n, z_{n+1}) + \Phi(x_n, y_n, z_{n-1})] \\ &\quad -\frac{2}{3}\pi\rho(x_n, y_n, z_n)\Delta x^2, \qquad (3.10)\end{aligned}$$

The potential is just the average over the potential on the six neighboring sites plus a term proportinal to the density ρ.

A solution can be obtained by iterating equation 3.10:

$$\begin{aligned}\Phi(x_n, y_n, z_n) &\leftarrow \frac{1}{6}[\Phi(x_{n+1}, y_n, z_n) + \Phi(x_{n-1}, y_n, z_n) + \Phi(x_n, y_{n+1}, z_n) \\ &\quad +\Phi(x_n, y_{n-1}, z_n) + \Phi(x_n, y_n, z_{n+1}) + \Phi(x_n, y_n, z_{n-1})] \\ &\quad -\frac{2}{3}\pi\rho(x_n, y_n, z_n)\Delta x^2, \qquad (3.11)\end{aligned}$$

This iterative solver will be implemented in the exercises for two examples:

1. Calculate the potential between two concentric metal squares of size a and $2a$. The potential difference between the two squares is V. Starting with a potential 0 on the inner square, V on the outer square, and arbitrary values in-between, a two-dimensional variant of equation 3.11 is iterated until the differences drop below a given threshold. Since there are no charges the iteration is simply:

$$\Phi(x_n, y_n) \leftarrow \frac{1}{4}[\Phi(x_{n+1}, y_n) + \Phi(x_{n-1}, y_n) + \Phi(x_n, y_{n+1}) + \Phi(x, y_{n-1})]. \qquad (3.12)$$

2. Calculate the potential of a distribution of point charges: starting from an arbitrary initial condition, e.g. $\Phi(x_n, y_n, z_n) = 0$, equation 3.11 is iterated until convergence.

[3]http://www.comp-phys.org/software/ietl/

[4]Z. Bai, J. Demmel and J. Dongarra (Eds.), *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide* (SIAM, 2000).

Since these iterations are quite slow it is important to improve them by one of two methods discussed below.

3.3.1 Gauss-Seidel Overrelaxtion

Gauss-Seidel overrelaxtion determines the change in potential according to equation 3.11 but then changes the value by a multiple of this proposed change:

$$\begin{aligned}\Delta\Phi(x_n, y_n, z_n) &= \frac{1}{6}[\Phi(x_{n+1}, y_n, z_n) + \Phi(x_{n-1}, y_n, z_n) + \Phi(x_n, y_{n+1}, z_n) \\ &\quad +\Phi(x_n, y_{n-1}, z_n) + \Phi(x_n, y_n, z_{n+1}) + \Phi(x_n, y_n, z_{n-1})] \\ &\quad -\frac{2}{3}\pi\rho(x_n, y_n, z_n)\Delta x^2 - \Phi(x_n, y_n, z_n) \\ \Phi(x_n, y_n, z_n) &\leftarrow \Phi(x_n, y_n, z_n) + w\Delta\Phi(x_n, y_n, z_n) \qquad (3.13)\end{aligned}$$

with an overrelaxation factor of $1 < w < 2$. You can easily convince yourself, by considering a single charge and initial values of $\Phi(x_n, y_n, z_n) = 0$ that choosing value $w \geq 2$ is unstable.

3.3.2 Multi-grid methods

Multi-grid methods dramatically accelerate the convergence of many iterative solvers. We start with a very coarse grid spacing $\Delta x \Delta x_0$ and iterate

- solve the Poisson equation on the grid with spacing Δx
- refine the grid $\Delta x \leftarrow \Delta x/2$
- interpolate the potential at the new grid points
- and repeat until the desired final fine grid spacing Δx is reached.

Initially convergence is fast since we have a very small lattice. In the later steps convergence remains fast since we always start with a very good guess.

3.4 Solving time-dependent PDEs by the method of lines

3.4.1 The diffusion equation

Our next problem will include a first order time-derivative, with a partial differential equation of the form

$$\frac{\partial f(\vec{x}, t)}{\partial t} = F(f, t) \qquad (3.14)$$

where f contains only spatial derivatives and the initial condition at time t_0 is given by

$$f(\vec{x}, t_0) = u(\vec{x}). \qquad (3.15)$$

One common equation is the diffusion equation, e.g. for heat transport

$$\frac{\partial T(\vec{x},t)}{\partial t} = -\frac{K}{C\rho}\nabla^2 T(\vec{x},t) + \frac{1}{C\rho}W(\vec{x},t) \tag{3.16}$$

where T is the temperature, C the specific heat, ρ the density and K the thermal conductivity. External heat sources or sinks are specified by $W(\vec{x},t)$.

This and similar initial value problems can be solved by the method of lines: after discretizing the spatial derivatives we obtain a set of coupled ordinary differential equations which can be evolved fort each point along the time line (hence the name) by standard ODE solvers. In our example we obtain, specializing for simplicity to the one-dimensional case:

$$\frac{\partial T(x_n,t)}{\partial t} = -\frac{K}{C\rho\Delta x^2}\left[T(x_{n+1},t) + T(x_{n-1},t) - 2T(x_n,t)\right] + \frac{1}{C\rho}W(x_n,t) \tag{3.17}$$

Using a forward Euler algorithm we finally obtain

$$T(x_n,t+\Delta t) = T(x_n,t) - \frac{K\Delta t}{C\rho\Delta x^2}\left[T(x_{n+1},t) + T(x_{n-1},t) - 2T(x_n,t)\right] + \frac{\Delta t}{C\rho}W(x_n,t) \tag{3.18}$$

This will be implemented in the exercises and used in the supercomputing examples.

3.4.2 Stability

Great care has to be taken in choosing appropriate values of Δx and Δt, as too long time steps Δt immediately lead to instabilities. By considering the case where the temperature is 0 everywhere except at one point it is seen immediately, like in the case of overrelaxation that a choice of $K\Delta t/C\rho\Delta x^2 > 1/2$ is unstable. A detailed analysis, which is done e.g. in the lectures on numerical solutions of differential equations, shows that this heat equation solver is only stable for

$$\frac{K\Delta t}{C\rho\Delta x^2} < \frac{1}{4}. \tag{3.19}$$

We see that, for this PDE with second order spatial and first order temporal derivatives, it is not enough to just adjust Δt proportional to Δx, but $\Delta t \ll \mathrm{O}(\Delta x^2)$ is needed. Here it is even more important to **check for instabilities** than in the case of PDEs!

3.4.3 The Crank-Nicolson method

The simple solver above can be improved by replacing the forward Euler method for the time integration by a midpoint method:

$$T(x,t+\Delta t) = T(x,t) + \frac{K\Delta t}{2C\rho}\left[\nabla^2 T(x,t) + \nabla^2 T(x,t+\Delta t)\right] + \frac{\Delta t}{2C\rho}\left[W(x,t) + W(x,t+\Delta t)\right] \tag{3.20}$$

Discretizing space and introducing the linear operator A defined by

$$AT(x_n,t) = \frac{K\Delta t}{C\rho\Delta x^2}\left[T(x_{n+1},t) + T(x_{n-1},t) - 2T(x_n,t)\right] \tag{3.21}$$

to simplify the notation we obtain an *implicit* algorithm:

$$(2 \cdot \mathbf{1} - A)\vec{T}(t+\Delta t) = (2 - A)\vec{T}(t) + \frac{\Delta t}{C\rho}\left[\vec{W}(t) + \vec{W}(t+\Delta t)\right], \tag{3.22}$$

where $\mathbf{1}$ is the unit matrix and

$$\vec{T}(t) = (T(x_1,t), \dots T(x_N,t)) \tag{3.23}$$

$$\vec{W}(t) = (W(x_1,t), \dots W(x_N,t)) \tag{3.24}$$

are vector notations for the values of the temperature and heat source at each point. In contrast to the *explicit* solver (3.18) the values at time $t+\Delta t$ are not given explicitly on the right hand side but only as a solution to a linear system of equations. After evaluating the right hand side, still a linear equation needs to be solved. This extra effort, however, gives us greatly improved stability and accuracy.

Note that while we have discussed the Crank-Nicolson method here in the context of the diffusion equation, it can be applied to any time-dependent PDE.

3.5 The wave equation

3.5.1 A vibrating string

Another simple PDE is the wave equation, which we will study for the case of a string running along the x-direction and vibrating transversely in the y-direction:

$$\frac{\partial^2 y}{\partial t^2} = c^2 \frac{\partial^2 y}{\partial x^2}. \tag{3.25}$$

The wave velocity $c = \sqrt{T/\mu}$ is a function of the string tension T and the mass density μ of the string.

As you can easily verify, analytic solutions of this wave equation are of the form

$$y = f_+(x+ct) + f_-(x-ct). \tag{3.26}$$

To solve the wave equation numerically we again discretize time and space in the usual manner and obtain, using the the second order difference expressions for the second derivative:

$$\frac{y(x_i,t_{n+1}) + y(x_i,t_{n-1}) - 2y(x_i,t_n)}{(\Delta t)^2} \approx c^2 \frac{y(x_{i+1},t_n) + y(x_{i-1},t_n) - 2y(x_i,t_n)}{(\Delta x)^2}. \tag{3.27}$$

This can be transformed to

$$y(x_i,t_{n+1}) = 2(1-\kappa^2)y(x_i,t_n) - y(x_i,t_{n-1}) + \kappa^2\left[y(x_{i+1},t_n) + y(x_{i-1},t_n)\right], \tag{3.28}$$

with $\kappa = c\Delta t/\Delta x$.

Again, we have to choose the values of Δt and Δx carefully. Surprisingly, for the wave equation when choosing $\kappa = 1$ we obtain the exact solution without any error! To check this, insert the exact solution (3.26) into the difference equation (3.27).

Decreasing both Δx and Δt does not increase the accuracy but only the spatial and temporal resolution. This is a very special feature of the linear wave equation.

Choosing smaller time steps and thus $\kappa < 1$ there will be solutions propagating faster than the speed of light, but since they decrease with the square of the distance r^{-2} this does not cause any major problems.

On the other hand, choosing a slightly larger time step and thus $\kappa > 1$ has catastrophic consequences: these unphysical numerical solution *increase* and diverge rapidly, as can be seen in the Mathematica Notebook posted on the web page.

3.5.2 More realistic models

Real strings and musical instruments cause a number of changes and complications to the simple wave equation discussed above:

- Real strings vibrate in both the y and z direction, which is easy to implement.
- Transverse vibrations of the string cause the length of the string and consequently the string tension T to increase. This leads to an increase of the velocity c and the value of κ. The nice fact that $\kappa = 1$ gives the exact solution can thus no longer be used and special care has to be taken to make sure that $\kappa < 1$ even for the largest elongations of the string.
- Additionally there will be longitudinal vibrations of the string, with a much higher velocity $c_{||} \gg c$. Consequently the time step for longitudinal vibrations $\Delta t_{||}$ has to be chosen much smaller than for transverse vibrations. Instead of of simulating both transverse and longitudinal vibrations with the same small time step $\Delta t_{||}$ one still uses the larger time step Δt for the transverse vibrations but updates the transverse positions only every $\Delta t/\Delta t_{||}$ iterations of the longitudinal positions.
- Finally the string is not in vacuum and infinitely long, but in air and attached to a musical instrument. Both the friction of the air and forces exerted by the body of the instrument cause damping of the waves and a modified sound.

For more information about applications to musical instruments I refer to the article by N. Giordano in Computers in Phsyics **12**, 138 (1998). This article also discusses numerical approaches to the following problems

- How is sound created in an acoustic guitar, an electric guitar and a piano?
- What is the sound of these instruments?
- How are the strings set into motion in these instruments?

The simulation of complex instruments such as pianos still poses substantial unsolved challenges.

3.6 The finite element method

3.6.1 The basic finite element method

While the finite difference method used so far is simple and straightforward for regular mesh discretizations it becomes very hard to apply to more complex problems such as:

- spatially varying constants, such as spatially varying dielectric constants in the Poisson equation.
- irregular geometries such as airplanes or turbines.
- dynamically adapting geometries such as moving pistons.

In such cases the finite element method has big advantages over finite differences since it does not rely on a regular mesh discretization. We will discuss the finite element method using the one-dimensional Poisson equation

$$\phi''(x) = -4\pi\rho(x) \tag{3.29}$$

with boundary conditions

$$\phi(0) = \phi(1) = 0. \tag{3.30}$$

as our example.

The first step is to expand the solution $\phi(x)$ in terms of basis functions $\{v_i\}$, $i = 1, \ldots, \infty$ of the function space:

$$\phi(x) = \sum_{i=1}^{\infty} a_i v_i(x). \tag{3.31}$$

For our numerical calculation the infinite basis set needs to be truncated, choosing a finite subset $\{u_i\}$, $i = 1, \ldots, N$ of N linearly independent, but not necessarily orthogonal, functions:

$$\phi_N(x) = \sum_{i=1}^{N} a_i u_i(x). \tag{3.32}$$

The usual choice are functions localized around some mesh points x_i, which in contrast to the finite difference method do not need to form a regular mesh.

The coefficients $\vec{a} = (a_1, \ldots, a_N)$ are chosen to minimize the residual

$$\phi_N''(x) + 4\pi\rho(x) \tag{3.33}$$

over the whole interval. Since we can choose N coefficients we can impose N conditions

$$0 = g_i \int_0^1 \left[\phi_N''(x) + 4\pi\rho(x)\right] w_i(x)dx, \tag{3.34}$$

where the weight functions $w_i(x)$ are often chosen to be the same as the basis functions $w_i(x) = u_i(x)$. This is called the Galerkin method.

In the current case of a linear PDE this results in a linear system of equations

$$A\vec{a} = \vec{b} \tag{3.35}$$

with

$$\begin{aligned} A_{ij} &= -\int_0^1 u_i''(x)w_j(x)dx = \int_0^1 u_i'(x)w_j'(x)dx \\ b_i &= 4\pi\int_0^1 \rho(x)w_i(x)dx, \end{aligned} \tag{3.36}$$

where in the first line we have used integration by parts to circumvent problems with functions that are not twice differentiable.

A good and simple choice of local basis functions fulfilling the boundary conditions (3.30) are local triangles centered over the points $x_i = i\Delta x$ with $\Delta x = 1/(n+1)$:

$$u_i(x) = \begin{cases} (x - x_{i-1})/\Delta x & \text{for } x \in [x_i - 1, x_i] \\ (x_{i+1} - x)/\Delta x & \text{for } x \in [x_i, x_i + 1] \\ 0 & \text{otherwise} \end{cases}, \tag{3.37}$$

but other choices such as local parabolas are also possible.

With the above choice we obtain

$$A_{ij} = -\int_0^1 u_i''(x)u_j(x)dx = \int_0^1 u_i'(x)u_j'(x)dx = \begin{cases} 2/\Delta x & \text{for } i = j \\ -1/\Delta x & \text{for } i = j \pm 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.38}$$

and, choosing a charge density $\rho(x) = (\pi/4)\sin(\pi x)$

$$b_i = 4\pi\int_0^1 \rho(x)u_i(x)dx = \frac{1}{\Delta x}\left(2\sin\pi x_i - \sin\pi x_{i-1} - \sin\pi x_{i+1}\right) \tag{3.39}$$

In the one-dimensional case the matrix A is tridiagonal and efficient linear solvers for this tridiagonal matrix can be found in the LAPACK library. In higher dimensions the matrices will usually be sparse band matrices and iterative solvers will be the methods of choice.

3.6.2 Generalizations to arbitrary boundary conditions

Our example assumed boundary conditions $\phi(0) = \phi(1) = 0$. These boundary conditions were implemented by ensuring that all basis functions $u_i(x)$ were zero on the boundary. Generalizations to arbitrary boundary conditions $\phi(0) = \phi_0$ and $\phi(1) = \phi_1$ are possible either by adding additional basis functions that are non-zero at the boundary or be starting from a generalized ansatz that automatically ensures the correct boundary conditions, such as

$$\phi_N(x) = \phi_0(1-x) + \phi_1 x \sum_{i=1}^{N} a_i u_i(x). \tag{3.40}$$

3.6.3 Generalizations to higher dimensions

Generalizations to higher dimensions are done by

- creating higher-dimensional meshes

- and providing higher-dimensional basis functions, such as pyramids centered on a mesh point.

While the basic principles remain the same, stability problems can appear at sharp corners and edges and for time-dependent geometries. The creation of appropriate meshes and basis functions is an art in itself, and an important area of industrial research. Interested students are referred to advanced courses on the subject of finite element methods

3.6.4 Nonlinear partial differential equations

The finite element method can also be applied to non-linear partial differential equations without any big changes. Let us consider a simple example

$$\phi(x)\frac{d^2\phi}{dx^2}(x) = -4\pi\rho(x) \tag{3.41}$$

Using the same ansatz, Eq. (3.32) as before and minimizing the residuals

$$g_i = \int_0^1 \left[\phi\phi''(x) + 4\pi\rho(x)\right] w_i(x)dx \tag{3.42}$$

as before we now end up with a nonlinear equation instead of a linear equation:

$$\sum_{i,j} A_{ijk}a_i a_j = b_k \tag{3.43}$$

with

$$A_{ijk} = -\int_0^1 u_i(x)u_j''(x)w_k(x)dx \tag{3.44}$$

and b_k defined as before.

The only difference between the case of linear and nonlinear partial differential equations is that the former gives a set of coupled linear equations, while the latter requires the solution of a set of coupled nonlinear equations.

Often, a Picard iteration can be used to transform the nonlinear problem into a linear one. In our case we can start with a crude guess $\phi_0(x)$ for the solution and use that guess to linearize the problem as

$$\phi_0(x)\frac{d^2\phi_1}{dx^2}(x) = -4\pi\rho(x) \tag{3.45}$$

to obtain a better solution ϕ_1. Replacing ϕ_0 by ϕ_1 an iterating the procedure by solving

$$\phi_n(x)\frac{d^2\phi_{n+1}}{dx^2}(x) = -4\pi\rho(x) \tag{3.46}$$

for ever better solutions ϕ_{n+1} we converge to the solution of the nonlinear partial differential equation by solving a series of linear partial differential equations.

3.7 Maxwell's equations

The last linear partial differential equation we will consider in this section are Maxwell's equations for the electromagnetic field. We will first calculate the field created by a single charged particle and then solve Maxwell's equations for the general case.

3.7.1 Fields due to a moving charge

The electric potential at the location $\vec{R}$ due to a single static charge q at the position $\vec{r}$ can directly be written as

$$V(\vec{R}) = \frac{q}{|\vec{r} - \vec{R}|}, \tag{3.47}$$

and the electric field calculated from it by taking the gradient $\vec{E} = -\nabla V$.

When calculating the fields due to moving charges one needs to take into account that the electromagnetic waves only propagate with the speed of light. It is thus necessary to find the retarded position

$$r_{\mathrm{ret}} = \left|\vec{R} - \vec{r}(t_{\mathrm{ret}})\right| \tag{3.48}$$

and time

$$t_{\mathrm{ret}} = t - \frac{r_{\mathrm{ret}}(t_{\mathrm{ret}})}{c} \tag{3.49}$$

so that the distance r_{ret} of the particle at time t_{ret} was just ct_{ret}. Given the path of the particle this just requires a root solver. Next, the potential can be calculated as

$$V(\vec{R}, t) = \frac{q}{r_{\mathrm{ret}}\left(1 - \hat{r}_{\mathrm{ret}} \cdot \vec{v}_{\mathrm{ret}}/c\right)} \tag{3.50}$$

with the retarded velocity given by

$$\vec{v}_{\mathrm{ret}} = \left.\frac{d\vec{r}(t)}{dt}\right|_{t=t_{\mathrm{ret}}} \tag{3.51}$$

The electric and magnetic field then work out as

$$\begin{aligned} \vec{E}(\vec{R}, t) &= \frac{q r_{\mathrm{ret}}}{\vec{r}_{\mathrm{ret}} \vec{u}_{\mathrm{ret}}} \left[\vec{u}_{\mathrm{ret}}\left(c^2 - v_{\mathrm{ret}}^2\right) + \vec{r}_{\mathrm{ret}} \times \left(\vec{u}_{\mathrm{ret}} \times \vec{a}_{\mathrm{ret}}\right)\right] & (3.52) \\ \vec{B}(\vec{R}, t) &= \hat{r}_{\mathrm{ret}} \times \vec{E}(\vec{R}, t) & (3.53) \end{aligned}$$

with

$$\vec{a}_{\mathrm{ret}} = \left.\frac{d^2\vec{r}(t)}{dt^2}\right|_{t=t_{\mathrm{ret}}} \tag{3.54}$$

and

$$\vec{u}_{\mathrm{ret}} = c\hat{r}_{\mathrm{ret}} - \vec{v}_{\mathrm{ret}}. \tag{3.55}$$

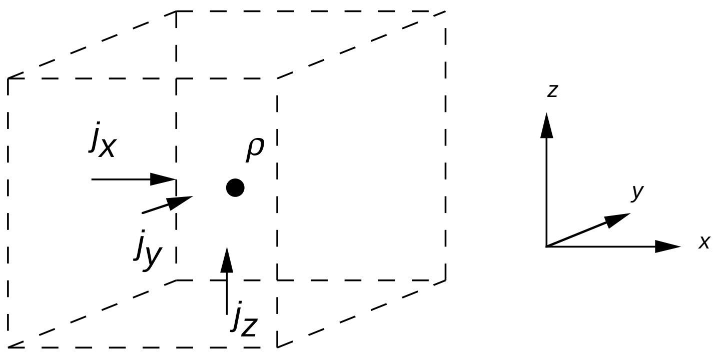


Figure 3.1: Definition of charges and currents for the Yee-Vischen algorithm

3.7.2 The Yee-Vischen algorithm

For the case of a single moving charge solving Maxwell's equation just required a root solver to determine the retarded position and time. In the general case of many particles it will be easier to directly solve Maxwell's equations, which (setting $\epsilon_0 = \mu_0 = c = 1$) read

$$\frac{\partial \vec{B}}{\partial t} = -\nabla \times \vec{E} \tag{3.56}$$

$$\frac{\partial \vec{E}}{\partial t} = \nabla \times \vec{B} - 4\pi \vec{j} \tag{3.57}$$

$$\frac{\partial \rho}{\partial t} = -\nabla \cdot \vec{j} \tag{3.58}$$

The numerical solution starts by dividing the volume into cubes of side length Δx, as shown in figure 3.7.2 and defining by $\rho(\vec{x})$ the total charge inside the cube.

Next we need to define the currents flowing between cubes. They are most naturally defined as flowing perpendicular to the faces of the cube. Defining as $j_x(\vec{x})$ the current flowing into the box from the left, $j_y(\vec{x})$ the current from the front and $j_z(\vec{x})$ the current from the bottom we can discretize the continuity equation (3.58) using a half-step method

$$\rho(\vec{x}, t + \Delta t/2) = \rho(\vec{x}, t + \Delta t/2) - \frac{\Delta t}{\Delta x} \sum_{f=1}^{6} j_f(\vec{x}, t). \tag{3.59}$$

The currents through the faces $j_f(\vec{x}, t)$ are defined as

$$\begin{aligned} j_1(\vec{x}, t) &= -j_x(\vec{x}, t) \\ j_2(\vec{x}, t) &= -j_y(\vec{x}, t) \\ j_3(\vec{x}, t) &= -j_z(\vec{x}, t) \\ j_4(\vec{x}, t) &= j_x(\vec{x} + \Delta x \hat{e}_x, t) \\ j_5(\vec{x}, t) &= j_y(\vec{x} + \Delta x \hat{e}_y, t) \\ j_6(\vec{x}, t) &= j_z(\vec{x} + \Delta x \hat{e}_z, t). \end{aligned} \tag{3.60}$$

Be careful with the signs when implementing this.

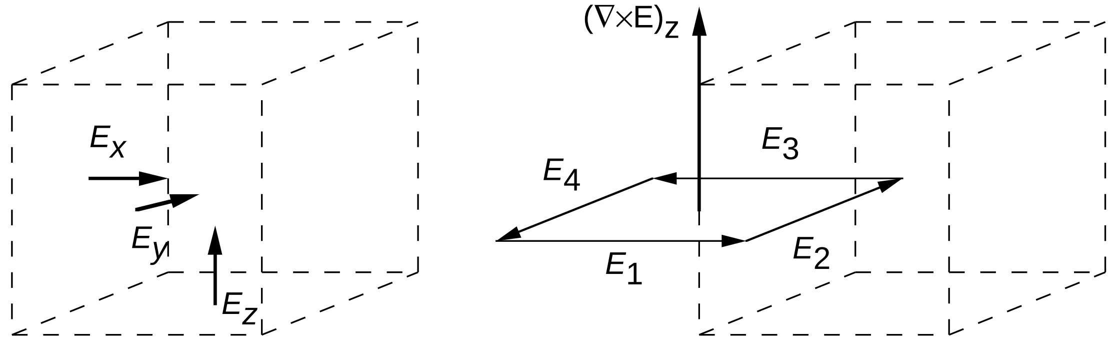


Figure 3.2: Definition of electric field and its curl for the Yee-Vischen algorithm.

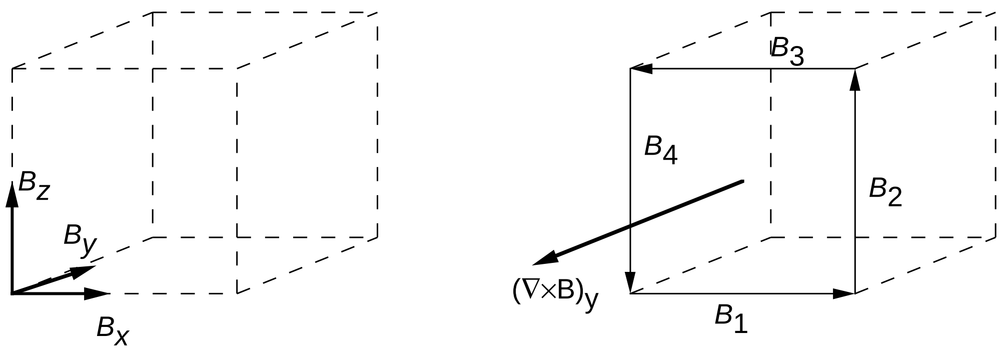


Figure 3.3: Definition of magnetic field and its curl for the Yee-Vischen algorithm.

Next we observe that equation (3.57) for the electric field $\vec{E}$ contains a term proportional to the currents j and we define the electric field also perpendicular to the faces, but offset by a half time step. The curl of the electric field, needed in equation (3.56) is then most easily defined on the edges of the cube, as shown in figure 3.7.2, again taking care of the signs when summing the electric fields through the faces around an edge.

Finally, by noting that the magnetic field term (3.56) contains terms proportional to the curl of the electric field we also define the magnetic field on the edges of the cubes, as shown in figure 3.7.2. We then obtain for the last two equations:

$$\vec{E}(\vec{x}, t+\Delta t/2) = \vec{E}(\vec{x}, t+\Delta t/2) + \frac{\Delta t}{\Delta x}\left[\sum_{e=1}^{4} \vec{B}_e(\vec{x}, t) - 4\pi\vec{j}(\vec{x}, t)\right] \tag{3.61}$$

$$\vec{B}(\vec{x}, t+\Delta t) = \vec{B}(\vec{x}, t) - \frac{\Delta t}{\Delta x}\sum_{f=1}^{4} \vec{E}_f(\vec{x}, t+\Delta t/2) \tag{3.62}$$

which are stable if $\Delta t/\Delta x \leq 1/\sqrt{3}$.

3.8 Hydrodynamics and the Navier Stokes equation

3.8.1 The Navier Stokes equation

The Navier Stokes equation is one of the most famous, if not the most famous set of partial differential equations. They describe the flow of a classical Newtonian fluid.

The first equation describing the flow of the fluid is the continuity equation, describing conservation of mass:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{v}) = 0 \tag{3.63}$$

where ρ is the local mass density of the fluid and $\vec{v}$ its velocity. The second equation is the famous Navier-Stokes equation describing the conservation of momentum:

$$\frac{\partial}{\partial t}(\rho \vec{v}) + \nabla \cdot \Pi = \rho \vec{g} \tag{3.64}$$

where $\vec{g}$ is the force vector of the gravitational force coupling to the mass density, and Π_{ij} is the momentum tensor

$$\Pi_{ij} = \rho v_i v_j - \Gamma_{ij} \tag{3.65}$$

with

$$\Gamma_{ij} = \eta \left[\partial_i v_j + \partial_j v_i\right] + \left[\left(\zeta - \frac{2\eta}{3}\right) \nabla \cdot \vec{v} - P\right] \delta_{ij}. \tag{3.66}$$

The constants η and ζ describe the shear and bulk viscosity of the fluid, and P is the local pressure.

The third and final equation is the energy transport equation, describing conservation of energy:

$$\frac{\partial}{\partial t}\left(\rho \epsilon + \frac{1}{2}\rho v^2\right) + \nabla \cdot \vec{j}_e = 0 \tag{3.67}$$

where ϵ is the local energy density, the energy current is defined as

$$\vec{\mathrm{j}}_{\mathrm{e}} = \vec{v}\left(\rho \epsilon + \frac{1}{2}\rho v^2\right) - \vec{v} \cdot \Gamma - \kappa \nabla k_B T, \tag{3.68}$$

where T is the temperature and κ the heat conductivity.

The only nonlinearity arises from the momentum tensor Π_{ij} in equation (3.65). In contrast to the linear equations studied so far, where we had nice and smoothly propagating waves with no big surprises, this nonlinearity causes the fascinating and complex phenomenon of turbulent flow.

Despite decades of research and the big importance of turbulence in engineering it is still not completely understood. Turbulence causes problems not only in engineering applications but also for the numerical solution, with all known numerical solvers becoming unstable in highly turbulent regimes. Its is then hard to distinguish the chaotic effects caused by turbulence from chaotic effects caused by an instability of the numerical solver. In fact the question of finding solutions to the Navier Stokes equations, and whether it is even possible at all, has been nominated as one of the seven millennium challenges in mathematics, and the Clay Mathematics Institute (`http:/www.claymath.org/`) has offered a prize money of one million US$ for solving the Navier-Stokes equation or for proving that they cannot be solved.

Just keep these convergence problems and the resulting unreliability of numerical solutions in mind the next time you hit a zone of turbulence when flying in an airplane, or read up on what happened to American Airlines flight AA 587.

3.8.2 Isothermal incompressible stationary flows

For the exercises we will look at a simplified problem, the special case of an isothermal (constant T) static ($\partial/\partial T = 0$) flow of an incompressible fluid (constant ρ). In this case the Navier-Stokes equations simplify to

$$\rho\vec{v}\cdot\nabla\vec{v} + \nabla P - \eta\nabla^2\vec{v} = \rho\vec{g} \qquad (3.69)$$

$$\nabla\cdot\vec{v} = 0 \qquad (3.70)$$

In this stationary case there are no problems with instabilities, and the Navier-Stokes equations can be solved by a linear finite-element or finite-differences method combined with a Picard-iteration for the nonlinear part.

3.8.3 Computational Fluid Dynamics (CFD)

Given the importance of solving the Navier-Stokes equation for engineering the numerical solution of these equations has become an important field of engineering called Computational Fluid Dynamics (CFD). For further details we thus refer to the special courses offered in CFD.

3.9 Solitons and the Korteveg-de Vries equation

As the final application of partial differential equations for this semester – quantum mechanics and the Schrödinger equation will be discussed in the summer semester – we will discuss the Korteveg-de Vries equations and solitons.

3.9.1 Solitons

John Scott Russell, a Scottish engineer working on boat design made a remarkable discovery in 1834:

> *I was observing the motion of a boat which was rapidly drawn along a narrow channel by a pair of horses, when the boat suddenly stopped - not so the mass of water in the channel which it had put in motion; it accumulated round the prow of the vessel in a state of violent agitation, then suddenly leaving it behind, rolled forward with great velocity, assuming the form of a large solitary elevation, a rounded, smooth and well-defined heap of water, which continued its course along the channel apparently without change of form or diminution of speed. I followed it on horseback, and overtook it still rolling on at a rate of some eight or nine miles an hour, preserving its original figure some thirty feet long and a foot to a foot and a half in height. Its height gradually diminished, and after a chase of one or two miles I lost it*

in the windings of the channel. Such, in the month of August 1834, was my first chance interview with that singular and beautiful phenomenon which I have called the Wave of Translation.

John Scott Russell's "wave of translation" is nowadays called a soliton and is a wave with special properties. It is a time-independent stationary solution of special non-linear wave equations, and remarkably, two solitions pass through each other without interacting.

Nowadays solitons are far from being just a mathematical curiosity but can be used to transport signal in specially designed glass fibers over long distances without a loss due to dispersion.

3.9.2 The Korteveg-de Vries equation

The Korteveg-de Vries (KdV) equation is famous for being the first equation found which shows soliton solutions. It is a nonlinear wave equation

$$\frac{\partial u(x,t)}{\partial t} + \epsilon u \frac{\partial u(x,t)}{\partial x} + \mu \frac{\partial^3 u(x,t)}{\partial x^3} = 0 \tag{3.71}$$

where the spreading of wave packets due to dispersion (from the third term) and the sharpening due to shock waves (from the non-linear second term) combine to lead to time-independent solitons for certain parameter values.

Let us first consider these two effects separately. First, looking at a linear wave equation with a higher order derivative

$$\frac{\partial u(x,t)}{\partial t} + c \frac{\partial u(x,t)}{\partial x} + \beta \frac{\partial^3 u(x,t)}{\partial x^3} = 0 \tag{3.72}$$

and solving it by the usual ansatz $u(x,t) = \exp(i(kx \pm \omega t)$ we find dispersion due to wave vector dependent velocities:

$$\omega = \pm ck \mp \beta k^3 \tag{3.73}$$

Any wave packet will thus spread over time.

Next let us look at the nonlinear term separately:

$$\frac{\partial u(x,t)}{\partial t} + \epsilon u \frac{\partial u(x,t)}{\partial x} = 0 \tag{3.74}$$

The amplitude dependent derivative causes taller waves to travel faster than smaller ones, thus passing them and piling up to a large shock wave, as can be seen in the Mathematica Notebook provided on the web page.

Balancing the dispersion caused by the third order derivative with the sharpening due to the nonlinear term we can obtain solitions!

3.9.3 Solving the KdV equation

The KdV equation can be solved analytically by making the ansatz $u(x,t) = f(x-ct)$. Inserting this ansatz we obtain an ordinary differential equation

$$\mu f^{(3)} + \epsilon f f' - c f' = 0, \tag{3.75}$$

which can be solved analytically in a long and cumbersome calculation, giving e.g. for $\mu = 1$ and $\epsilon = -6$:

$$u(x,t) = -\frac{c}{2} sech^2 \left[\frac{1}{2}\sqrt{c}\,(x - ct - x_0)\right] \tag{3.76}$$

In this course we are more interested in numerical solutions, and proceed to solve the KdV equation by a finite difference method

$$\begin{aligned} u(x_i, t+\Delta t) &= u(x_i, t-\Delta t) \qquad (3.77)\\ &-\frac{\epsilon}{3}\frac{\Delta t}{\Delta x}\left[u(x_{i+1},t) + u(x_i,t) + u(x_{i-1},t)\right]\left[u(x_{i+1},t) - u(x_{i-1},t)\right] \\ &-\mu\frac{\Delta t}{\Delta x^3}\left[u(x_{i+2},t) + 2u(x_{i+1},t) - 2u(x_{i-1},t) - u(x_{i-2},t)\right]. \end{aligned}$$

Since this integrator requires the wave at two previous time steps we start with an initial step of

$$\begin{aligned} u(x_i, t_0+\Delta t) &= u(x_i, t_0) \qquad (3.78)\\ &-\frac{\epsilon}{6}\frac{\Delta t}{\Delta x}\left[u(x_{i+1},t) + u(x_i,t) + u(x_{i-1},t)\right]\left[u(x_{i+1},t) - u(x_{i-1},t)\right] \\ &-\frac{\mu}{2}\frac{\Delta t}{\Delta x^3}\left[u(x_{i+2},t) + 2u(x_{i+1},t) - 2u(x_{i-1},t) - u(x_{i-2},t)\right] \end{aligned}$$

This integrator is stable for

$$\frac{\Delta t}{\Delta x}\left[|\varepsilon u| + 4\frac{|\mu|}{\Delta x^2}\right] \leq 1 \tag{3.79}$$

Note that as in the case of the heat equation, a progressive decrease of space steps or even of space and time steps by the same factor will lead to instabilities!

Using this integrator, also provided on the web page as a Mathematica Notebook you will be able to observe:

- The decay of a wave due to dispersion
- The creation of shock waves due to the nonlinearity
- The decay of a step into solitons
- The crossing of two solitons

Chapter 4

The classical N-body problem

4.1 Introduction

In this chapter we will discuss algorithms for classical N-body problems, whose length scales span many orders of magnitudes

- the universe ($\approx 10^{26}$m)
- galaxy clusters ($\approx 10^{24}$m)
- galaxies ($\approx 10^{21}$m)
- clusters of stars ($\approx 10^{18}$m)
- solar systems ($\approx 10^{13}$m)
- stellar dynamics ($\approx 10^{9}$m)
- climate modeling ($\approx 10^{6}$m)
- gases, liquids and plasmas in technical applications ($\approx 10^{-3} \ldots 10^{2}$m)

On smaller length scales quantum effects become important. We will deal with them later.

The classical N-body problem is defined by the following system of ordinary differential equations:

$$\begin{aligned} m_i \frac{d\vec{v}_i}{dt} &= \vec{F}_i = -\nabla_i V(\vec{x}_1, \ldots, \vec{x}_N) \\ \frac{d\vec{x}_i}{dt} &= \vec{v}_i, \end{aligned} \tag{4.1}$$

where m_i, $\vec{v}_i$ and $\vec{x}_i$ are the mass, velocity and position of the i-the particle.

The potential $V(\vec{x}_1, \ldots, \vec{x}_N)$ is often the sum of an external potential and a two-body interaction potential:

$$V(\vec{x}_1, \ldots, \vec{x}_N) = \sum_i V_{\text{ext}}(\vec{x}_i) + \sum_{i<j} U_{ij}(|\vec{x}_i - \vec{x}_j|) \tag{4.2}$$

The special form $U(|\vec{x}_i - \vec{x}_j|)$ of the two-body potential follows from translational and rotational symmetry.

4.2 Applications

There are many different forms of the potential U:

1. In astrophysical problems gravity is usually sufficient, except in dense plasmas, interiors of stars and close to black holes:
$$U_{ij}^{(\text{gravity})}(r) = -G\frac{m_i m_j}{r}. \tag{4.3}$$

2. The simplest model for non-ideal gases are hard spheres with radius a_i:
$$U_{ij}^{(\text{hard sphere})}(r) = \begin{cases} 0 & \text{for } r >= a_i + a_j \\ \infty & \text{for } r < a_i + a_j \end{cases} \tag{4.4}$$

3. Covalent crystals and liquids can be modeled by the Lennard-Jones potential
$$U_{ij}^{(\text{LJ})}(r) = 4\epsilon_{ij}\left[(\frac{\sigma}{r}^{12}) - (\frac{\sigma}{r})^6\right]. \tag{4.5}$$
The r^{-6}-term describes the correct asymptotic behavior of the covalent van der Waals forces. The r^{-12}-term models the hard core repulsion between atoms. The special form r^{-12} is chosen to allow for a fast and efficient calculation as square of the r^{-6} term. Parameters for liquid argon are $\epsilon = 1.65 \times 10^{-21}$J and $\sigma = 3.4 \times 10^{-10}$m.

4. In ionic crystals and molten salts the electrostatic forces are dominant:
$$U_{ij}^{(\text{ionic})}(r) = b_{ij}r^{-n} + e^2\frac{Z_i Z_j}{r}, \tag{4.6}$$
where Z_i and Z_j are the formal charges of the ions.

5. The simulation of large biomolecules such as proteins or even DNA is a big challenge. For non-bonded atoms often the 1-6-12 potential, a combination of Lennard-Jones and electrostatic potential is used:
$$U_{ij}^{(1-6-12)}(r) = e^2\frac{Z_i Z_j}{r} + 4\epsilon_{ij}\left[(\frac{\sigma}{r}^{12}) - (\frac{\sigma}{r})^6\right]. \tag{4.7}$$
For bonded atoms there are two ways to model the bonding. Either the distances between two atoms can be fixed, or the bonding can be described by a harmonic oscillator:
$$U_{ij}^{(\text{bond})}(r) = \frac{1}{2}K_{ij}(r - b_{ij})^2. \tag{4.8}$$
The modeling of fixed angles between chemical bonds (like in water molecules) is a slightly more complex problem. Again, either the angle can be fixed, or modeled by a harmonic oscillator in the angle θ. Note that the angle θ is determined by the location of three atoms, and that this is thus a *three-body-interaction*! Students who are interested in such biomolecules are referred to the research group of Prof. van Gunsteren in the chemistry department.

6. More complex potentials are used in the simulation of dense plasmas and of collisions of heavy atomic nuclei.

7. The Car-Parrinello method combines a classical simulation of the molecular dynamics of the motion of atomic nuclei with a quantum chemical ab-initio calculation of the forces due to electronic degrees of freedom. This gives more accurate forces than a Lennard-Jones potential but is possible only on rather small systems due to the large computational requirements. If you are interested in the Car-Parrinello method consider the research group of Prof. Parrinello in Lugano.

4.3 Solving the many-body problem

The classical many-body problem can be tackled with the same numerical methods that we used for the few-body problems, but we will encounter several additional difficulties, such as

- the question of boundary conditions
- measuring thermodynamic quantities such as pressure
- performing simulations at constant temperature or pressure instead of constant energy or volume
- reducing the scaling of the force calculation for long-range forces from $O(N^2)$ to $O(N \ln N)$
- overcoming the slowing down of simulations at phase transitions

4.4 Boundary conditions

Open boundary conditions are natural for simulations of solar systems or for collisions of galaxies, molecules or atomic nuclei. For simulations of crystals, liquids or gases on the other hand, effects from open boundaries are not desired, except for the investigation of surface effects. For these systems periodic boundary conditions are better. As we discussed earlier, they remove all boundary effects.

In the calculation of forces between two particle all periodic images of the simulation volume have to be taken into account. For short range forces, like a Lennard-Jones force, the "minimum image" is the method of choice. Here the distance between a particle and the nearest of all periodic images of a second particle is chosen for the calculation of the forces between the two particles.

For long range forces on the other hand (forces that as r^{-d} or slower) the minimum image method is not a good approximation because of large finite size effects. Then the forces caused by all the periodic images of the second particle have to be summed over. The electrostatic potential acting on a particle caused by other particles with charge q_i at sites $\vec{r}_i$ is

$$\Phi_p = \sum_{\vec{n}} \sum_i \frac{q_i}{|\vec{r}_{\vec{n}} - \vec{r}_i|}, \tag{4.9}$$

where $\vec{n}$ is an integer vector denoting the periodic translations of the root cell and $\vec{r}_{\vec{n}}$ is the position of the particle in the corresponding image of the root cell.

This direct summation converges very slowly. It can be calculated faster by the Ewald summation technique[1], which replaces the sum by two faster converging sums:

$$\begin{aligned}\Phi_p &= \sum_{\vec{n}}\sum_i q_i \frac{\mathrm{erfc}(\alpha|\vec{r}_{\vec{n}}-\vec{r}_i|)}{|\vec{r}_{\vec{n}}-\vec{r}_i|} + \\ &\quad + \frac{1}{\pi L}\sum_i \sum_{\vec{h}\neq 0} q_i \exp\left(\frac{-\pi|h|^2}{\alpha L^2}\right)\cos\left(\frac{2\pi}{L}\vec{h}\cdot(\vec{r}_o-\vec{r}_i)\right). \end{aligned} \tag{4.10}$$

In this sum the $\vec{h}$ are integer reciprocal lattice vectors. The parameter α is arbitrary and can be chosen to optimize convergence.

Still the summation is time-consuming. Typically one tabulates the differences between Ewald sums and minimum image values on a grid laid over the simulation cell and interpolates for distances between the grid points. For details we refer to the detailed discussion in M.J. Sangster and M. Dixon, Adv. in Physics **25**, 247 (1976).

4.5 Molecular dynamics simulations of gases, liquids and crystals

4.5.1 Ergodicity, initial conditions and equilibration

In scattering problems or in the simulation of cosmological evolution the initial conditions are usually given. The simulation then follows the time evolution of these initial conditions. In molecular dynamics simulations on the other hand one is interested in thermodynamic averages $\langle A\rangle$. In an *ergodic* system the phase space average is equivalent to the time average:

$$\langle A\rangle := \frac{\int A(\Gamma)P[\Gamma]d\Gamma}{\int P[\Gamma]d\Gamma} = \lim_{\tau\to\infty}\frac{1}{\tau}\int_0^\tau A(t)dt. \tag{4.11}$$

Initial conditions are best chosen as a regular crystal lattice. The velocities are picked randomly for each component, according to a Maxwell distribution

$$P[v_\alpha] \propto \exp\left(\frac{-mv_\alpha^2}{2k_BT}\right). \tag{4.12}$$

Finally the velocities are all rescaled by a constant factor to obtain the desired total energy.

An important issue is that the system has to be equilibrated (thermalized) for some time before thermal equilibrium is reached and measurements can be started. This thermalization time is best determined by observing time series of physical observables, such as the kinetic energy (temperature) or other quantities of interest.

[1]P.P. Ewald, Ann. Physik **64**, 253 (1921).

4.5.2 Measurements

A simple measurement is the self-diffusion constant D. In a liquid or gaseous system it can be determined from the time dependence of the positions:

$$\Delta^2(t) = \frac{1}{N}\sum_{i=1}^{N}[\vec{r}_i(t) - \vec{r}_i(0)]^2 = 2dDt + \Delta_0^2 \tag{4.13}$$

In a crystal the atoms remain at the same location in the lattice and thus $D = 0$. A measurement of D is one way to observe melting of a crystal.

Another quantity that is easy to measure is the mean kinetic energy

$$\langle E_k \rangle = \frac{1}{2}\langle \sum_{i=1}^{N} m_i \vec{v}_i^2 \rangle. \tag{4.14}$$

$\langle E_k \rangle$ is proportional to the mean temperature

$$\langle E_k \rangle = \frac{G}{2} k_B T, \tag{4.15}$$

where $G = d(N-1) \approx dN$ is the number of degrees of freedom.

In a system with fixed boundaries the particles are reflected at the boundaries. The pressure P is just the force per area acting on the boundary walls of the system In the case of periodic boundary conditions there are no walls. The pressure P can then be measured using the following equation, derived from the virial theorem:

$$P = \frac{N k_B T}{V} + \frac{1}{dV}\sum_{i<j} \vec{r}_{ij} \cdot \vec{F}_{ij}(t), \tag{4.16}$$

where $\vec{F}_{ij}$ denotes the force between particles i and j and $\vec{r}_{ij}$ is their distance.

The first term of equation (4.16) is the kinetic pressure, due to the kinetic energy of the particles. This term alone gives the ideal gas law. The second term is the pressure (force per area) due to the interaction forces.

More information can usually be extracted from the pair correlation function

$$g(\vec{r}) = \frac{1}{\rho(N-1)}\left\langle \sum_{i \neq j} \delta(\vec{r} + \vec{r}_i - \vec{r}_j) \right\rangle \tag{4.17}$$

or its Fourier transform, the static structure factor $S(\vec{k})$

$$g(\vec{r}) - 1 = \frac{1}{(2\pi)^d \rho}\int [S(\vec{k}) - 1] \exp(i\vec{k}\cdot\vec{r}) d\vec{k} \tag{4.18}$$

$$S(\vec{k}) - 1 = \rho \int [g(\vec{r}) - 1] \exp(-i\vec{k}\cdot\vec{r}) d\vec{r} \tag{4.19}$$

If the angular dependence is of no interest, a radial pair correlation function

$$g(r) = \frac{1}{4\pi}\int g(\vec{r}) \sin\theta d\theta d\phi \tag{4.20}$$

and corresponding structure factor

$$S(k) = 4\pi\rho \int_0^\infty \frac{\sin kr}{kr} [g(r) - 1] r^2 dr \tag{4.21}$$

can be used instead.

This structure factor can be measured in X-ray or neutron scattering experiments. In a perfect crystal the structure factor shows sharp δ-function like Bragg peaks and a periodic long range structure in $g(\vec{r})$. Liquids still show broad maxima at distances of nearest neighbors, second nearest neighbors, etc., but these features decay rapidly with distance.

The specific heat at constant volume c_V can in principle be calculated as a temperature derivative of the internal energy. Since such numerical derivatives are numerically unstable the preferred method is a calculation from the energy fluctuations

$$c_v = \frac{\langle E^2 \rangle - \langle E \rangle^2}{k_B T^2}. \tag{4.22}$$

4.5.3 Simulations at constant energy

The equations of motion of a disspiationless system conserve the total energy and the simulation is thus done in the microcanonical ensemble. Discretization of the time evolution however introduces errors in the energy conservation, and as a consequence the total energy will slowly change over time. To remain in the microcanonical ensemble energy corrections are necessary from time to time. These are best done by a rescaling of all the velocities with a constant factor. The equations are easy to derive and will not be listed here.

4.5.4 Constant temperature

The canonical ensemble at constant temperature is usually of greater relevance than the microcanonical ensemble at constant energy. The crudest, ad-hoc method for obtaining constant temperature is a rescaling like we discussed for constant energy. This time however we want rescale the velocities to achieve a constant kinetic energy and thus, by equation (4.15) constant temperature. Again the equations can easily be derived.

A better method is the Nosé-Hoover thermostat. In this algorithm the system is coupled reversibly to a heat bath by a friction term η:

$$\begin{aligned} m_i \frac{d\vec{v}_i}{dt} &= \vec{F}_i - \eta \vec{v}_i \\ \frac{d\vec{r}_i}{dt} &= \vec{v}_i \qquad (4.23) \\ & \qquad (4.24) \end{aligned}$$

The friction term η is chosen such that constant temperature is achieved on average. We want this term to heat up the system if the temperature is too low and to cool it down if the temperature is too high. One way of doing this is by setting

$$\frac{d\eta}{dt} = \frac{1}{m_s} \left(E_k - \frac{1}{2} G k_B T \right), \tag{4.25}$$

where m_s is the coupling constant to the heat bath.

4.5.5 Constant pressure

Until now we always worked at fixed volume. To perform simulations at constant pressure we need to allow the volume to change. This can be done by rescaling the coordinates with the linear size L of the system:

$$\vec{r} = L\vec{x}. \tag{4.26}$$

The volume of the system is denoted by $\Omega = L^D$. We extend the Lagrangian by including an external pressure P_0 and an inertia M for pressure changes (e.g. the mass of a piston):

$$\mathcal{L} = \sum_{i=1}^{N} \frac{m_i}{2} L^2 \left(\frac{d\vec{x}_i}{dt}\right)^2 - \sum_{i<j} V(L(\vec{x}_i - \vec{x}_j)) + \frac{M}{2}\left(\frac{d\Omega}{dt}\right)^2 + P_0\Omega \tag{4.27}$$

The Euler equations applied to above Lagrangian give the equations of motion:

$$\begin{aligned} \frac{d^2\vec{x}_i}{dt^2} &= \frac{1}{m_i L}\vec{F}_i - \frac{2}{D\Omega}\frac{d\Omega}{dt}\frac{d\vec{x}_i}{dt} \\ \frac{d^2\Omega}{dt^2} &= \frac{P - P_0}{M}, \end{aligned} \tag{4.28}$$

where P turns out to be just the pressure defined in equation (4.16). These equations of motion are integrated with generalizations of the Verlet algorithm.

Generalizations of this algorithm allow changes not only of the total volume but also of the shape of the simulation volume.

4.6 Scaling with system size

The time intensive part of a classical N-body simulation is the calculation of the forces. The updating of positions and velocities according to the forces is rather fast and scales linearly with the number of particles N.

For short range forces the number of particles within the interaction range is limited, and the calculation of the forces, while it might still be a formidable task, scales with $O(N)$ and thus poses no big problems.

Rapidly decaying potentials, like the Lennard-Jones potential can be cut off at a distance r_c. The error thus introduced into the estimates for quantities like the pressure can be estimated from equations (4.16) using equation (4.17) as:

$$\Delta P = -\frac{2\pi\rho^2}{3}\int_{r_c}^{\infty} \frac{\partial V}{\partial r} g(r) r^3 dr \tag{4.29}$$

where a common two-body potential $V(r)$ between all particle pairs was assumed. If $V(R)$ decays faster than r^{-3} (in general r^{-d}, where d is the dimensionality) this correction becomes small as r_c is increased.

Long range forces, like Coulomb forces or gravity, on the other hand, pose a big problem. No finite cut-off may be introduced without incurring substantial errors.

Each particle asserts a force onto every other particle, thus requiring $(N-1)N/2 \sim O(N^2)$ force calculations. This is prohibitive for large scale simulations. In numerical simulations there is a solution to this problem. Due to discrete time steps Δt we cannot avoid making errors in the time integration. Thus we can live with a small error in the force calculations and use one of a number of algorithms that, while introducing small controllable errors in the forces, need only $O(N \log N)$ computations.

4.6.1 The Particle-Mesh (PM) algorithm

The Particle-Mesh (PM) algorithm maps the force calculation to the solution of a Poisson equation which can be done in a time proportional to $O(N \log N)$. It works as follows:

1. A regular mesh with $M \sim N$ mesh points is introduced in the simulation volume
2. The masses of the particles are assigned – in a clever way – to nearby mesh points.
3. The potential equation (often a Poisson equation) is solved on the mesh using a fast solver in $O(M \log M) \sim O(N \log N)$ steps.
4. The potential at the position of each particle is interpolated – again in a clever way – from the potential at the nearby mesh points and the force upon the particle calculated from the gradient of the potential.

The charge assignment and potential interpolation are the tricky parts. They should be done such that errors are minimized. The standard textbook "Computer Simulations Using Particles" by R.W. Hockney and J.W. Eastwood discusses the PM method in great detail.

We want to fulfill at least the following conditions

1. At large particle separations the errors should become negligible
2. The charge assigned to the mesh points and the forces interpolated from mesh points should vay smoothly as the particle position changes.
3. Total momentum should be conserved, i.e. the force $\vec{F}_{ij}$ acting on particle i from particle j should fulfill $\vec{F}_{ij} = -\vec{F}_{ji}$.

The simplest scheme is the NGP scheme (neasrest grid point), where the full particle mass is assigned to the nearest grid point and the force is also evaluated at the nearest grid point. More elaborate schemes like the CIC (cloud in cell) scheme assign the charge to the 2^d nearest grid points and also interpolate the forces from these grid points. The algorithm becomes more accurate but also more complex as more points are used. For detailed discussions read the book by Hockney and Eastwood.

In periodic systems and for forces which have a Greens function $g(\vec{r})$ (e.g. solutions of a Poisson equation) one of the best methods is the fast Fourier method, which we will describe for the case of gravity, where the potential can be calculated as

$$\Phi(\vec{r}) = \int d^3\vec{r}' \rho(\vec{r}') g(\vec{r} - \vec{r}'), \tag{4.30}$$

where $\rho(\vec{r})$ is the charge distribution and the Greens function is

$$g(\vec{r}) = \frac{G}{||\vec{r}||} \tag{4.31}$$

in three space dimension. The convolution in equation (4.30) is best performed by Fourier transforming the equation, which reduces it to a multiplication

$$\hat{\Phi}(\vec{k}) = \hat{\rho}(\vec{k})\hat{g}(\vec{k}) \tag{4.32}$$

On the finite mesh of the PM method, the discrete charge distribution is Fourier transformed in $O(M \log M)$ steps using the Fast Fourier Transform (FFT) algorithm. The Fourier transform of the Greens function in equation (4.31) is

$$\hat{g}(\vec{k}) = \frac{G}{||\vec{k}||^2}. \tag{4.33}$$

Using this equation gives the "poor man's Poisson solver". A suitably modified Greens function, such as

$$\hat{g}(\vec{k}) \propto \frac{1}{\sin^2(k_x L/2) + \sin^2(k_y L/2) + \sin^2(k_z L/2)}, \tag{4.34}$$

where L is the linear dimension of the simulation volume can reduce the discretization errors caused by the finite mesh. In contrast to equation (4.33) this form differentiable also at the Brillouin zone boundary, e.g. when $k_x = \pm\pi/L$ or $k_y = \pm\pi/L$ or $k_z = \pm\pi/L$. Before writing any program yourself we recommend that you study the textbooks and literature in detail as there are many subtle issues you have to take care of.

The PM algorithm is very efficient but has problems with

- non-uniform particle distributions, such as clustering of stars and galaxies.
- strong correlations effects between particles. Bound states, such as binary stars, will never be found in PM simulations of galaxies.
- complex geometries.

The first two of these problems can be solved by the P^{3}M and AP^{3}M algorithms

4.6.2 The P^{3}M and AP^{3}M algorithms

The PM method is good for forces due to far away particles but bad for short ranges. The P^{3}M method solves this problem by splitting the force $\vec{F}$ into a long range force $\vec{F}_l$ and a short range force $\vec{F}_s$:

$$\vec{F} = \vec{F}_l + \vec{F}_s \tag{4.35}$$

The long range force $\vec{F}_l$ is chosen to be small and smoothly varying for short distances. It can be efficiently computed using the particle-mesh (PM) method. The short range force $\vec{F}_s$ has a finite interaction radius R is calculated exactly, summing up the particle-particle forces. Thus the name particle-particle/particle mesh (P^{3}M) algorithm.

For nearly uniform particle distributions the number of particles within the range of $\vec{F}_s$ is small and independent of N. The P^{3}M algorithm then scales as $O(N) + O(M \log M)$ with $M \sim N$.

Attractive long range forces, like gravity, tend to clump particles and lead to extremely non-uniform particle distribution. Just consider solar systems, star clusters, galaxies and galaxy clusters to see this effect. Let us consider what happens to the P^{3}M method in this case. With a mesh of $N \approx M$ points it will happen that almost all particles (e.g. a galaxy) clump within the range R of the short range force $\vec{F}_s$. Then the PP part scales like $O(N^2)$. Alternatively we can increase the number of mesh points M to about $M \gg N$, which again is non-optimal.

The solution to this problem refining the mesh in the regions of space with a high particle density. In a simulation of a collision between two galaxies we will use a fine mesh at the location of the galaxies and a coarse mesh in the rest of the simulation space. The adaptive P^{3}M method (AP^{3}M) automatically refines the mesh in regions of space with high particle densities and is often used, besides tree codes, for cosmological simulations.

4.6.3 The tree codes

Another approach to speeding up the force calculation is by collecting clusters of far away particles into effective pseudoparticles. The mass of these pseudo particles is the total mass of all particles in the cluster they represent. To keep track of these clusters a tree is constructed. Details of this method are explained very well in the book "Many-Body Tree Methods in Physics" by Susanne Pfalzner and Paul Gibbon.

4.6.4 The multipole expansion

In more or less homogeneous systems another algorithm can be used which at first sight scales like $O(N)$ The "Fast Multipole Method" (FMM) calculates a high order multipole expansion for the potential due to the particles and uses this potential to calculate the forces. The calculation of the high order multipole moments is a big (programming) task, but scales only like $O(N)$.

Is this the optimal method? No! There are two reasons. First of all, while the calculation of the multipole moments takes only $O(N)$ time we need to go to about $O(\log N)$-th order to achieve good accuracy, and the overall scaling is thus also $O(N \log N)$. Secondly, the calculation of the multipole moments is a computationally intensive task and the prefactor of the $N \log N$ term much larger than in tree codes.

The multipole method is still useful in combination with tree codes. Modern tree codes calculate not only the total mass of a cluster, but also higher order multipole moments, up to the hexadecapole. This improves the accuracy and efficiency of tree codes.

4.7 Phase transitions

In the molecular dynamics simulations of a Lennard-Jones liquid in the exercises we can see the first example of a phase transition: a first order phase transition between a crystal and a solid. Structural phase transitions in continuous systems are usually of first order, and second order transition occur only at special points, such as the critical point of a fluid. We will discuss second order phase transitions in more detail in the later chapter on magnetic simulations and will focus here on the first order melting transition.

In first order phase transitions both phases (e.g. ice and water) can coexist at the same time. There are several characteristic features of first order phase transitions that can be used to distinguish them from second order ones. One such feature is the latent heat for melting and evaporation. If the internal energy is increased, e.g. by a a heat source, the temperature first increases until the phase transition. Then it stays constant as more and more of the crystal melts. The temperature will rise again only once enough energy is added to melt the whole crystal. Alternatively this can be seen as a jump at the transition temperature of the internal energy as a function of temperature. Similarly, at constant pressure a volume change can be observed at the melting transition. Another indication is a jump in the self diffusion constant at T_c.

A more direct observation is the measurement of a quantity like the structure factor in different regions of the simulation volume. At first order phase transitions regions of both phases (e.g. crystal and liquid or liquid and gas) can be observed at the same time. In second order phase transitions (like in crystal structure changes from tetragonal to orthorhombic), on the other hand a smooth change as a function of temperature is observed, and the whole system is always either in one phase or in the other.

When simulating first order phase transitions one encounters a problem which is actually a well-known phenomenon. To trigger the phase transition a domain of the new phase has to be formed. As this formation of the domain can cost energy proportional to its boundary the formation of such new domains can be suppressed, resulting in undercooled or overheated liquids. The huge time scales for melting (just watch an ice cube melt!) are a big problem for molecular dynamics simulations of first order phase transitions. Later we will learn how Monte Carlo simulations can be used to introduce a faster dynamics, speeding up the simulation of phase transitions.

4.8 From fluid dynamics to molecular dynamics

Depending on strength and type of interaction different algorithms are used for the simulation of classical systems.

1. Ideal or nearly ideal gases with weak interaction can be modeled by the Navier-Stokes equations.

2. If the forces are stronger, the Navier-Stokes equations are no longer appropriate. In that case particle-in-cell (PIC) algorithms can be used:

    - Like in the finite element method the simulation volume is split into cells. Here the next step is not the solution of a partial differential equation on this mesh, but instead the fluid volume in each cell is replaced by a pseudo-particle. These pseudo-particles, which often correspond to millions of real fluid particles, carry the total mass, charge and momentum of the fluid cell.

   - The pseudo-particle are then propagated using molecular dynamics.
   - Finally, the new mass, charge and momentum densities on the mesh are interpolated from the new positions of the pseudoparticles.

3. If interactions or correlations are even stronger, each particle has to be simulated explicitly, using the methods discussed in this chapter.

4. In astrophysical simulations there are huge differences in density and length scales - from interstellar gases to neutron stars or even black holes. For these simulations *hybrid methods* are needed. Parts of the system (e.g. interstellar gases and dark matter) are treated as fluids and simulated using fluid dynamics. Other parts (e.g. galaxies and star clusters) are simulated as particles. The border line between fluid and particle treatment is fluid and determined only by the fact that currently not more than 10^8 particles can be treated.

4.9 Warning

Tree codes and the (A)P^3M methods accept a small error in exchange for a large speedup. However, when simulating for several million time steps these errors will add up. Are the results reliable? This is still an open question.

Chapter 5

Integration methods

In thermodynamics, as in many other fields of physics, often very high dimensional integrals have to be evaluated. Even in a classical N-body simulation the phase space has dimension $6N$, as there are three coordinates each for the location and position of each particle. In a quantum mechanical problem of N particles the phase space is even exponentially large as a function of N. We will now review what we learned last semester about integration methods and Monte Carlo integrators.

5.1 Standard integration methods

A Riemannian integral $f(x)$ over an interval $[a, b]$ can be evaluated by replacing it by a finite sum:

$$\int_a^b f(x)dx = \sum_{i=1}^{N} f(a + i\Delta x)\Delta x + \mathrm{O}(\Delta x^2), \tag{5.1}$$

where $\Delta x = (a - b)/N$. The discretization error decreases as $1/N$ for this simple formula. Better approximations are the trapezoidal rule

$$\int_a^b f(x)dx = \Delta x \left[\frac{1}{2}f(a) + \sum_{i=1}^{N-1} f(a + i\Delta x) + \frac{1}{2}f(b)\right] + \mathrm{O}(\Delta x^2), \tag{5.2}$$

or the Simpson rule

$$\int_a^b f(x)dx = \frac{\Delta x}{3}\left[f(a) + \sum_{i=1}^{N/2} 4f(a + (2i-1)\Delta x) + \sum_{i=1}^{N/2-1} 2f(a + 2i\Delta x) + f(b)\right] + \mathrm{O}(\Delta x^4), \tag{5.3}$$

which scales like N^{-4}.

For more elaborate schemes like the Romberg method or Gaussian integration we refer to textbooks.

In higher dimensions the convergence is much slower though. With N points in d dimensions the linear distance between two points scales only as $N^{-1/d}$. Thus the Simpson rule in d dimensions converges only as $N^{-4/d}$, which is very slow for large d. The solution are Monte Carlo integrators.

5.2 Monte Carlo integrators

With randomly chosen points the convergence does not depend on dimensionality. Using N randomly chosen points $\mathbf{x}_i$ the integral can be approximated by

$$\frac{1}{\Omega}\int f(\mathbf{x})d\mathbf{x} \approx \overline{f} := \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_i), \tag{5.4}$$

where $\Omega := \int d\mathbf{x}$ is the integration volume. As we saw in the previous chapter the errors of such a Monte Carlo estimate the errors scale as $N^{-1/2}$. In $d \geq 9$ dimensions Monte Carlo methods are thus preferable to a Simpson rule.

5.2.1 Importance Sampling

This simple Monte Carlo integration is however not the ideal method. The reason is the variance of the function

$$\mathrm{Var} f = \Omega^{-1}\int f(\mathbf{x})^2 d\mathbf{x} - \left[\Omega^{-1}\int f(\mathbf{x})d\mathbf{x}\right]^2 \approx \frac{N}{N-1}(\overline{f^2} - \overline{f}^2). \tag{5.5}$$

The error of the Monte Carlo simulation is

$$\Delta = \sqrt{\frac{\mathrm{Var} f}{N}} \approx \sqrt{\frac{\overline{f^2} - \overline{f}^2}{N-1}}. \tag{5.6}$$

In phase space integrals the function is often strongly peaked in a small region of phase space and has a large variance. The solution to this problem is "importance sampling", where the points $\mathbf{x}_i$ are chosen not uniformly but according to a probability distribution $p(\mathbf{x})$ with

$$\int p(\mathbf{x})d\mathbf{x} = 1. \tag{5.7}$$

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

$$\langle f\rangle = \Omega^{-1}\int A(\mathbf{x})d\mathbf{x} = \Omega^{-1}\int \frac{f(\mathbf{x})}{p(\mathbf{x})}p(\mathbf{x})d\mathbf{x} \approx \frac{1}{N}\sum_{i=1}^{N}\frac{f(\mathbf{x}_i)}{p(\mathbf{x}_i)} \tag{5.8}$$

and the error is

$$\Delta = \sqrt{\frac{\mathrm{Var} f/p}{N}}. \tag{5.9}$$

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

As an example, the function $f(x) = \exp(-x^2)$ is much better integrated using exponentially distributed random numbers with $p(x) = \exp(-\lambda x)$ instead of uniformly distributed random numbers.

A natural choice for the weighting function p is often given in the case of phase space integrals or sums, where an observable A is averaged over all configurations $\mathbf{x}$ in phase space where the probability of a configuration is $p(\mathbf{x})$. The phase space average $\langle A\rangle$ is:

$$\langle A\rangle = \frac{\int A(\mathbf{x})p(x)d\mathbf{x}}{\int p(x)d\mathbf{x}}. \tag{5.10}$$

5.3 Pseudo random numbers

The most important ingredient for a Monte Carlo calculation is a source of random numbers. The problem is: how can a deterministic computer calculate true random numbers. One possible source of random numbers is to go to a casino (not necessarily in Monte Carlo) and obtain random numbers from the roulette wheel.

Since this is not a useful suggestion the best remaining solution is to calculate pseudo random numbers using a numerical algorithm. Despite being deterministic these pseudo random number generators can produce sequences of numbers that look random if one does not know the underlying algorithm. As long as they are sufficiently random (you might already see a problem appearing here), these pseudo random numbers can be used instead of true random numbers.

5.3.1 Uniformly distributed random numbers

A popular type of random number generator producing uniformly distributed is the linear congruential generator (LCG)

$$x_n = (a x_{n-1} + c) \bmod m, \tag{5.11}$$

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

Modern generators are usually based on lagged Fibonacci methods, such as the generator

$$x_n = x_{n-607} + x_{n-253} \bmod m \tag{5.12}$$

The first 607 numbers need to be produced by another generator, e.g. an LCG generator. Instead of the shifts $(607, 253)$ other good choices can be $(2281, 1252)$,$(9689, 5502)$ or $(44497, 23463)$. By calculating the next number from more than one previous numbers these generator can have extremely long periods.

One of the most recent generators is the Mersenne Twister, a combination of a lagged Fibonacci generator with a bit twister, shuffling around the bits of the random numbers to improve the quality of the generator.

Instead of coding a pseudo random number generator yourself, use one of the libraries such as the SPRNG library in Fortran or C or the Boost random number library in C++.

5.3.2 Testing pseudo random numbers

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

- The period has to be longer than the number of pseudo random numbers required.

- The numbers have to be uniformly distributed. This can be tested by a χ^2 or a Kolmogorov-Smirnov test.
- Successive number should not be correlated. This can be tested by a χ^2 test or by a simple graphical test: fill a square with successive points at the coordinates (x_{2i-1}, x_{2i}) and look for structures.

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

5.3.3 Non-uniformly distributed random numbers

Non-uniformly distributed random numbers can be obtained from uniformly distributed random numbers in the following way. Consider the probability that a random number y, distributed with a distribution f is less than x. This probability is just the integral

$$P_f[y < x] = \int_{-\infty}^{x} f(y)dy =: F(x). \tag{5.13}$$

But this is also just the probability that a uniformly distributed random number u is less than $F(x)$:

$$P_f[y < x] = F(x) = P_u[u < F(x)] \tag{5.14}$$

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

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

$$x_{exp} = -\frac{1}{\lambda}\ln(1-u). \tag{5.15}$$

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

$$\begin{aligned} n_1 &= \sqrt{-2\ln(1-u_1)}\cos 2\pi u_2 \qquad (5.16)\\ n_2 &= \sqrt{-2\ln(1-u_1)}\sin 2\pi u_2. \end{aligned}$$

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

5.4 Markov chains and the Metropolis algorithm

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

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

$$\mathbf{x}_0 \to \mathbf{x}_1 \to \mathbf{x}_2 \to \ldots \to \mathbf{x}_n \to \mathbf{x}_{n+1} \to \ldots \tag{5.17}$$

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

$$\sum_{\mathbf{y}} W_{\mathbf{xy}} = 1 \tag{5.18}$$

A consequence is that the Markov process conserves the total probability. Another consequence is that the largest eigenvalue of the transition matrix W is 1 and the corresponding eigenvector with only positive entries is the equilibrium distribution which is reached after a large number of Markov steps.

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

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

2. **Detailed balance:** The probability distribution $p_{\mathbf{x}}^{(n)}$ changes at each step of the Markov process:
$$\sum_{\mathbf{x}} p_{\mathbf{x}}^{(n)} W_{\mathbf{xy}} = p_{\mathbf{y}}^{(n+1)}. \tag{5.19}$$
but converges to the equilibrium distribution $p_{\mathbf{x}}$. This equilibrium distribution $p_{\mathbf{x}}$ is an eigenvector with left eigenvalue 1 and the equilibrium condition
$$\sum_{\mathbf{x}} p_{\mathbf{x}} W_{\mathbf{xy}} = p_{\mathbf{y}} \tag{5.20}$$
must be fulfilled. It is easy to see that the detailed balance condition
$$\frac{W_{\mathbf{xy}}}{W_{\mathbf{yx}}} = \frac{p_{\mathbf{y}}}{p_{\mathbf{x}}} \tag{5.21}$$
is sufficient.

The simplest Monte Carlo algorithm is the Metropolis algorithm:

- Starting with a point $\mathbf{x}_i$ choose randomly one of a fixed number N of changes $\Delta\mathbf{x}$, and propose a new point $\mathbf{x}' = \mathbf{x}_i + \Delta\mathbf{x}$.

- Calculate the ratio os the probabilities $P = p_{\mathbf{x}'}/p_{\mathbf{x}_i}$.
- If $P > 1$ the next point is $\mathbf{x}_{i+1} = \mathbf{x}'$
- If $P < 1$ then $\mathbf{x}_{i+1} = \mathbf{x}'$ with probability P, otherwise $\mathbf{x}_{i+1} = \mathbf{x}_i$. We do that by drawing a random number r uniformly distributed in the interval $[0, 1[$ and set $\mathbf{x}_{i+1} = \mathbf{x}'$ if $r < P$.
- Measure the quantity A at the new point $\mathbf{x}_{i+1}$.

The algorithm is ergodic if one ensures that the N possible random changes allow all points in the integration domain to be reached in a finite number of steps. If additionally for each change $\Delta\mathbf{x}$ there is also an inverse change $-\Delta\mathbf{x}$ we also fulfill detailed balance:

$$\frac{W_{ij}}{W_{ji}} = \frac{\frac{1}{N}\min(1, p(j)/p(i))}{\frac{1}{N}\min(1, p(i)/p(j))} = \frac{p(j)}{p(i)}. \tag{5.22}$$

As an example let us consider summation over integers i. We choose $N = 2$ possible changes $\Delta i{=}\pm 1$ and fulfill both ergodicity and detailed balance as long as $p(i)$ is nonzero only over a finite contiguous subset of the integers.

To integrate a one-dimensional function we take the limit $N \to \infty$ and pick any change $\delta \in [-\Delta, \Delta]$ with equal probability. The detailed balance equation (5.22) is only modified in minor ways:

$$\frac{W_{ij}}{W_{ji}} = \frac{\frac{d\delta}{2\Delta}\min(1, p(j)/p(i))}{\frac{d\delta}{2\Delta}\min(1, p(i)/p(j))} = \frac{p(j)}{p(i)}. \tag{5.23}$$

Again, as long as $p(x)$ is nonzero only on a finite interval detailed balance and ergodicity are fulfilled.

5.5 Autocorrelations, equilibration and Monte Carlo error estimates

5.5.1 Autocorrelation effects

In the determination of statistical errors of the Monte Carlo estimates we have to take into account correlations between successive points $\mathbf{x}_i$ in the Markov chain. These correlations between configurations manifest themselves in correlations between the measurements of a quantity A measured in the Monte Carlo process. Denote by $A(t)$ the measurement of the observable A evaluated at the t-th Monte Carlo point $\mathbf{x}_t$. The autocorrelations decay exponentially for large time differences Δ:

$$\langle A_t A_{t+\Delta}\rangle - \langle A\rangle^2 \propto \exp(-\Delta/\tau_A^{(exp)}) \tag{5.24}$$

Note that the autocorrelation time τ_A depends on the quantity A.

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

$$\tau_A^{(int)} = \frac{\sum_{\Delta=1}^{\infty}\left(\langle A_t A_{t+\Delta}\rangle - \langle A\rangle^2\right)}{\langle A^2\rangle - \langle A\rangle^2} \tag{5.25}$$

As usual the expectation value of the quantity A can be estimated by the mean $\overline{A}$, using equation (5.4). The error estimate [equation (5.6)] has to be modified. The error estimate $(\Delta A)^2$ is the expectation value of the squared difference between sample average and expectation value:

$$\begin{aligned}
(\Delta A)^2 &= \langle(\overline{A} - \langle A\rangle)^2\rangle = \langle\left(\frac{1}{N}\sum_{t=1}^{N} A(t) - \langle A\rangle\right)^2\rangle \\
&= \langle\frac{1}{N^2}\sum_{i=1}^{N}\left(A(t)^2 - \langle A\rangle^2\,\rangle\right) + \frac{2}{N^2}\sum_{t=1}^{N}\sum_{\Delta=1}^{N-t}\left(\langle A(t)A(t+\Delta)\rangle - \langle < A >^2\rangle\right) \\
&\approx \frac{1}{N}\mathrm{Var}A\;(1 + 2\tau_A^{(int)}) \\
&\approx \frac{1}{N-1}\langle\overline{A^2} - \overline{A}^2\rangle(1 + 2\tau_A^{(int)})
\end{aligned} \tag{5.26}$$

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

In many Monte Carlo simulations the error analysis is unfortunately not done accurately. Thus we wish to discuss this topic here and in the exercises.

5.5.2 The binning analysis

The binning analysis is a reliable way to estimate the integrated autocorrelation times. Starting from the original series of measurements $A_i^{(0)}$ with $i = 1, \ldots, N$ we iteratively create "binned" series by averaging over to consecutive entries:

$$A_i^{(l)} := \frac{1}{2}\left(A_{2i-1}^{(l-1)} + A_{2i}^{(l-1)}\right), \qquad i = 1, \ldots, N_l \equiv N/2^l. \tag{5.27}$$

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

The errors $\Delta A^{(l)}$, estimated incorrectly using equation (5.6)

$$\Delta A^{(l)} = \sqrt{\frac{\mathrm{Var}A^{(l)}}{N_l - 1}} \approx \frac{1}{N_l}\sqrt{\sum_{i=1}^{N_l}\left(A_i^{(l)} - \overline{A^{(l)}}\right)^2} \tag{5.28}$$

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

$$\Delta A = \lim_{l\to\infty} \Delta A^{(l)}. \tag{5.29}$$

This binning analysis gives a reliable recipe for estimating errors and autocorrelation times. One has to calculate the error estimates for different bin sizes l and check if they converge to a limiting value. If convergence is observed the limit ΔA is a reliable error estimate, and $\tau_A^{(int)}$ can be obtained from equation (5.26) as

$$\tau_A^{(int)} = \frac{1}{2}\left[\left(\frac{\Delta A}{\Delta A^{(0)}}\right)^2 - 1\right] \tag{5.30}$$

If however no convergence of the $\Delta A^{(l)}$ is observed we know that $\tau_A^{(int)}$ is longer than the simulation time and we have to perform *much* longer simulations to obtain reliable error estimates.

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

For the projects we will provide you with a simple observable class that implements this binning analysis.

5.5.3 Jackknife analysis

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

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

We could now evaluate the complex quantity U in each of the M bins and obtain an error estimate from the variance of these estimates. As each of the bins contains only a rather small number of measurements N/M the statistics will not be good. The jackknife procedure instead works with $M+1$ evaluations of U. U_0 is the estimate using all bins, and U_i for $i = 1, \dots M$ is the value when all bins *except* the i-th bin are used. That way we always work with a large data set and obtain good statistics.

The resulting estimate for U will be:

$$U = U_0 - (M-1)(\overline{U} - U_0) \tag{5.31}$$

with a statistical error

$$\Delta U = \sqrt{M-1}\left(\frac{1}{M}\sum_{i=1}^{M}(U_i)^2 - (\overline{U})^2\right)^{1/2}, \tag{5.32}$$

where

$$\overline{U} = \frac{1}{M}\sum_{i=1}^{M} U_i, \tag{5.33}$$

5.5.4 Equilibration

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

$$\tau_A^{(eq)} = \frac{\sum_{\Delta=1}^{\infty}(\langle A_0 A_\Delta \rangle - \langle A \rangle^2)}{\langle A_0 \rangle \langle A \rangle - \langle A \rangle^2} \tag{5.34}$$

It can be shown that the thermalization time is the maximum of all autocorrelation times for all observables and is related to the second largest eigenvalue Λ_2 of the Markov transition matrix W by $\tau^{(th)} = -1/\log \Lambda_2$. It is recommended to thermalize the system for at least ten times the thermalization time before starting measurements.

Chapter 6

Percolation

6.1 Introduction

While the molecular dynamics and Monte Carlo methods can, in principle, be used to study any physical system, difficulties appear close to phase transitions. In our investigation of phase transitions we will start with percolation, a very simple and purely geometric topic.

Although there is no dynamics of any kind involved, percolation nevertheless exhibits complex behavior and a phase transition. This simple model will allow us to introduce many concepts that we will need later for the simulation of dynamic systems. These concepts include:

- Phase transitions
- Scaling
- Finite size effects and finite size scaling
- Monte Carlo Simulations
- Analytic and numeric renormalization group methods
- Series expansion methods

Percolation models can be applied to a variety of problems:

- *Oil fields*: The "swiss cheese model" can be used as a simplified model for the storage and flow of liquids in porous media. The porous stone is modeled by spherical cavities distributed randomly in the surrounding medium. If the density of cavities gets larger they start to overlap and form large cluster of cavities. Important questions that can be asked include:
  - For oil drilling we want to know how the amount of oil stored varies with the size of the oil field.
  - Fluids can only flow through the medium of there is a cluster connecting opposite sides. Such a cluster is called a "percolating" cluster. What density of cavities we need to create a percolating cluster.

  - Next we want to know how the the speed of fluid flow through the medium depends on the density.

- **Forest fires**: We model forest fires by assuming that a tree neighboring a burning tree catches fire with a probability p. Fire fighters will want to know:
  - How much of the forest will burn?
  - Will the fire spread throughout the whole forest?
- **Spread of diseases**: The spread of diseases can be modeled in a simplified way similar to the forest fire model. Now the important question is: what part of the population will fall ill? Will the disease by an epidemic, spreading throughout the whole country, or even an endemic, spreading over the whole world?
- **Conductance of wire meshes**:
  - How many links can be cut in a wire mesh so that it is still' conducting?
  - What is the resistivity as a function of the ratio of cut links?
- **Vulnerability of the internet**: The internet was designed in 1964 to make computer networks reliable even in the case of attacks in war time. The ' most urgent question is:
  - What portion of the internet is still connected if a fraction p of switches fails?
- **Gelation of liquids**: Gelation of liquids can be modeled by allowing a molecule to form a bond with a neighboring molecule with probability p. Again the interesting questions are:
  - What is the average size of molecule clusters as a function of p?
  - What is the critical concentration at which the largest molecule cluster percolates and the liquid solidifies?
- **Baking of cookies**: A nice household example is given in the textbook by Gould and Tobochnik. Distribute several drops of dough randomly on a cookie tray. When baking the dough will spread and cookies that are close will bake together. If the number of drops is too large we will obtain a percolating cookie, spanning the baking tray from one edge to the other.

For a detailed discussion we refer to the book by Stauffer and Aharony.

6.2 Site percolation on a square lattice

In this lecture we will discuss in detail the simplest of percolation models: site percolation on a two dimensional square lattice. Each square shall be occupied with probability p. Occupied squares that share edges form a cluster.

As can be seen in figure 6.1 or by playing with the Java apples on our web page, for large p there is a **percolating cluster**, i.e. a cluster that spans the lattice from one

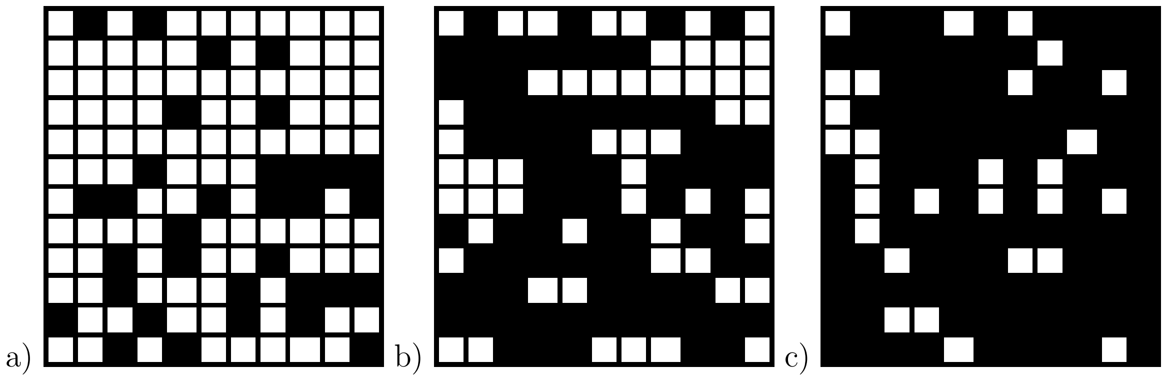


Figure 6.1: Examples of site percolation on a 16 × 16 square lattice for several probabilities: a) $p = 0.2$, b) $p = p_c = 0.5927$, and c) $p = 0.8$. It can be sen that for $p \geq p_c$ there is a spanning cluster, reaching from the top to the bottom edge.

edge to the other. In the infinite lattice limit there is a sharp transition at a critical density p_c. For $p < p_c$ there is *never* a percolating cluster and for $p > p_c$ there is *always* a percolating cluster.

We will ask the following questions:

- What is the critical concentration p_c?
- How do the number of clusters and average cluster size S depend on p?
- What is the probability P that a site is on the percolating cluster?
- What is the resistivity/conductance of the percolating cluster?
- How does a finite lattice size L change the results?
- How do the results depend on the lattice structure and on the specific percolation model used?

The answer to the last question will be, that close to the critical concentration p_c the properties depend only on the dimensionality d and on the type (continuum or lattice) but not on the lattice structure or specific percolation model. Thus our results obtained for the percolation transition on the square lattice will be "universal" in the sense that they will apply also to all other two-dimensional percolation models, like the forest fire model, the spread of diseases and the problems encountered when baking cookies.

6.3 Exact solutions

6.3.1 One dimension

In one dimension the percolation problem can be solved exactly. We will use this exactly soluble case to introduce some quantities of interest.

A percolating cluster spans the whole chain and is not allowed to contain any empty site, thus $p_c = 1$.

The probability that a site is the left edge of a cluster of *finite* size s is simply

$$n_s = (1-p)^2 p^s \tag{6.1}$$

as the cluster consists of s occupied sites neighbored by two empty ones. The probability that a random site is anywhere on an s-site cluster is sn_s. The sum over all cluster sizes s leads to a sum rule which is valid not only in one dimension. The probability that a site is on a cluster of any size is just p:

$$P + \sum_s sn_s = p, \tag{6.2}$$

where P is the probability that a site is on one of the the *infinite* percolating cluster. In one dimension $P = 1$ for $p = p_c$ and $P = 0$ otherwise.

The average cluster size is

$$S = \frac{\sum_s s^2 n_s}{\sum_s sn_s} = \frac{1+p}{1-p} \simeq (p_c - p)^{-1}, \tag{6.3}$$

where the last expression is the universal asymptotic form near p_c.

The pair correlation function $g(r)$ gives the probability that for an occupied site a site at distance r belongs to the same cluster. In one dimensions it is

$$g(r) = p^{|r|} = \exp(-|r|/\xi), \tag{6.4}$$

where

$$\xi = -\frac{1}{\log p} \simeq (p_c - p)^{-1}. \tag{6.5}$$

is called the correlation length. Again there is a sum rule for the pair correlation function:

$$\sum_{r=-\infty}^{\infty} g(r) = S \tag{6.6}$$

6.3.2 Infinite dimensions

Another exact solution is available in infinite dimensions on the Bethe lattice, shown in figure 6.2. It is a tree which, starting from one site branches out, with every site having $z \geq 3$ neighbors.

Why is this lattice infinite dimensional? In d dimensions the volume scales like L^d and the surface like L^{d-1}, thus surface $\propto$ volume$^{1-1/d}$. The Bethe lattice with R generations of sites has a boundary of $z(z-1)^{R-1}$ sites and a volume of $1 + z[(z-1)^R - 1]/(z-2)$ sites. For large R we have surface $\simeq$ volume $\times\, (z-2)/(z-1)$ and thus $d = \infty$.

As the Bethe lattice contains no loops everything can again be calculated exactly. Let us follow a branch in a cluster. At each site it is connected to z sites and branches out to $z-1$ new branches. At each site the cluster branches into $p(z-1)$ occupied branches. If $p(z-1) < 1$ the number of branches decreases and the cluster will be

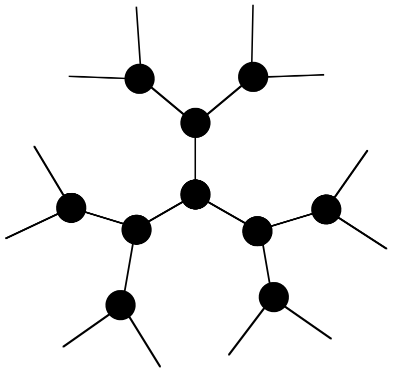


Figure 6.2: Part of the infinite dimensional Bethe lattice for $z = 3$. In this lattice each site is connected to z neighbors.

finite. If $p(z-1) > 1$ the number of branches increases and the cluster will be infinite. Thus

$$p_c = \frac{1}{z-1} \tag{6.7}$$

P can be calculated exactly for $z = 3$ using simple arguments that are discussed in detail in the book by Aharony and Stauffer. We define by Q the probability that a cluster starting at the root of one of the branches does *not* connect to infinity. This is the case if either (i) the root of the branch is empty or (ii) it is occupied but neither of the branches connected to it extends to infinity. Thus for $z = 3$

$$Q = 1 - p + pQ^2 \tag{6.8}$$

This equation has two solutions: $Q = 1$ and $Q = (1-p)/p$. The probability that a site is *not* on the percolating cluster is then $1 - p + pQ^3$, as it is either empty or occupied but not connected to infinty by any of the $z = 3$ branches connected to the site. This gives $P = 0$, corresponding to $p < p_c$ and

$$P = 1 - (1 - p + pQ^3) = p\left[1 - \left(\frac{1-p}{p}\right)^3\right] \simeq (p - p_c) \tag{6.9}$$

for $p > p_c = 1/2$ and $P = 0$ for $p < p_c$. Similar calculations can also be performed analytically also for larger z, always leading to the same power law.

A similar argument can be found for the mean cluster size for $p < p_c$. Let us call the size of a cluster on a branch T. This is 0 if the root of the branch is empty and $1 + (z-1)T$ if it is occupied. Thus: $T = p(1 + (z-1)T)$, with the solution $T = p/(p + 1 - pz)$. The size of the total cluster is the root plus the three branches:

$$S = 1 + zT = \frac{1+p}{1-(z-1)p} \simeq (p - p_c)^{-1} \tag{6.10}$$

What about the cluster probabilities n_s? A cluster of s sites has $2 + (z-2)s$ neighboring empty sites. By looking at the ratio $n_s(p)/n_s(p_c)$ we get rid of prefactors and obtain:

$$\frac{n_s(p)}{n_s(p_c)} = \left(\frac{p}{p_c}\right)^s \left(\frac{1-p}{1-p_c}\right)^{2+(z-2)s} \tag{6.11}$$

For $z = 3$ this is asymptotically is asymptotically

$$\frac{n_s(p)}{n_s(p_c)} \simeq \exp(-cs) \qquad \text{with} \qquad c = -\log[1 - 4(p - p_c)^2] \simeq (p - p_c)^2. \tag{6.12}$$

All that is missing now is an expression for $n_s(p_c)$. Unfortunately this cannot be obtained exactly for arbitrary n. Instead we make an educated guess. As $S = \sum_s s^2 n_s(p)/\sum_s sn_s(p)$ must diverge at $p = p_c$ and $p = \sum_s sn_s(p) + P$ is finite, $n_s(p_c)$ must decay faster than s^{-2} but slower than s^{-3}. We make the ansatz first suggested by M.E. Fisher:

$$n_s(p_c) \simeq s^{-\tau} \tag{6.13}$$

with $2 < \tau \leq 3$.

From the summation

$$S = \frac{\sum_s s^2 n_s(p)}{\sum_s sn_s(p)} \simeq (p_c - p)^{2\tau-6} \tag{6.14}$$

we obtain by comparing with equation (6.10) that

$$\tau = 5/2. \tag{6.15}$$

By using equations (6.12) and (6.13) we can re-derive the asymptotic power laws of equations (6.9) and (6.10).

6.4 Scaling

We have seen that in both exactly solvable cases the interesting quantities have power law singularities at p_c. The generalization of this to arbitrary lattices is the "scaling ansatz", which cannot be proven. However it can be motivated from the fractal behavior of the percolating cluster, from renormalization group arguments and from the good agreement of this ansatz with numerical results.

6.4.1 The scaling ansatz

We generalize the scaling for the average cluster size S to:

$$S \propto |p - p_c|^{\gamma} \tag{6.16}$$

where γ need not be equal to 1 in general. In the calculation of the average cluster size we omit, as done before, any existing percolating cluster, as that would give an infinite contribution.

The correlation length ξ can be defined as

$$\xi^2 = \frac{\sum_r r^2 g(r)}{\sum_r g(r)}, \tag{6.17}$$

which is equivalent to the previous definition from the exponential decay of the correlation function. The sum over all distances can be split into sums over all clusters adding the contribution of each cluster:

$$\xi^2 = \frac{2\sum_{cluster} R^2_{cluster} n(cluster)^2}{\sum_{cluster} n(cluster)^2}, \tag{6.18}$$

where

$$R^2_{cluster} = \frac{1}{2n(cluster)^2} \sum_{i,j \in cluster} |\mathbf{r}_i - \mathbf{r}_j|^2 \tag{6.19}$$

is the average radius of the cluster, $n(cluster)$ the size of the cluster and $\mathbf{r}_i$ the location of the i-th site . The definition 6.18) through the cluster "moment of inertia" is the most natural one for simulations.

Also for the correlation length ξ we define an exponent ν:

$$\xi \propto |p - p_c|^{-\nu} \tag{6.20}$$

The pair correlation function $g(r)$ for $p \neq p_c$ decays exponentially with the correlation length ξ. At the critical concentration $p = p_c$ however the correlation length ξ diverges and we assume a power law:

$$g(r) \propto r^{-(d-2+\eta)} \tag{6.21}$$

For the probability P that a site is on the percolating cluster we make the ansatz:

$$P \propto (p - p_c)^{\beta} \tag{6.22}$$

Finally we define an exponent for the cluster density $M_0 = \sum_s n_s$ which scales as

$$M_0 \propto |p - p_c|^{2-\alpha} \tag{6.23}$$

Fortunately we do not have to calculate all five exponents, as there exist scaling relations between them. To derive them we start from an ansatz for the cluster numbers n_s which is motivated from the previous exact results:

$$n_s(p) = s^{-\tau} f[(p - p_c)s^{\sigma}], \tag{6.24}$$

where f is a scaling function that needs to be determined and σ and τ are two universal exponents. Both the one-dimensional as well as the infinite-dimensional results can be cast in that scaling form.

Starting from that ansatz we can – using some series summation tricks presented in the exercises – calculate P, S, and M_0 similar to what we did in one dimension. We obtain:

$$\beta = \frac{\tau - 2}{\sigma} \tag{6.25}$$

$$\gamma = \frac{3 - \tau}{\sigma} \tag{6.26}$$

$$2 - \alpha = \frac{\tau - 1}{\sigma} \tag{6.27}$$

from which we can derive the scaling law:

$$2 - \alpha = 2\beta + \gamma \tag{6.28}$$

6.4.2 Fractals

For the further discussion we need to introduce the concept of fractal dimensions, which should be familiar to all of you. As you can see by looking at pictures or playing with the applet on the web page, the percolating cluster at criticality is a complex object and self-similar on all length scales.

This self-similarity follows naturally from the divergence of the dominant length scale ξ at the critical point and is reflected in the power-law behavior of all properties at the critical point. Power laws are the only scale-invariant functions $f(r/l) \propto f(r)$, as $r^\zeta \propto (r/l)^\zeta$.

Thus self-similarity and fractal behavior are intimately related to the scaling ansatz. Whether you use the scaling ansatz to motivate fractal behavior, or use apparent fractal behavior to motivate the scaling ansatz is a matter of taste.

Self-similar objects like the percolating cluster at criticality are called fractals, since their dimension D defined by the relationship of volume to linear dimension:

$$V(R) \propto R^D \tag{6.29}$$

is a non-integral fraction D. This is in contrast to simple objects like lines, squares or cubes which have integer dimension.

Applying these idea to clusters we make the ansatz

$$R_s \propto s^{1/D} \tag{6.30}$$

for the average radius of a cluster of s sites. This allows us to evaluate equation (6.18) as:

$$\xi^2 = \frac{2\sum_s R_s^2 s^2 n_s}{\sum_s s^2 n_s}, \tag{6.31}$$

since a) R_s is the mean distance between two sites in a cluster b) a site is connected to s other sites, and c) $n_s s$ is the probability of the site belonging to an s-site cluster. The summation can again be performed analogous to the ones used to obtain Eqs. (6.25)-(6.27). Using these equations to replace σ and τ by the other exponents we find:

$$D = (\beta + \gamma)/\nu = d - \beta/\nu, \tag{6.32}$$

where the last equality will be derived later using finite size scaling.

By integrating equation (6.21) one can also obtain the fractal dimension, leading to another scaling law:

$$d - 2 + \eta = 2\beta/\nu \tag{6.33}$$

6.4.3 Hyperscaling and upper critical dimension

Finally, we wish to mention the“hyper scaling” - law:

$$d\nu = \gamma + 2\beta = 2 - \alpha \tag{6.34}$$

which is obtained combining several of the previously derived scaling relations The scaling laws involving dimensions are usually called “hyper scaling” laws. While the other scaling laws hold for-all dimensions this one will eventually break down for large d as it is clearly not valid for the Bethe lattice. It holds up to the "upper critical dimension" $d_u = 6$ in the case of percolation. For $d \geq d_u$ the exponents are always the same as those of the infinite dimensional Bethe lattice, but hyperscaling breaks down for $d > d_u = 6$. As the most interesting physical problems are in dimensions below d_u. we will not discuss this issue further but instead concentrate on methods to calculate properties of percolation models below d_u.

6.5 Renormalization group

The renormalization group method is intricately linked to the self similarity of the percolating cluster and to the scaling ansatz. It also provides a "motivation" for the scaling ansatz.

The idea is to ignore unimportant microscopic details and to concentrate on the important physics on large scales. We do that by replacing a $b \times b$ square of our square lattice by a single square. We choose it to be filled if a percolating cluster exists on the $b \times b$ square and empty otherwise. This process is iterated until we are left with just one single square which is either filled or empty.

6.5.1 The square lattice

As the simplest example let us choose $b = 2$. On a 2×2 square there are two vertically spanning clusters with 2 occupied sites, four spanning clusters with three occupied sites and one spanning-cluster with four occupied sites. The total probability $R(p)$ for a vertically spanning cluster on a 2 x 2 is thus

$$R(p) = 2p^2(1-p)^2 + 4p^3(1-p)^3 + p^4 \tag{6.35}$$

The renormalization group transformation

$$p \leftarrow R(p) = 2p^2(1-p)^2 + 4p^3(1-p) + p^4 \tag{6.36}$$

has two trivial fixed points $p = 0$ and $p = 1$ as well as one non-trivial fixed point $p_2^* = (\sqrt{(5)} - 1)/2 \approx 0.6180$. This is surprisingly close to the correct result $p_c = 0.5927$

To obtain better accuracy one needs to work with larger cell sizes b. In the limit $b \to \infty$ we have $p_b^* \to p_c$ and the renormalization group calculation becomes exact.

It is important to note that the choice of percolation criterion is ambiguous. Here we have chosen to define a percolating cluster as one that spans the lattice vertically. We can as well choose that the lattice should be spanned horizontally, obtaining the identical results. The other choices, while giving the same results in the limit $b \to \infty$ have larger finite size corrections. If we define a cluster as percolating if it spans horizontally or vertically, the renormalization group transformation is

$$R(p) = 4p^2(1-p)^2 + 4p^3(1-p) + p^4 \tag{6.37}$$

with a fixed point at $(3 - \sqrt{5})/2 \approx 0.382$. If on the other hand we define a percolating cluster as spanning both horizontally *and* vertically, the renormalization group transformation is

$$R(p) = 4p^3(1-p)^3 + p^4 \tag{6.38}$$

with a fixed point at $(\sqrt{13}-1)/6 \approx 0.768$. Both these estimates are much farther from p_c. Our first choice thus turns out to have been the best.

The renormalization group method provides us not only with an estimate for p_c but also with estimates for the exponents. In one step we transform:

$$\begin{aligned} p' &= R(p) && (6.39) \\ \xi' &= \xi/b && (6.40) \end{aligned}$$

At the same time the scaling law (6.20) is valid and we obtain:

$$(p' - p_b^*)^{-\nu} = \xi' = \frac{\xi}{b} = \frac{1}{b}(p - p_b^*)^{-\nu} \tag{6.41}$$

By expanding $R(p)$ in a Taylor series around p_b^* we obtain after a few algebraic transformations:

$$\nu = \frac{\log b}{\log \left.\frac{dR}{dp}\right|_{p_b^*}}. \tag{6.42}$$

For $b = 2$ this gives the very crude estimate $\nu \approx 1.635$, compared to the exact value $\nu = 4/3$.

6.5.2 The triangular lattice

A better estimate is obtained on the triangular lattice. There we replace three sites by one (thus $b^2 = 3$), obtaining the transformation

$$p \leftarrow R(p) = 3p^2(1-p) + p^3 \tag{6.43}$$

with fixed points at 0, 1/2 and 1. In this case surprisingly the value $p^*_{\sqrt{3}} = p_c$ is exact.

Also the estimate for ν is much better: 1.355. It will be necessary to go to larger values of b if we want to improve the accuracy of our results. Up to $b = 6$ we might be able to determine the RG equations exactly. For larger b we have to use numerical methods, which will be the topic of the next section.

6.6 Monte Carlo simulation

Monte Carlo simulations are the simplest method to investigate percolation and can be done, for example, by visual inspection using our applet or by a more extended simulation. We will focus on three types of questions that can be answered by Monte Carlo simulations:

1. What is the p-dependence of an arbitrary quantity X?

2. What is the critical probability p_c?

3. What are the values of the universal critical exponents?

X can be a quantity like ξ, S, P or any other interesting observable.

For now we treat only with finite lattices of linear dimension $L < \infty$. The problem of extrapolation to $L \to \infty$ will be discussed later.

The expectation value of the quantity X can be calculated exactly for small lattices by a sum over all possible configurations on a system with $N = L^d$ sites:

$$\langle X \rangle = \sum_{n=0}^{N} \sum_{c \in \mathcal{C}_{N,n}} p^n (1-p)^{N-n} X(c) \tag{6.44}$$

where $\mathcal{C}_{N,n}$ is the set of configurations of n occupied sites in a lattice with N sites, and $X(c)$ is the value of the quantity X measured in the configuartion c. It is obvious that this sum can be performed exactly only for small lattice sizes up to $N \approx 30$ sites, as the number of terms increases exponentially like 2^N.

6.6.1 Monte Carlo estimates

Larger lattice sizes with N up to 10^6 can no longer be done exactly, but Monte CarIo summation can provide estimates of this average to any desired accuracy. The average over the complete set of configurations is replaced by a random sample of $M \approx 10^6 \dots 10^{12}$ configurations c_i drawn randomly with the correct probabilities $p^n(1-p)^{N-n}$ for a configuration with n occupied sites.

To create configurations wth the correct probabilities we draw a pseudo-random number u, uniformly distributed in $[0, 1[$ for each site j. We set the site occupied if $u < p$ and empty otherwise. This corresponds to importance sampling, where each configuration is created with the correct probability.

6.6.2 Cluster labeling

Next we identify clusters using, for example, the Hoshen-Kopelman cluster labeling algorithm, which works the following way:

- Allocate an array of size N to store the cluster label for each site.
- Loop through all sites i in the lattice. For occupied sites check if a cluster label has already been assigned to any occupied neighboring sites.
  - If no neighboring site is occupied or has a label assigned, assign a new cluster label to this site.
  - If one neighboring site is occupied and has a cluster label assigned, assign the same label to this site.
  - If more than one neighboring sites are occupied and all have the same label assigned, assign this same label to the current site.

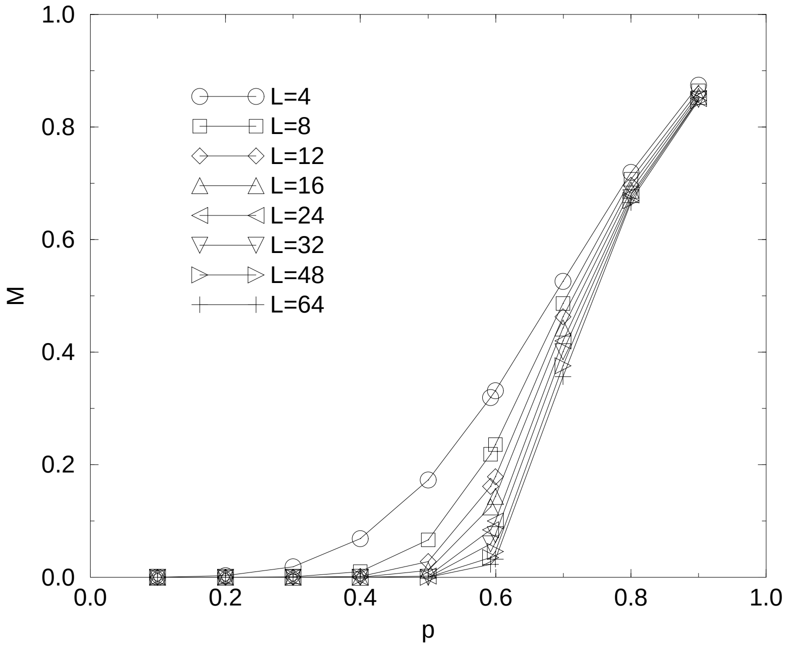


Figure 6.3: Probability P for a site to be on the percolating cluster, as a function of lattice size L for different concentrations p.

  – If more than one neighboring site is occupied and the sites have different labels assigned, we have a problem. The current site connects two cluster parts which until now were disconnected and labeled as different clusters. We take the smallest of label numbers as the proper label and assign this to the current site. For the other, larger, labels we would need to relabel all wrongly labeled sites.

- We use the following trick to avoid relabeling all wrongly labeled sites. For each label we keep a list of the "proper" labels, initialized originally to the label itself. To obtain the cluster label of a site we first obtain the label number assigned to the site. Next we check its proper label. If the two agree we are done. Otherwise we replace the label by the stored proper label and repeat this process until the label agrees with its proper label. This way we avoid relabeling all wrongly labeled sites.

Having thus created a configuration and identified the existing clusters we can measure the value of any quantity of interest. Care must be taken that in the averages over clusters for S and other quantities any existing percolating cluster is always excluded since its contribution is infinite in the thermodynamic limit $L \to \infty$.

6.6.3 Finite size effects

In figure 6.3 we show Monte Carlo results obtained for P in a series of Monte Carlo simulations. We plot P as a function of p for several different system sizes L. It can be seen that for $p \ll p_c$ P converges rapidly to zero as L is increased. For $p \gg p_c$ on the other hand it converges rapidly to a finite value, as expected since P is the probability of a site belonging to the percolating cluster, which exists for $p > p_c$. We can also see that as L is increased a singularity starts to develop at p_c and it is plausible that P follows a power law $P \propto (p - p_c)^\beta$ in the infinite system.

However we also see that convergence is slow around p_c and we have no good way to directly estimate p_c nor the exponents. The problem is that the correlation length ξ diverges as $|p - p_c|^{-\nu}$ and we would need lattices $L \gg \xi \to \infty$, which is impossible. We thus need a new idea.

6.6.4 Finite size scaling

The new idea is to extend scaling so that it also contains the system size L as a parameter. The motivation is simple. A finite system size L introduces a cutoff for all length scales. Then on the finite system ξ cannot grow larger than L. This means that a finite system size L has the same effect as a finite distance from the critical point, related by $L \sim \xi \propto (p - p_c)^{-\nu}$. Thus, for example, at the critical point

$$P(L) \simeq (p - p_c)^\beta \propto L^{-\beta/\nu}. \tag{6.45}$$

More formally we can make the following scaling ansatz. Close to criticality the only important length scale is ξ. The effects of finite system sizes are thus determined only by the ratio L/ξ. For a quantity X, which diverges as $(p - p_c)^{-\chi}$ for $L \gg \xi$ we make the ansatz:

$$X(L, p) = (p - p_c)^{-\chi} \tilde{\mathcal{F}}_1(L/\xi) = (p - p_c)^{-\chi} \mathcal{F}_1((p - p_c)L^{1/\nu}) \tag{6.46}$$

or equivalently

$$X(L, \xi) = \xi^{\chi/\nu} \mathcal{F}_2(L/\xi) \propto \begin{cases} \xi^{\chi/\nu} & \text{for } L \gg \xi \\ L^{\chi/\nu} & \text{for } \xi \gg L \end{cases} \tag{6.47}$$

Applying this scaling ansatz to the size of the percolating cluster $L^d P$ we immediately derive the second expression for the fractal dimension D in equation (6.32):

$$L^D = L^d P(L, \xi) = L^d \xi^{-\beta/\nu} f(L/\xi) \propto L^{d - \beta/\nu} \tag{6.48}$$

where we have chosen $L = \mathit{const} \times \xi$, allowing us to replace ξ by L, and giving a constant value for the scaling function $f(L/\xi)$.

Thus we see that finite size scaling allows us to determine the ratio of exponents like β/ν by calculating the L-dependence of P at $p = p_c$.

One problem however still remains: how can we determine p_c on a finite lattice? The answer is again finite size scaling. Consider the probability $\Pi(p)$ for the existence of a percolating cluster. In the infinite system it is

$$\Pi(p) = \Theta(p - p_c). \tag{6.49}$$

In a finite system the step function is smeared out. We can make the usual finite size scaling ansatz using equation (6.46)

$$\Pi(p, L) = \Phi((p - p_c)L^{1/\nu}). \tag{6.50}$$

The derivative $d\Pi/dp$ gives the probability for first finding a percolating cluster at concentrations in the interval $[p, p + dp[$. The probability p at which a percolating cluster appears can easily be measured in a Monte Carlo simulation. Its average

$$p_{av} = \int p \frac{d\Pi}{dp} dp \tag{6.51}$$

is slightly different from the exact value p_c on any finite lattice, but converges like

$$p_{av} - p_c \propto L^{-1/\nu} \tag{6.52}$$

as can be seen by integrating the scaling ansatz 6.50. Similarly the variance

$$\Delta^2 = \int (\tilde{p} - p_{av})^2 \frac{d\Pi}{dp} dp \tag{6.53}$$

decreases as

$$\Delta \propto L^{-1/\nu} \tag{6.54}$$

That we we can obtain ν and p_c from the finite size scaling of the average p at which a percolating cluster appears on a finite lattice.

We do this by drawing a pseudo-random number u_j, uniformly distributed in $[0, 1[$ for each site j. Next we use a binary search to determine the probability $\tilde{p}$ where a percolating cluster appears for this realization of the u_js. We start by checking $\tilde{p} = 0.5$. We occupy all squares with $u_j < \tilde{p}$. If a percolating cluster exists we set $\tilde{p} = 0.25$, otherwise $\tilde{p} = 0.75$. Next we check if there is a percolating cluster for the same u_js and the new $\tilde{p}$. By repeating this bisection we improve the accuracy by a factor of two in each step and obtain the desired value in a logarithmic number of steps $-\log \epsilon / \log 2$, where ϵ is the desired accuracy. This gives us one sample for $\tilde{p}$. By repeating the whole procedure with new sets of u_js we can obtain good estimates for p_{av} and Δ.

Finally we fit the results obtained for several lattice sizes L to obtain p_c and ν as fitting parameters.

Once we have a good estimate for p_c we can also use equation (6.45) to get estimates for further exponents. By measuring $P(p_c)$ as a function of lattice size L and fitting to equation (6.45) we estimate the ratio of exponents β/ν. Similarly, by fitting $S(p_c)$ as a function of L we estimate γ/ν. The scaling laws which relate the exponents can be used as a check on our simulations and as a tool to increase the accuracy of our estimates.

6.7 Monte Carlo renormalization group

Higher precision than in simple Monte Carlo simulations can be obtained by using Monte Carlo estimates to perform renormalization group calculations for huge block sizes b, often $b \approx 500$.

To determine the renormalization group transformation $R(p)$ in a Monte Carlo simulation we write it as:

$$R(p) = \sum_{n=0}^{N} \begin{pmatrix} N \\ n \end{pmatrix} p^n (1-p)^{N-n} P(n), \tag{6.55}$$

where $N = b^2$ is the number of sites in the block and $P(n)$ is the probability that a percolating cluster exists if n sites are occupied. The easiest way to calculate $P(n)$ is to start from an empty lattice and to add occupied sites until at a certain number of s sites a percolating cluster is formed. Then we set $P(n) = 1$ for $n \geq s$ and $P(n) = 0$ for $n < s$. Averaging over many configurations provides us with a good Monte Carlo estimate for $P(n)$. The critical point p_b^* is found by looking for the fixed point with $R(p_b^*) = p_b^*$. The exponents, like ν are obtained from the derivatives of $R(p)$ using equation (6.42), as we did in the analytic RG calculation.

The corrections due to finite block sizes b can be extrapolated using the following equations, again determined by finite size scaling:

$$\begin{aligned} \nu(b) - \nu &\propto 1/\log b, & (6.56) \\ p_b^* - p_c &\propto b^{-\frac{1}{\nu}} & (6.57) \end{aligned}$$

Using $b \approx 500$ one can obtain estimates for the exponents that are accurate to four digits!

6.8 Series expansion

Series expansion is a very different method, trying to make maximum use of analytically known results for small systems to obtain high precision e xtrapolations to large systems. The basis is an exact calculation for the probability $n_s(p)$ of small clusters. For example, on a square lattice we have

$$\begin{aligned} n_1(p) &= p(1-p)^4 & (6.58) \\ n_2(p) &= 2p^2(1-p)^6 & (6.59) \\ n_3(p) &= 2p^3(1-p)^8 + 4p^3(1-p)^7 & (6.60) \\ &\vdots & (6.61) \end{aligned}$$

In the calculation of these probabilities we needed to enumerate the different geometries of s-site clusters and calculate the number of times they can be embedded into a square lattice. Once we have calculated the cluster numbers n_s for clusters up to size s we can calculate the first s terms of a Taylor expansion in p for any quantity of interest. This Taylor series can be used as the basis of several analysis procedures, of which we mention only two often used approaches.

The ratio method determines critical points and exponents from ratios of successive expansion coefficients. Look, for example, at the Taylor series expansion for a function like $\xi \propto (p_c - p)^{-\nu}$:

$$(p_c - p)^{-\nu} = p_c^{-\nu} \sum_{i=0}^{\infty} a_i p^i = p_c^{-\nu} \left(1 + \frac{\nu}{p_c} p + \frac{\nu(\nu+1)}{2p_c^2} p^2 + \ldots\right) \tag{6.62}$$

Table 6.1: Critical exponents for percolation in any dimension

functional form	exponent	$d=1$	$d=2$	$d=3$	$d=4$	$d=5$	$d\geq 6$
$M_0 \propto \|p-p_c\|^{2-\alpha}$	α	1	-2/3	-0.62	-0.72	-0.86	- 1
$n_s(p=p_c) \propto s^{-\tau}$	τ	2	187/91	2.18	2.31	2.41	5/2
$S \propto \|p-p_c\|^{-\gamma}$	γ	1	43/18	1.80	1.44	1.18	1
$\xi \propto \|p-p_c\|^{-\nu}$	ν	1	4/3	0.88	0.68	0.57	1/2
$g(r,p=p_c) \propto r^{-(d-2+\eta)}$	η	1	5/24	-0.07	-0.12	-0.05	0
$P \propto (p-p_c)^{\beta}$	β	–	5/36	0.41	0.64	0.84	1
fractal dimension	D	1	91/48	2.53	3.06	3.54	4

with the following simple relation between the coefficients:

$$\frac{a_i}{a_{i-1}} = \frac{1}{p_c}\left(1 + \frac{\nu-1}{i}\right) \tag{6.63}$$

As $\xi(p)$ is asymptotically proportional to $(p-p_c)^{-\nu}$ the series will be dominated by this asymptotic behavior for large coefficients and a fit of the ratios to above equation (6.63) will give estimates for p_c and ν.

The other, often more accurate method is the Dlog-Padé method. Its basis is that a function like $\xi \simeq (p-p_c)^{-\nu}$ has a logarithmic derivative

$$\frac{d}{dp}\log\xi(p) \simeq \frac{d}{dp}\log(p-p_c)^{-\nu} = \frac{\nu}{p-p_c} \tag{6.64}$$

The logarithmic derivative of ξ with respect to p will thus have a pole at p_c with residuum ν, refecting the singularity at p_c. To estimate the pole and residuum we use Padé approximants to represent the series for ξ as a rational function:

$$\xi \approx \sum_{i=0}^{N} a_i p^i = \frac{\sum_{j=0}^{L} b_j p^j}{\sum_{k=0}^{M} c_k p^k} \tag{6.65}$$

with $L+M=N$. The coefficients are determined by matching the first N terms in the Taylor series of the left and right hand sides.

The zeroes of the polynomial in the denominator give the poles, one of which will be a good estimate for p_c. By using different values L and M the accuracy of the resulting estimates can be checked. This was the method of choice before large computers became available and is still in use today, when computers are used to obtain n_s symbolically for large values of s.

6.9 Listing of the universal exponents

In table 6.1 we give an overview over the various exponents calculated for dimensions $d < d_u = 6$ as well as the Bethe lattice results, valid for any $d \geq d_u$.

We wish to point out that besides the exponents there exist further universal quantities, like the universal amplitude ratios

$$\Gamma = \frac{S(p_c-\epsilon)}{S(p_c+\epsilon)} \tag{6.66}$$

Finally we wish to repeat that these exponents are universal in the sense that they depend only on dimensionality, but not on the specific percolation model used. Thus all the examples mentioned in the introduction to this chapter: forest fires, oil reservoirs, gelation, spreading of diseases, etc., share the same exponents as long as the dimensionality is the same.

Thus, although we have considered only toy models we get widely applicable results. This is a common thread that we will encounter again. While performing numerical simulations for specific models is nice, being able to extract universally valid results from specific model simulations is the high art of computer simulations.

Chapter 7

Magnetic systems

In this chapter we move away from the 'static' problem of percolation to the more dynamic problem of evaluating thermodynamic averages through phase space integrals. In this context we will encounter many of the same problems as in percolation.

7.1 The Ising model

The Ising model is the simplest model for a magnetic system and a prototype statistical system. We will use it for our discussion of thermodynamic phase transitions. It consists of an array of classical spins $\sigma_i = \pm 1$ that can point either up ($\sigma_i = +1$) or down ($\sigma_i = -1$). The Hamiltonian is

$$H = -J \sum_{\langle i,j \rangle} \sigma_i \sigma_j, \tag{7.1}$$

where the sum goes over nearest neighbor spin pairs.

Two parallel spins contribute an energy of $-J$ while two antiparallel ones contribute $+J$. In the ferromagnetic case the state of lowest energy is the fully polarized state where all spins are aligned, either pointing up or down.

At finite temperatures the spins start to fluctuate and also states of higher energy contribute to thermal averages. The average magnetization thus decreases from its full value at zero temperature. At a critical temperature T_c there is a second order phase transition to a disordered phase, similar to the one we discussed in the percolation problem.

The Ising model is the simplest magnetic model exhibiting such a phase transition and is often used as a prototype model for magnetism. To discuss the phase transition the scaling hypothesis introduced in the context of percolation will again be used.

As is known from the statistical mechanics course the thermal average of a quantity A at a finite temperature T is given by a sum over all states:

$$\langle A \rangle = \frac{1}{Z} \sum_i A_i \exp(-\beta E_i), \tag{7.2}$$

where $\beta = 1/k_B T$ is the inverse temperature. A_i is the value of the quantity A in the configuration i. E_i is the energy of that configuration.

The partition function ("Zustandssumme")

$$Z = \sum_i \exp(-\beta E_i) \tag{7.3}$$

normalizes the probabilities $p_i = \exp(-\beta E_i)/Z$.

For small systems it is possible to evaluate these sums exactly. As the number of states grows like 2^N a straight-forward summation is possible only for very small N. For large higher dimensional systems Monte Carlo summation/integration is the method of choice.

7.2 The single spin flip Metropolis algorithm

As was discussed in connection with integration it is usually not efficient to estimate the average (7.2) using simple sampling. The optimal method is importance sampling, where the states i are not chosen uniformly but with the correct probability p_i, which we can again do using the Metropolis algorithm.

The simplest Monte Carlo algorithm for the Ising model is the single spin flip Metropolis algorithm which defines a Markov chain through phase space.

- Starting with a configuration c_i propose to flip a single spin, leading to a new configuration c'.
- Calculate the energy difference $\Delta E = E[c'] - E[c_i]$ between the configurations c' and c_i.
- If $\Delta E < 0$ the next configuration is $c_{i+1} = c'$
- If $\Delta E > 0$ then $c_{i+1} = c'$ with probability $\exp(-\beta\Delta E)$, otherwise $c_{i+1} = c_i$. We do that by drawing a random number r uniformly distributed in the interval $[0, 1[$ and set $c_{i+1} = c'$ if $r < \exp(-\beta\Delta E)$.
- Measure all the quantities of interest in the new configuration.

This algorithm is ergodic since any configuration can be reached from any other in a finite number of spin flips. It also fulfills the detailed balance condition.

7.3 Systematic errors: boundary and finite size effects

In addition to statistical errors due to the Monte Carlo sampling our simulations suffer from systematic errors due to boundary effects and the finite size of the system.

In contrast to the percolation problems boundary effects can be avoided completely by using periodic boundary conditions. The lattice is continued periodically, forming a torus. The left neighbor of the leftmost spin is just the rightmost boundary spin, etc..

Although we can thus avoid boundary effects, finite size effects remain since now all correlations are periodic with the linear system size as period. In the context of the percolation problem we have already learned how to deal with finite size effects:

- *Away from phase transitions* the correlation length ξ is finite and finite size effects are negligible if the linear system size $L \gg \xi$. Usually $L > 6\xi$ is sufficient, but this should be checked for each simulation.

- *In the vicinity of continuous phase transitions* we encounter the same problem as in percolation: the correlation length ξ diverges. Again finite size scaling comes to the rescue and we can obtain the critical behavior as discussed in the chapter on percolation.

7.4 Critical behavior of the Ising model

Close to the phase transition at T_c again scaling laws characterize the behavior of all physical quantities. The average magnetization scales as

$$m(T) = \langle |M|/V \rangle \propto (T_c - T)^{\beta}, \tag{7.4}$$

where M is the total magnetization and V the system volume (number of spins).

The magnetic susceptibility $\chi = \frac{dm}{dh}|_{h=0}$ can be calculated from magnetization fluctuations and diverges with the exponent γ:

$$\chi(T) = \frac{\langle M^2/V \rangle - \langle |M| \rangle^2/V}{T} \propto |T_c - T|^{-\gamma}. \tag{7.5}$$

The correlation length ξ is defined by the asymptotically exponential decay of the two-spin correlations:

$$\langle \sigma_0 \sigma_{\mathbf{r}} \rangle - \langle |m| \rangle^2 \propto \exp(-r/\xi). \tag{7.6}$$

It is best calculated from the structure factor $S(\mathbf{q})$, defined as the Fourier transform of the correlation function. For small $\mathbf{q}$ the structure factor has a Lorentzian shape:

$$S(\mathbf{q}) = \frac{1}{1 + q^2\xi^2} + \mathrm{O}(q^4). \tag{7.7}$$

The correlation length diverges as

$$\xi(p) \propto |T - T_c|^{-\nu}. \tag{7.8}$$

At the critical point the correlation function again follows the same power law as in the percolation problem:

$$\langle \sigma_0 \sigma_{\mathbf{r}} \rangle \propto r^{-(d-2+\eta)} \tag{7.9}$$

where $\eta = 2\beta/\nu - d + 2$, derived from the same scaling laws as in percolation.

The specific heat $C(T)$ diverges logarithmically in two dimensions:

$$C(T) \propto \ln |T - T_c| \propto |T - T_c|^{-\alpha} \tag{7.10}$$

and the critical exponent $\alpha = 0$.

Like in percolation, finite size scaling is the method of choice for the determination of these exponents.

Table 7.1: Critical exponents for the Ising model in two and three dimensions.

quantity	functional form	exponent	Ising $d = 2$	Ising $d = 3$
magnetization	$m \propto (T_c - T)^\beta$	β	1/8	0.3258(44)
susceptibility	$\chi \propto \lvert T - T_c\rvert^{-\gamma}$	γ	7/4	1.2390(25)
correlation length	$\xi \propto \lvert T - T_c\rvert^{-\nu}$	ν	1	0.6294(2)
specific heat	$C(T) \propto \lvert T - T_c\rvert^{-\alpha}$	α	0	
inverse critical temp.		$1/T_c$	$\frac{1}{2}\ln(1+\sqrt{2})$	0.221657(2)

A good estimate of T_c is obtained from the Binder cumulant

$$U = 1 - \frac{\langle M^4\rangle}{3\langle M^2\rangle^2}. \tag{7.11}$$

Just as the function $\Pi(L,p)$ in the percolation problem has a universal value at p_c, also the Binder cumulant has a universal value at T_c. The curves of $U(T)$ for different system sizes L all cross in one point at T_c. This is a consequence of the finite size scaling ansatz:

$$\begin{aligned}\langle M^4\rangle &= (T-T_c)^{4\beta}u_4((T-T_c)L^{1/\nu})\\ \langle M^2\rangle &= (T-T_c)^{2\beta}u_2((T-T_c)L^{1/\nu}).\end{aligned} \tag{7.12}$$

Thus

$$U(T,L) = 1 - \frac{u_4((T-T_c)L^{1/\nu})}{3u_2((T-T_c)L^{1/\nu})^2}, \tag{7.13}$$

which for $T = T_c$ is universal and independent of system size L:

$$U(T_c,L) = 1 - \frac{u_4(0)}{3u_2(0)^2} \tag{7.14}$$

High precision Monte Carlo simulations actually show that not all lines cross exactly at the same point, but that due to higher order corrections to finite size scaling the crossing point moves slightly, proportional to $L^{-1/\nu}$, allowing a high precision estimate of T_c and ν. [1]

In the table we show the exponents and critical temperature of the Ising model in two and three dimensions.

7.5 "Critical slowing down" and cluster Monte Carlo methods

The importance of autocorrelation becomes clear when we wish to simulate the Ising model at low temperatures. The mean magnetization $\langle m\rangle$ is zero on any finite cluster,

[1]See, e.g., A. M. Ferrenberg and D. P. Landau, Phys. Rev. B **44** 5081 (1991); K. Chen, A. M. Ferrenberg and D. P. Landau, Phys. Rev. B **48** 3249 (1993).

as there is a degeneracy between a configuration and its spin reversed counterpart. If, however, we start at low temperatures with a configuration with all spins aligned up it will take extremely long time for all spins to be flipped by the single spin flip algorithm. This problem appears as soon as we get close to the critical temperature, where it was observed that the autocorrelation times diverge as

$$\tau \propto [\min(\xi, L)]^z. \tag{7.15}$$

with a dynamical critical exponents $z \approx 2$ for all local update methods like the single spin flip algorithm.

The reason is that at low temperatures it is very unlikely that even one spin gets flipped, and even more unlikely for a large cluster of spins to be flipped.

The solution to this problem was found in 1987 and 1989 by Swendsen and Wang[2] and by Wolff.[3]

Instead of flipping single spins they propose to flip big clusters of spins and choose them in a clever way so that the probability of flipping these clusters is large.

7.5.1 Kandel-Domany framework

We use the Fortuin-Kastelyn representation of the Ising model, as generalized by Kandel and Domany. The phase space of the Ising model is enlarged by assigning a set $\mathcal{G}$ of possible "graphs" to each configuration C in the set of configurations $\mathcal{C}$. We write the partition function as

$$Z = \sum_{C\in\mathcal{C}} \sum_{G\in\mathcal{G}} W(C,G) \tag{7.16}$$

where the new weights $W(C,G) > 0$ are chosen such that Z is the partition function of the original model by requiring

$$\sum_{G\in\mathcal{G}} W(C,G) = W(C) := \exp(-\beta E[C]), \tag{7.17}$$

where $E[C]$ is the energy of the configuration C.

The algorithm now proceeds as follows. First we assign a graph $G \in \mathcal{G}$ to the configuration C, chosen with the correct probability

$$P_C(G) = W(C,G)/W(C). \tag{7.18}$$

Then we choose a new configuration C' with probability $p[(C,G) \to (C',G)]$, keeping the graph G fixed; next a new graph G' is chosen

$$C \to (C,G) \to (C',G) \to C' \to (C',G') \to \dots \tag{7.19}$$

What about detailed balance? The procedure for choosing graphs with probabilities P_G obeys detailed balance trivially. The non-trivial part is the probability of choosing a new configuration C'. There detailed balance requires:

$$W(C,G)p[(C,G) \to (C',G)] = W(C',G)p[(C',G) \to (C,G)], \tag{7.20}$$

[2]R.H. Swendsen and J-S. Wang, Phys. Rev. Lett. **58**, 86 (1987).
[3]U. Wolff, Phys. Rev. Lett. **62**, 361 (1989)

Table 7.2: Local bond weights for the Kandel-Domany representation of the Ising model.

	$c=\uparrow\uparrow$	$c=\downarrow\uparrow$	$c=\uparrow\downarrow$	$c=\downarrow\downarrow$	V(g)
$\Delta(c,\text{discon.})$	1	1	1	1	$\exp(-\beta J)$
$\Delta(c,\text{con.})$	1	0	0	1	$\exp(\beta J)-\exp(-\beta J)$
w(c)	$\exp(\beta J)$	$\exp(-\beta J)$	$\exp(-\beta J)$	$\exp(\beta J)$	

which can be fulfilled using either the heat bath algorithm

$$p[(C,G)\to(C',G)] = \frac{W(C',G)}{W(C,G)+W(C',G)} \tag{7.21}$$

or by again using the Metropolis algorithm:

$$p[(C,G)\to(C',G)] = \min(W(C',G)/W(C,G),1) \tag{7.22}$$

The algorithm simplifies a lot if we can find a graph mapping such that the graph weights do not depend on the configuration whenever it is nonzero in that configuration. This means, we want the graph weights to be

$$W(C,G) = \Delta(C,G)V(G), \tag{7.23}$$

where

$$\Delta(C,G) := \begin{cases} 1 \text{ if } W(C,G)\neq 0, \\ 0 \text{ otherwise.} \end{cases} . \tag{7.24}$$

Then equation (7.21) simply becomes $p=1/2$ and equation (7.22) reduces to $p=1$ for any configuration C' with $W(C',G)\neq 0$.

7.5.2 The cluster algorithms for the Ising model

Let us now show how this abstract and general algorithm can be applied to the Ising model. Our graphs will be bond-percolation graphs on the lattice. Spins pointing into the same direction can be connected or disconnected. Spins pointing in opposite directions will always be disconnected. In the Ising model we can write the weights $W(C)$ and $W(C,G)$ as products over all bonds b:

$$W(C) = \prod_b w(C_b) \tag{7.25}$$

$$W(C,G) = \prod_b w(C_b,G_b) = \prod_b \Delta(C_b,G_b)V(G_b) \tag{7.26}$$

where the local bond configurations C_b can be one of $\{\uparrow\uparrow,\downarrow\uparrow,\uparrow\downarrow,\downarrow\downarrow\}$

and the local graphs can be "connected" or "disconnected". The graph selection can thus be done locally on each bond.

Table 7.2 shows the local bond weights $w(c,g)$, $w(c)$, $\Delta(c,g)$ and $V(g)$. It can easily be checked that the sum rule (7.17) is satisfied.

The probability of a connected bond is $[\exp(\beta J) - \exp(-\beta J)]/\exp(\beta J) = 1 - \exp(-2\beta J)$ if two spins are aligned and zero otherwise. These connected bonds group the spins into clusters of aligned spins.

A new configuration C' with the same graph G can differ from C only by flipping clusters of connected spins. Thus the name "cluster algorithms". The clusters can be flipped independently, as the flipping probabilities $p[(C, G) \to (C', G)]$ are configuration independent constants.

There are two variants of cluster algorithms that can be constructed using the rules derived above.

7.5.3 The Swendsen-Wang algorithm

The Swendsen-Wang or multi-cluster algorithm proceeds as follows:

i) Each bond in the lattice is assigned a label "connected" or "disconnected" according to above rules. Two aligned spins are connected with probability $1 - \exp(-2\beta J)$. Two antiparallel spins are never connected.

ii) Next a cluster labeling algorithm, like the Hoshen-Kopelman algorithm is used to identify clusters of connected spins.

iii) Measurements are performed, using improved estimators discussed in the next section.

iv) Each cluster of spins is flipped with probability 1/2.

7.5.4 The Wolff algorithm

The Swendsen Wang algorithm gets less efficient in dimensions higher than two as the majority of the clusters will be very small ones, and only a few large clusters exist. The Wolff algorithm is similar to the Swendsen-Wang algorithm but builds only one cluster starting from a randomly chosen point. As the probability of this point being on a cluster of size s is proportional to s the Wolff algorithm builds preferedly larger clusters. It works in the following way:

i) Choose a random spin as the initial cluster.

ii) If a neighboring spin is parallel to the initial spin it will be added to the cluster with probability $1 - \exp(-2\beta J)$.

iii) Repeat step ii) for all points newly added to the cluster and repeat this procedure until no new points can be added.

iv) Perform measurements using improved estimators.

v) Flip all spins in the cluster.

We will see in the next section that the linear cluster size diverges with the correlation length ξ and that the average number of spins in a cluster is just χT. Thus the algorithm adapts optimally to the physics of the system and the dynamical exponent $z \approx 0$, thus solving the problem of critical slowing down. Close to criticality these algorithms are many orders of magnitudes (a factor L^2) better than the local update methods. Away from criticality sometimes a hybrid method, mixing cluster updates and local updates can be the ideal method.

7.6 Improved Estimators

In this section we present a neat trick that can be used in conjunction with cluster algorithms to reduce the variance, and thus the statistical error of Monte Carlo measurements. Not only do these "improved estimators" reduce the variance. They are also much easier to calculate than the usual "simple estimators".

To derive them we consider the Swendsen-Wang algorithm. This algorithm divides the lattice into N_c clusters, where all spins within a cluster are aligned. The next possible configuration is any of the 2^{N_c} configurations that can be reached by flipping any subset of the clusters. The idea behind the "improved estimators" is to measure not only in the new configuration but in all equally probable 2^{N_c} configurations.

As simplest example we consider the average magnetization $\langle m \rangle$. We can measure it as the expectation value $\langle \sigma_{\vec{i}} \rangle$ of a single spin. As the cluster to which the spin belongs can be freely flipped, and the flipped cluster has the same probability as the original one, the improved estimator is

$$\langle m \rangle = \langle \frac{1}{2}(\sigma_{\vec{i}} - \sigma_{\vec{i}}) \rangle = 0. \tag{7.27}$$

This result is obvious because of symmetry, but we saw that at low temperatures a single spin flip algorithm will fail to give this correct result since it takes an enormous time to flip all spins. Thus it is encouraging that the cluster algorithms automatically give the exact result in this case.

Correlation functions are not much harder to measure:

$$\langle \sigma_{\vec{i}} \sigma_{\vec{j}} \rangle = \begin{cases} 1 & \text{if } \vec{i} \text{ und } \vec{j} \text{ are on the same cluster} \\ 0 & \text{otherwise} \end{cases} \tag{7.28}$$

To derive this result consider the two cases and write down the improved estimators by considering all possible cluster flips.

Using this simple result for the correlation functions the mean square of the magnetization is

$$\langle m^2 \rangle = \frac{1}{N^2} \sum_{\vec{i},\vec{j}} \langle \sigma_{\vec{i}} \sigma_{\vec{j}} \rangle = \frac{1}{N^2} \langle \sum_{cluster} S(cluster)^2 \rangle, \tag{7.29}$$

where $S(cluster)$ is the number of spins in a cluster. The susceptibility above T_c is simply given by $\beta \langle m^2 \rangle$ and can also easily be calculated by above sum over the squares of the cluster sizes.

In the Wolff algorithm only a single cluster is built. Above sum (7.29) can be rewritten to be useful also in case of the Wolff algorithm:

$$\begin{aligned}\langle m^2\rangle &= \frac{1}{N^2}\langle \sum_{cluster} S(cluster)^2\rangle \\ &= \frac{1}{N^2}\sum_{\vec{i}}\frac{1}{S(\text{cluster containing }\vec{i})}S(\text{cluster containing }\vec{i})^2 \\ &= \frac{1}{N^2}\sum_{\vec{i}}S(\text{cluster containing }\vec{i}) = \frac{1}{N}\langle S(\text{cluster})\rangle. \end{aligned} \tag{7.30}$$

The expectation value for m^2 is thus simply the mean cluster size. In this derivation we replaced the sum over all clusters by a sum over all sites and had to divide the contribution of each cluster by the number of sites in the cluster. Next we can replace the average over all lattice sites by the expectation value for the cluster on a randomly chosen site, which in the Wolff algorithm will be just the one Wolff cluster we build.

Generalizations to other quantities, like the structure factor $S(\vec{q})$ are straightforward. While the calculation of $S(\vec{q})$ by Fourier transform needs at least $O(N\log N)$ steps, it can be done much faster using improved estimators, here derived for the Wolff algorithm:

$$\begin{aligned}\langle S(\vec{q})\rangle &= \frac{1}{N^2}\sum_{\vec{r},\vec{r}'}\sigma_{\vec{r}}\sigma_{\vec{r}'}\exp(i\vec{q}(\vec{r}-\vec{r}')) \\ &= \frac{1}{NS(cluster)}\sum_{\vec{r},\vec{r}'\in cluster}\sigma_{\vec{r}}\sigma_{\vec{r}'}\exp(i\vec{q}(\vec{r}-\vec{r}')) \\ &= \frac{1}{NS(cluster)}\left|\sum_{\vec{r}\in cluster}\exp(i\vec{q}\vec{r})\right|^2, \end{aligned} \tag{7.31}$$

This needs only $O(S(cluster))$ operations and can be measured directly when constructing the cluster.

Care must be taken for higher order correlation functions. Improved estimators for quantities like m^4 contain terms of the form $\langle S(cluster_1)S(cluster_2)\rangle$, which need at least two clusters and cannot be measured in an improved way using the Wolff algorithm.

7.7 Generalizations of cluster algorithms

Cluster algorithms can be used not only for the Ising model but for a large class of classical, and even quantum spin models. The quantum version is the "loop algorithm", which will be discussed later in the course. In this section we discuss generalizations to other classical spin models.

Before discussing specific models we remark that generalizations to models with different coupling constants on different bonds, or even random couplings are straightforward. All decisions are done locally, individually for each spin or bond, and the couplings can thus be different at each bond.

7.7.1 Potts models

q-state Potts models are the generalization of the Ising model to more than two states. The Hamilton function is

$$H = -J \sum_{\langle i,j \rangle} \delta_{s_i,s_j}, \tag{7.32}$$

where the states s_i can take any integer value in the range $1, \ldots, q$. The 2-state Potts model is just the Ising model with some trivial rescaling.

The cluster algorithms for the Potts models connect spins with probability $1 - e^{-\beta J}$ if the spins have the same value. The clusters are then "flipped" to any arbitrarily chosen value in the range $1, \ldots, q$.

7.7.2 $O(N)$ models

Another, even more important generalization are the $O(N)$ models. Well known examples are the XY-model with $N = 2$ and the Heisenberg model with $N = 3$. In contrast to the Ising model the spins can point into any arbitrary direction on the N-sphere. The spins in the XY model can point into any direction in the plane and can be characterized by a phase. The spins in the Heisenberg model point into any direction on a sphere.

The Hamilton function is:

$$H = -J \sum_{\langle i,j \rangle} \vec{S}_i \vec{S}_j, \tag{7.33}$$

where the states $\vec{S}_i$ are N-dimensional unit vectors.

Cluster algorithms are constructed by projecting all spins onto a random direction $\hat{e}$. The cluster algorithm for the Ising model can then be used for this projection. Two spins $\vec{S}_i$ and $\vec{S}_j$ are connected with probability

$$1 - \exp\left(\min[0, -2\beta J(\hat{e} \cdot \vec{S}_i)(\hat{e} \cdot \vec{S}_j)]\right). \tag{7.34}$$

The spins are flipped by inverting the projection onto the $\hat{e}$-direction:

$$\vec{S}_i \to \vec{S}_i - 2(\hat{e} \cdot \vec{S}_i)\hat{e}. \tag{7.35}$$

In the next update step a new direction $\hat{e}$ on the unit sphere is chosen.

The Heisenberg model

Table 7.3 lists the critical exponents and temperature for the three dimensional Heisenberg model.

In two dimensions models with a continuous symmetry, like the $O(N)$ models with $N \geq 2$ do not exhibit a phase transition at a finite critical temperature, as is proven by the Mermin-Wagner theorem. The reason are the Goldstone-modes, long wavelength spin waves that have vanishingly low excitation energies at long wavelengths. As a consequence the two-dimensional Heisenberg model has $T_c = 0$ and an exponentially growing correlation length at low temperatures $\xi \propto \exp(2\pi J/T)$. We will learn more about this model and finite size scaling from one of the projects.

Table 7.3: Critical properties of the three-dimensional classical Heisenberg model.

quantity	functional form	exponent	
magnetization	$m \propto (T_c - T)^\beta$	β	0.3639(35)
susceptibility	$\chi \propto \|T - T_c\|^{-\gamma}$	γ	1.3873(85)
correlation length	$\xi \propto \|T - T_c\|^{-\nu}$	ν	0..7048(30)
specific heat	$C(T) \propto \|T - T_c\|^{-\alpha}$	α	0.1144(90)
inverse critical temperature	simple cubic	$1/T_c$	0.693035(37)
	bc cubic		0.486798(12)

The XY-model

The only exception to the just stated rule that models with $N \geq 2$ do not exhibit any finite temperature phase transition in two dimensions is the XY model which has a finite temperature Kosterlitz-Thouless transition.

This is a very special kind of phase transition. In accordance with the Mermin-Wagner theorem there is no finite magnetization at any finite temperature. However the vorticity remains finite up to a critical temperature $T_c > 0$. At T_c it jumps from the universal value $2T_c/\pi$ to 0. This model will be investigated in another of the projects.

7.7.3 Generic implementation of cluster algorithms

The cluster algorithms for many models, including the models discussed above, can be implemented in a very generic way using template constructs in C++. This generic program, as well as performance comparisons which show that the generic program is on average as fast (and sometimes even faster) than a specific optimized Fortran program will be presented at the end of the course - after you have written your versions of the algorithms.

7.8 The Wang-Landau algorithm

7.8.1 Flat histograms

While the cluster algorithms discussed in this section solve the problem of critical slowing down at second order (continuous) phase transitions they do not help at all at first order phase transitions. At first order phase transitions there is no divergence of any quantity and both the disordered and the ordered phase remain (meta)-stable throughout the transition. The most famous example is the solid/liquid transition where you can every day observe the coexistence of both (meta)-stable phases, e.g. water and ice. Water remains liquid even when cooled below the freezing temperature, until some ice crystal nucleates in the super-cooled water when it starts freezing. There is a large free energy, the surface energy of the first ice crystal, which has to be overcome before freezing sets in. This leads to macroscopically tunneling times between the two coexistent phases at the phase transition.

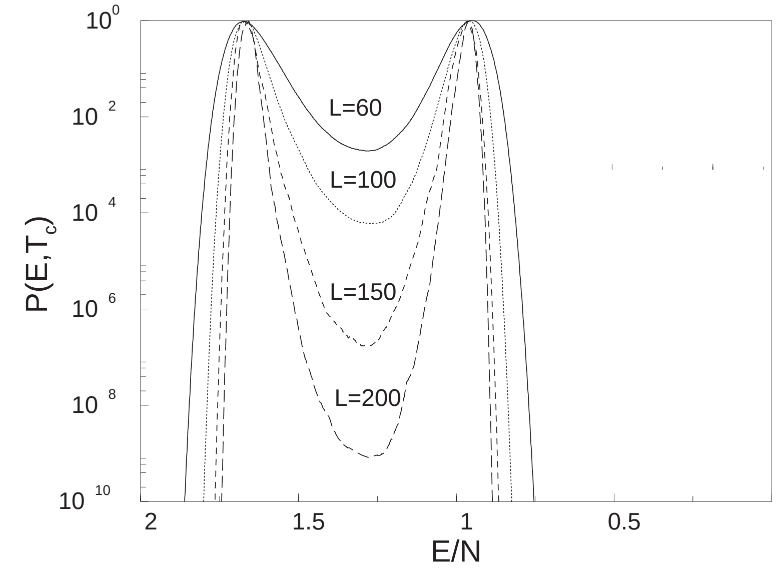


Figure 7.1: The probability $P(E,T) = \rho(E)\exp(-E/k_BT)$ of visiting a state with energy E in a $q = 10$-state Potts model at the critical temperature. The tunneling probability between the two phases (the dip between the two maxima) becomes exponentially small for large systems. This figure is taken from the paper: F. Wang and D.P. Landau, Phys. Rev. Lett. **86**, 2050 (2001).

The simplest lattice model showing such a first order thermal phase transition is a two-dimensional Potts model with large q, e.g. $q = 10$. For this model we show in Fig. 7.1 the probability $P(E,T)$ of visiting a configuration with energy E. This is:

$$P(E,T) = \rho(E)p(E) = \rho(E)e^{-E/k_BT}, \tag{7.36}$$

where the density of states $\rho(E)$ counts the number of states with energy E. At the critical temperature there are two coexisting phases, the ordered and disordered ones with different energies. In order to tunnel from one to the other one has to continuously change the energy and thus go through the probability minimum between the two peeks. This probability decreases exponentially with system size and we thus have a real problem!

The Wang-Landau algorithm is the latest in a series of attempts at solving this problem[4] It starts from the simple observation that the probability minimum vanishes if we choose $p(E) \propto 1/\rho(E)$ instead of the Boltzmann weight $p(E) \propto \exp(-E/k_BT)$:

$$P(E,T) = \rho(E)p(E) \propto \rho(E)/\rho(E) = \text{const.} \tag{7.37}$$

During our sampling we thus get a "flat histogram" in the energy.

7.8.2 Determining $\rho(E)$

The only problem now is that $\rho(E)$ is not known, since it requires a solution of the full problem. The approach by Wang and Landau is crude but simple. They start with a

[4]F. Wang and D.P. Landau, Phys. Rev. Lett. **86**, 2050 (2001); Phys. Rev. E **64**, 056101 (2001).

(very bad guess) $\rho(E) = 1$ for all energies an iteratively improve it:

- Start with $\rho(E) = 1$ and $f = e$
- Repeat
  - Reset a histogram of energies $H(E) = 0$
  - Perform simulations until a histogram of energies $H(E)$ is "flat"
    - pick a random site
    - attempt a local Metropolis update using $p(E) = 1/\rho(E)$
    - increase the histogram at the current energy E: $H(E) \leftarrow H(E) + 1$
    - increase the estimate for $\rho(E)$ at the current energy E: $\rho(E) \leftarrow \rho(E) \cdot f$
  - once $H(E)$ is "flat" (e.g. the minimum is at least 80% of the mean), reduce $f \leftarrow \sqrt{f}$
- stop once $f \approx 1 + 10^{-8}$

As you can see, only a few lines of code need to be changed in your local update algorithm for the Ising model, but a few remarks are necessary:

1. Check for flatness of the histogram not at very step but only after a reasonable number of sweeps N_{sweeps}. One sweep is defined as one attempted update per site.

2. The initial value for f needs to be carefully chosen, $f = e$ is only a rough guide. As discussed in the papers a good choice is picking the initial f such that $f^{N_{\text{sweeps}}}$ is approximately the total number of states (e.g. q^N for a q-state Potts model with N sites).

3. The flatness criterion is quite arbitrary and some research is still necessary to find the optimal choice.

4. The density of states $\rho(E)$ can become very large and easily exceed 10^{10000}. In order to obtain such large numbers the *multiplicative increase* $\rho(E) \leftarrow \rho(E) \cdot f$ is essential. A naive additive guess $\rho(E) \leftarrow \rho(E) + f$ would never be able to reach the large numbers needed.

5. Since $\rho(E)$ is so large, we only store its *logarithm*. The update step is thus $\ln \rho(E) \leftarrow \ln \rho(E) + \ln f$. The Metropolis acceptance probability will be

$$P = \min[1, \exp(\ln \rho(E_{\text{old}}) - \ln \rho(E_{\text{new}}))] \tag{7.38}$$

7.8.3 Calculating thermodynamic properties

Another advantage of the Wang-Landau algorithm is that, once we know the density of states $\rho(E)$, we can directly calculate the partition function

$$Z = \sum_c E_c e^{-E_c/k_B T} = \sum_E \rho(E) e^{-E/k_B T} \tag{7.39}$$

and the free energy

$$F = -k_B T \ln Z = -k_B T \ln \sum_E \rho(E) e^{-E/k_B T} \tag{7.40}$$

which are both not directly accessible in any other Monte Carlo algorithm. All other thermodynamic properties such as the susceptibility or the specific heat can now be calculated simply as derivatives of the free energy.

In evaluating these sums one has to be careful on how to avoid exponent overflow in the exponentials. More details on how to do that will be given in the exercises.

7.8.4 Optimized ensembles

Instead of choosing a flat histogram, any arbitrary ensemble can actually be simulated using generalizations of the Wang-Landau algorithm. And indeed, as realized by Prakash Dayal[5] the flat histogram ensemble is not yet optimal. Simon Trebst [6] has then derived an optimal ensemble and an algorithm to determine it. He will talk about this algorithm later during the semester.

7.9 The transfer matrix method

7.9.1 The Ising chain

As a last method we wish to discuss the transfer matrix method which gives exact numerical results for Ising models on infinite strips of small width W. The partition function of a periodic Ising chain of length L can be written as:

$$Z = \sum_{\{(\sigma_1,\ldots,\sigma_L)\}} \prod_{i=1}^{L} \exp(\beta J \sigma_i \sigma_{i+1}), \tag{7.41}$$

where the sum goes over all configurations $(\sigma_1, \ldots, \sigma_L)$ and periodic boundary conditions are implemented by defining $\sigma_{L+1} \equiv \sigma_1$. The partition function can be rewritten as a trace over a product of transfer matrices U

$$U = \left(\begin{array}{cc} \exp(\beta J) & \exp(-\beta J) \\ \exp(-\beta J) & \exp(\beta J) \end{array} \right). \tag{7.42}$$

Using these transfer matrices the partition function becomes:

$$Z = \mathrm{Tr} \left(\sum_{\sigma_1=\uparrow,\downarrow} \sum_{\sigma_2=\uparrow,\downarrow} \cdots \sum_{\sigma_L=\uparrow,\downarrow} \prod_{i=1}^{L} \exp(\beta J \sigma_i \sigma_{i+1}) \right) = \mathrm{Tr} U^L \tag{7.43}$$

For strips of finite width W transfer matrices of dimension 2^W can be constructed in a similar fashion, as we will see in the next section.

[5]P. Dayal *et al.*, Phys. Rev. Lett. **92**, 097201 (2004)
[6]S. Trebst, D. Huse and M. Troyer, Phys. Rev. E **70**, 046701 (2004)

The free energy density is:

$$f = -\frac{1}{\beta L}\ln Z = -\frac{1}{\beta L}\ln \mathrm{Tr} U^L \tag{7.44}$$

In the limit $L \to \infty$ only the largest eigenvalues of the transfer matrix U will be important. We label the eigenvalues λ_i, with $i = 1, \ldots, D := 2^W$, with $|\lambda_1| > |\lambda_{2|} \geq |\lambda_3| \geq \ldots$. Then the free energy density can be written as

$$\begin{aligned} f &= -\frac{1}{\beta L}\ln \mathrm{Tr} U^L = -\frac{1}{\beta L}\ln(\sum_{i=1}^{D}\lambda_i^L) \\ &= -\frac{1}{\beta L}\ln(\lambda_1^L\sum_{i=1}^{D}\lambda_i^L/\lambda_1^L) = -\frac{1}{\beta L}\left[\ln\lambda_1^L + \ln(\sum_{i=1}^{D}\lambda_i^L/\lambda_1^L)\right] \end{aligned} \tag{7.45}$$

In the limit $L \to \infty$ all λ_i^L/λ_1^L converge to zero except for the $i = 1$ term, which gives 1. Thus we obtain

$$f = -\frac{1}{\beta}\ln\lambda_1. \tag{7.46}$$

In a similar way we can show that correlations decay like $(\lambda_2/\lambda_1)^r$, which gives a correlation length

$$\xi = \frac{1}{\ln\lambda_1 - \ln\lambda_2} \tag{7.47}$$

7.9.2 Coupled Ising chains

The same procedure also works for infinitely long strips of W coupled Ising chains. In that case the transfer matrix has dimension $N = 2^W$ and is constructed just as before.

To obtain the eigenvalues of the transfer matrix we will use three tricks:

1. Representation of the dense transfer matrix U as a product of sparse matrices.
2. Calculation of the eigenvalues of the transfer matrix by iterative methods
3. Calculation of the matrix elements by multi-spin coding

The Hamilton function is

$$H = \sum_{x=-\infty}^{\infty} H^{(x)} \tag{7.48}$$

with

$$H^{(x)} = -J\sum_{y=1}^{W}\sigma_{x,y}\sigma_{x+1,y} - J\sum_{y=1}^{W-1}\sigma_{x,y}\sigma_{x,y+1} \tag{7.49}$$

The corresponding transfer matrix U is a dense matrix, with all elements non-zero. We can however write is as a product of sparse matrices:

$$U_{(x)} = \prod_{y=1}^{W-1} U^{1/2}_{(x,y)-(x,y+1)} \prod_{y=1}^{W} U_{(x,y)-(x+1,y)} \times \prod_{y=1}^{W-1} U^{1/2}_{(x,y)-(x,y+1)} \tag{7.50}$$

where the partial transfer matrices $U_{(x_1,y_1)-(x_2,y_2)}$ arise from the terms $\sigma_{x_1,y_1}\sigma_{x_2,y_2}$ in the Hamiltonian. The square roots $U^{1/2}_{(x,y)-(x,y+1)}$ are used to make the matrix U Hermitian, which will be necessary for the Lanczos algorithm.

All of these transfer matrices are sparse. The matrices $U_{(x,y)-(x,y+1)}$ are diagonal as they act only on the configuration at fixed x. The matrices $U_{(x,y)-(x+1,y)}$ contain a diagonal entry for $\sigma_{x,y} = \sigma_{x+1,y}$ and additionally an off-diagonal entry when $\sigma_{x,y} = -\sigma_{x+1,y}$.

We can thus replace a matrix-vector multiplication $U\vec{v}$, which would need $N^2 = 2^{2W}$ operations by $W-1$ multiplications with diagonal matrices $U_{(x,y)-(x,y+1)}$ and W multiplications with sparse matrices $U_{(x,y)-(x+1,y)}$. This requires only $(4W-2)N$ operations, significantly less than N^2!

In the next section we will discuss how the matrix-vector multiplications can be coded very efficiently. Finally we will explain the Lanczos algorithm which calculates the extreme eigenvalues of a sparse matrix.

7.9.3 Multi-spin coding

Multi-spin coding is an efficient technique to map a configuration of Ising spins to an array index. We interpret the bit pattern of an integer number as a spin configuration. A bit set to 1 corresponds to an up spin, a bit set to 0 corresponds to a down spin.

Let us now consider the matrices $U_{(x,y)-(x,y+1)}$. As mentioned before, they are diagonal as they do not propagate in the x-direction, but just give the Boltzmann weights of the current configuration. For a given y we loop over all N configurations $c = 0 \dots N-1$. In each configuration c we consider the bits y and $y+1$. If they are the same, the spins are aligned and the diagonal matrix element is $\exp(\beta J)$, otherwise the matrix element is $\exp(-\beta J)$.

The matrices $U_{(x,y)-(x+1,y)}$ are not much harder to compute. For each configuration c there is a diagonal element $\exp(\beta J)$. This corresponds to the case when the spin configuration does not change and the two spins at (x, y) and $(x + 1, y)$ are aligned. In addition there is one off-diagonal element $\exp(-\beta J)$, corresponding to a change in spin orientation. It connects the state c with a state c', which differs from c only by the bit y.

This bit-representation of the Ising spins thus allows a very simple end efficient way of performing these sparse matrix-vector products, as illustrated in this C++ code fragment:

```
unsigned int W=6; // the width of the strip
unsigned int N=1<<W; // 2^W = number of configurations on one crosssection

void transfer_matrix_multiply(vector<double>& v1) {

  vector<double> v2(N);

  double w1 = exp(-beta*J);
  double w2 = exp(+beta*J);
  double w1_root = exp(-0.5*beta*J);
  double w2_root = exp(+0.5*beta*J);

  // (x,y) - (x,y+1) terms
  for (int y=0; y<W-1; y++)
    for (int c=0 ; c<N; c++)
      v1[c]*=( ((c>>1) ^ c)&(1<<y) == 0 ? w1_root : w2_root);

  // (x,y) - (x+1,y) terms
  for (int y=0; y<W; y++ ) {
    for (int c=0 ; c<N; c++)
      v2[c]=v1[c]*w2+v1[c^(1<<y)]*w1;
    y++;
    if(y<W)
      for (int c=0 ; c<N; c++)
        v1[c]=v2[c]*w2+v2[c^(1<<y)]*w1;
    else
      v1=v2;
  }

  // (x,y) - (x,y+1) terms
  for (int y=0; y<W-1; y++)
    for (int c=0 ; c<N; c++)
      v1[c]*=( ((c>>1) ^ c)&(1<<y) == 0 ? w1_root : w2_root);


}
```

7.10 The Lanczos algorithm

Sparse matrices with only $O(N)$ non-zero elements are very common in scientific simulations. We have already encountered them in the winter semester when we discretized partial differential equations. Now we have reduced the transfer matrix of the Ising model to a sparse matrix product. We will later see that also the quantum mechanical Hamilton operators in lattice models are sparse.

The importance of sparsity becomes obvious when considering the cost of matrix operations as listed in table 7.4. For large N the sparsity leads to memory and time savings of several orders of magnitude.

Here we will discuss the iterative calculation of a few of the extreme eigenvalues of a matrix by the Lanczos algorithm. Similar methods can be used to solve sparse linear systems of equations.

To motivate the Lanczos algorithms we will first take a look at the power method for a matrix A. Starting from a random initial vector u_1 we calculate the sequence

$$u_{n+1} = \frac{Au_n}{||Au_n||}, \tag{7.51}$$

Table 7.4: Time and memory complexity for operations on sparse and dense $N \times N$ matrices

operation	time	memory
storage		
dense matrix	—	N^2
sparse matrix	—	$\mathrm{O}(N)$
matrix-vector multiplication		
dense matrix	$\mathrm{O}(N^2)$	$\mathrm{O}(N^2)$
sparse matrix	$\mathrm{O}(N)$	$\mathrm{O}(N)$
matrix-matrix multiplication		
dense matrix	$\mathrm{O}(N^{\ln 7/\ln 2})$	$\mathrm{O}(N^2)$
sparse matrix	$\mathrm{O}(N)\ldots\mathrm{O}(N^2)$	$\mathrm{O}(N)\ldots\mathrm{O}(N^2)$
all eigen values and vectors		
dense matrix	$\mathrm{O}(N^3)$	$\mathrm{O}(N^2)$
sparse matrix (iterative)	$\mathrm{O}(N^2)$	$\mathrm{O}(N^2)$
some eigen values and vectors		
dense matrix (iterative)	$\mathrm{O}(N^2)$	$\mathrm{O}(N^2)$
sparse matrix (iterative)	$\mathrm{O}(N)$	$\mathrm{O}(N)$

which converges to the eigenvector of the largest eigenvalue of the matrix A. The Lanczos algorithm optimizes this crude power method.

Lanczos iterations

The Lanczos algorithm builds a basis $\{v_1, v_2, \ldots, v_M\}$ for the Krylov-subspace $K_M = span\{u_1, u_2, \ldots, u_M\}$, which is constructed by M iterations of equation (7.51). This is done by the following iterations:

$$\beta_{n+1} v_{n+1} = A v_n - \alpha_n v_n - \beta_n v_{n-1}, \tag{7.52}$$

where

$$\alpha_n = v_n^\dagger A v_n, \qquad \beta_n = |v_n^\dagger A v_{n-1}|. \tag{7.53}$$

As the orthogonality condition

$$v_i^\dagger v_j = \delta_{ij} \tag{7.54}$$

does not determine the phases of the basis vectors, the β_i can be chosen to be real and positive. As can be seen, we only need to keep three vectors of size N in memory, which makes the Lanczos algorithm very efficient, when compared to dense matrix eigensolvers which require storage of order N^2.

In the Krylov basis the matrix A is tridiagonal

$$T^{(n)} := \begin{bmatrix} \alpha_1 & \beta_2 & 0 & \cdots & 0 \\ \beta_2 & \alpha_2 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \beta_n \\ 0 & \cdots & 0 & \beta_n & \alpha_n \end{bmatrix}. \tag{7.55}$$

The eigenvalues $\{\tau_1, \ldots, \tau_M\}$ of T are good approximations of the eigenvalues of A. The extreme eigenvalues converge very fast. Thus $M \ll N$ iterations are sufficient to obtain the extreme eigenvalues.

Eigenvectors

It is no problem to compute the eigenvectors of T. They are however given in the Krylov basis $\{v_1, v_2, \ldots, v_M\}$. To obtain the eigenvectors in the original basis we need to perform a basis transformation.

Due to memory constraints we usually do not store all the v_i, but only the last three vectors. To transform the eigenvector to the original basis we have to do the Lanczos iterations a second time. Starting from the same initial vector v_1 we construct the vectors v_i iteratively and perform the basis transformation as we go along.

Roundoff errors and ghosts

In exact arithmetic the vectors $\{v_i\}$ are orthogonal and the Lanczos iterations stop after at most $N-1$ steps. The eigenvalues of T are then the exact eigenvalues of A.

Roundoff errors in finite precision cause a loss of orthogonality. There are two ways to deal with that:

- Reorthogonalization of the vectors after every step. This requires storing all of the vectors $\{v_i\}$ and is memory intensive.
- Control of the effects of roundoff.

We will discuss the second solution as it is faster and needs less memory. The main effect of roundoff errors is that the matrix T contains extra spurious eigenvalues, called "ghosts". These ghosts are not real eigenvalues of A. However they converge towards real eigenvalues of A over time and increase their multiplicities.

A simple criterion distinguishes ghosts from real eigenvalues. Ghosts are caused by roundoff errors. Thus they do not depend on on the starting vector v_1. As a consequence these ghosts are also eigenvalues of the matrix $\tilde{T}$, which can be obtained from T by deleting the first row and column:

$$\tilde{T}^{(n)} := \begin{bmatrix} \alpha_2 & \beta_3 & 0 & \cdots & 0 \\ \beta_3 & \alpha_3 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \beta_n \\ 0 & \cdots & 0 & \beta_n & \alpha_n \end{bmatrix}. \tag{7.56}$$

From these arguments we derive the following heuristic criterion to distinguish ghosts from real eigenvalues:

- All multiple eigenvalues are real, but their multiplicities might be too large.
- All single eigenvalues of T which are *not* eigenvalues of $\tilde{T}$ are also real.

Numerically stable and efficient implementations of the Lanczos algorithm can be obtained from netlib. As usual, do not start coding your own algorithm but use existing optimal implementations.

7.11 Renormalization group methods for classical spin systems

Just as in the percolation problem renormalization group methods can also be applied to spin systems. A block of b spins is replaced by a single spin, and the interaction is renormalized by requiring that physical expectation values are invariant under the transformation.

Again, Monte Carlo renormalization group can be used for larger block sizes b. As we have already learned the basics of renormalization in the percolation problem we will not discuss any details here. Interested students are referred to the text book by Tao Pang and to references therein.

Chapter 8

The quantum one-body problem

8.1 The time-independent one-dimensional Schrödinger equation

We will start the discussion of quantum problems with the time-indepent one-dimensional Schrödinger equation for a particle with mass m in a Potential $V(x)$. For this problem the time-dependent Schrödinger equation

$$i\hbar\frac{\partial\psi}{\partial t} = -\frac{\hbar^2}{2m}\frac{\partial^2\psi}{\partial x^2} + V(x)\psi, \tag{8.1}$$

can be simplified to an ordinary differential equation using the ansatz $\psi(x,t) = \psi(x)\exp(-iEt)$

$$E\psi = -\frac{\hbar^2}{2m}\frac{\partial^2\psi}{\partial x^2} + V(x)\psi. \tag{8.2}$$

8.1.1 The Numerov algorithm

After rewriting this second order differential equation to a coupled system of two first order differential equations, any ODE solver such as the Runge-Kutta method could be applied, but there exist better methods.

For the special form

$$\psi''(x) + k(x)\psi(x) = 0, \tag{8.3}$$

of the Schrödinger equation, with $k(x) = 2mV(x)/\hbar^2$ we can derive the Numerov algorithm by starting from the Taylor expansion of $\psi_n = \psi(n\Delta x)$:

$$\psi_{n\pm1} = \psi_n \pm \Delta x\psi_n' + \frac{\Delta x^2}{2}\psi_n'' \pm \frac{\Delta x^3}{6}\psi_n^{(3)} + \frac{\Delta x^4}{24}\psi_n^{(4)} \pm \frac{\Delta x^5}{120}\psi_n^{(5)} + \mathrm{O}(\Delta x^6) \tag{8.4}$$

Adding ψ_{n+} and ψ_{n-} we obtain

$$\psi_{n+1} + \psi_{n-1} = 2\psi_n + (\Delta x)^2\psi_n''\frac{(\Delta x)^4}{12}\psi_n^{(4)}. \tag{8.5}$$

Replacing the fourth derivatives by a finite difference second derivative of the second derivatives

$$\psi_n^{(4)} = \frac{\psi_{n+1}'' + \psi_{n-1}'' - 2\psi_n''}{\Delta x^2} \tag{8.6}$$

and substituting $-k(x)\psi(x)$ for $\psi''(x)$ we obtain the Numerov algorithm

$$\begin{aligned}\left(1+\frac{(\Delta x)^2}{12}k_{n+1}\right)\psi_{n+1} = & \quad 2\left(1-\frac{5(\Delta x)^2}{12}k_n\right)\psi_n \\ & -\left(1+\frac{(\Delta x)^2}{12}k_{n-1}\right)\psi_{n-1}+\mathrm{O}(\Delta x^6), \qquad (8.7)\end{aligned}$$

which is locally of sixth order!

Initial values

To start the Numerov algorithm we need the wave function not just at one but at two initial values and will now present several ways to obtain these.

For potentials $V(x)$ with reflection symmetry $V(x) = V(-x)$ the wave functions need to be either even $\psi(x) = \psi(-x)$ or odd $\psi(x) = -\psi(-x)$ under reflection, which can be used to find initial values:

- For the even solution we use a half-integer mesh with mesh points $x_{n+1/2} = (n + 1/2)\Delta x$ and pick intiial values $\psi_(x_{-1/2}) = \psi(x_{1/2}) = 1$.
- For the odd solution we know that $\psi(0) = -\psi(0)$ and hence $\psi(0) = 0$, specifying the first starting value. Using an integer mesh with mesh points $x_n = n\Delta x$ we pick $\psi_(x_1) = 1$ as the second starting value.

In general potentials we need to use other approaches. If the potentials vanishes for large distances: $V(x) = 0$ for $|x| \geq a$ we can use the exact solution of the Schrdinger equation at large distances to define starting points, e.g.

$$\psi(-a) = 1 \qquad (8.8)$$

$$\psi(-a-\Delta x) = \exp(-\Delta x\sqrt{2mE/\hbar}). \qquad (8.9)$$

Finally, if the potential never vanishes we need to begin with a single starting value $\psi(x_0)$ and obtain the second starting value $\psi(x_1)$ by performing an integration over the first time step $\Delta\tau$ with an Euler or Runge-Kutta algorithm.

8.1.2 The one-dimensional scattering problem

The scattering problem is the numerically easiest quantum problem since solutions exist for all energies $E > 0$, if the potential vanishes at large distances ($V(x) \to 0$ for $|x| \to \infty$. The solution becomes particularly simple if the potential is nonzero only on a finite interval $[0, a]$. For a particle approaching the potential barrier from the left ($x < 0$) we can make the following ansatz for the free propagation when $x < 0$:

$$\psi_L(x) = A\exp(-ikx) + B\exp(ikx) \qquad (8.10)$$

where A is the amplitude of the incoming wave and B the amplitude of the reflected wave. On the right hand side, once the particle has left the region of finite potential ($x > a$), we can again make a free propagation ansatz,

$$\psi_R(x) = C\exp(-ikx) \qquad (8.11)$$

The coefficients A, B and C have to be determined self-consistently by matching to a numerical solution of the Schrödinger equation in the interval $[0, a]$. This is best done in the following way:

- Set $C = 1$ and use the two points a and $a + \Delta x$ as starting points for a Numerov integration.
- Integrate the Schrödinger equation numerically – backwards in space,from a to 0 – using the Numerov algorithm.
- Match the numerical solution of the Schrödinger equation for $x < 0$ to the free propagation ansatz (8.10) to determine A and B.

Once A and B have been determined the reflection and transmission probabilities R and T are given by

$$R = |B|^2/|A|^2 \tag{8.12}$$
$$T = 1/|A|^2 \tag{8.13}$$

8.1.3 Bound states and solution of the eigenvalue problem

While there exist scattering states for all energies $E > 0$, bound states solutions of the Schrödinger equation with $E < 0$ exist only for discrete energy eigenvalues. Integrating the Schrödinger equation from $-\infty$ to $+\infty$ the solution will diverge to $\pm\infty$ as $x \to \infty$ for almost all values. These functions cannot be normalized and thus do not constitute solutions to the Schrödinger equation. Only for some special eigenvalues E, will the solution go to zero as $x \to \infty/$

A simple eigensolver can be implemented using the following shooting method, where we again will assume that the potential is zero outside an interval $[0, a]$:

- Start with ann initial guess E
- Integrate the Schrödinger equation for $\psi_E(x)$ from $x = 0$ to $x_f \gg a$ and determine the value $\psi_E(x_f)$
- use a root solver, such as a bisection method, to look for an energy E with $\psi_E(x_f) \approx 0$

This algorithm is not ideal since the divergence of the wave function for $x \pm \infty$ will cause roundoff error to proliferate.

A better solution is to integrate the Schrödinger equation from both sides towards the center:

- We search for a point b with $V(b) = E$
- Starting from $x = 0$ we integrate the left hand side solution $\psi_L(x)$ to a chosen point b and obtain $\psi_L(b)$ and a numerical estimate for $\psi'_L(b) = (\psi_L(b) - \psi_L(b - \Delta x))/\Delta x$.
- Starting from $x = a$ we integrate the right hand solution $\psi_R(x)$ down to the same point b and obtain $\psi_R(b)$ and a numerical estimate for $\psi'_R(b) = (\psi_R(b + \Delta x) - \psi_R(b))/\Delta x$.

- At the point b the wave functions and their first two derivatives have to match, since solutions to the Schrödinger equation have to be twice continuously differentiable. Keeping in mind that we can multiply the wave functions by an arbitrary factor we obtain the conditions

$$\begin{aligned} \psi_L(b) &= \alpha\psi_R(b) && (8.14)\\ \psi'_L(b) &= \alpha\psi''_R(b) && (8.15)\\ \psi''_L(b) &= \alpha\psi''_R(b) && (8.16) \end{aligned}$$

  The last condition is automatically fulfilled since by the choice $V(b) = E$ the Schrödinger equation at b reduces to $\psi''(b) = 0$. The first two conditions can be combined to the condition that the logarithmic derivatives vanish:

$$\frac{d\log\psi_L}{dx}|_{x=b} = \frac{\psi'_L(b)}{\psi_L(b)} = \frac{\psi'_R(b)}{\psi_R(b)} = \frac{d\log\psi_R}{dx}|_{x=b} \tag{8.17}$$

- This last equation has to be solved for in a shooting method, e.g. using a bisection algorithm

8.2 The time-independent Schrödinger equation in higher dimensions

The time independent Schrödinger equation in more than one dimension is a partial differential equation and cannot, in general, be solved by a simple ODE solver such as the Numerov algorithm. Before employing a PDE solver we should thus always first try to reduce the problem to a one-dimensional problem. This can be done if the problem factorizes.

A first example is a three-dimensional Schrdinger equation in a cubic box with potential $V(\vec{r}) = V(x)V(y)V(z)$ with $\vec{r} = (x, y, z)$. Using the product ansatz

$$\psi(\vec{r}) = \psi_x(x)\psi_y(y)\psi_z(z) \tag{8.18}$$

the PDE factorizes into three ODEs which can be solved as above.

Another famous trick is possible for spherically symmetric potentials with $V(\vec{r}) = V(|\vec{r}|)$ where an ansatz using spherical harmonics

$$\psi_{l,m}(\vec{r}) = \psi l, m(r, \theta, \phi) = \frac{u(r)}{r} Y_{lm}(\theta, \phi) \tag{8.19}$$

can be used to reduce the three-dimensional Schrödinger equation to a one-dimensional one for the radial wave function $u(r)$:

$$\left[-\frac{\hbar^2}{2m}\frac{d^2}{dr^2} + \frac{\hbar^2 l(l+1)}{2mr^2} + V(r)\right] u(r) = Eu(r) \tag{8.20}$$

in the interval $[0, \infty[$. Given the singular character of the potential for $r \to 0$, a numerical integration should start at large distances r and integrate towards $r = 0$, so that the largest errors are accumulated only at the last steps of the integration.

8.2.1 Variational solutions using a finite basis set

In the case of general potentials, or for more than two particles, it will not be possible to reduce the Schrödinger equation to a one-dimensional problem and we need to employ a PDE solver. One approach will again be to discretize the Schrödinger equation on a discrete mesh using a finite difference approximation. A better solution is to expand the wave functions in terms of a finite set of basis functions

$$|\phi\rangle = \sum_{i=1}^{N} a_i |u_i\rangle, \tag{8.21}$$

as we did in the finite element method in section 3.6.

To estimate the ground state energy we want to minimize the energy of the variational wave function

$$E^* = \frac{\langle\phi|H|\phi\rangle}{\langle\phi|\phi\rangle}. \tag{8.22}$$

Keep in mind that, since we only chose a finite basis set $\{|u_i\rangle\}$ the variational estimate E^* will always be larger than the true ground state energy E_0, but will converge towards E_0 as the size of the basis set is increased, e.g. by reducing the mesh size in a finite element basis.

To perform the minimization we denote by

$$H_{ij} = \langle u_i|H|u_j\rangle = \int d\vec{r} u_i(\vec{r})^* \left(-\frac{\hbar^2}{2m}\nabla^2 + V\right) u_j(\vec{r}) \tag{8.23}$$

the matrix elements of the Hamilton operator H and by

$$S_{ij} = \langle u_i|u_j\rangle = \int d\vec{r} u_i(\vec{r})^* u_j(\vec{r}) \tag{8.24}$$

the overlap matrix. Note that for an orthogonal basis set, S_{ij} is the identity matrix δ_{ij}. Minimizing equation (8.22) we obtain a generalized eigenvalue problem

$$\sum_j H_{ij} a_j = E \sum_k S_{ik} a_k. \tag{8.25}$$

or in a compact notation with $\vec{a} = (a_1, \ldots, a_N)$

$$H\vec{a} = ES\vec{a}. \tag{8.26}$$

If the basis set is orthogonal this reduces to an ordinary eigenvalue problem and we can use the Lanczos algorithm.

In the general case we have to find orthogonal matrices U such that $U^T SU$ is the identity matrix. Introducing a new vector $\vec{b} = U^{-1}\vec{a}$. we can then rearrange the problem into

$$\begin{aligned} H\vec{a} &= S\vec{a} \\ HU\vec{b} &= ESU\vec{b} \\ HU\vec{b} &= ESU\vec{b} \\ U^T HU\vec{b} &= EU^T SU\vec{b} = E\vec{b} \end{aligned} \tag{8.27}$$

and we end up with a standard eigenvalue problem for $U^T HU$. Mathematica and LAPACK both contain eigensolvers for such generalized eigenvalue problems.

The final issue is the choice of basis functions. While a general finite element basis can be used it iss advantageous to make use of known solutions to similar problem as we will illustrate in the case of an anharmonic oscillator with Hamilton operator

$$\begin{aligned} H &= H_0 + \lambda q^4 \\ H_0 &= \frac{1}{2}(p^2 + q^2), \end{aligned} \tag{8.28}$$

where the momentum operator is $p = i\hbar\frac{\partial}{\partial q}$. The eigenstates $|n\rangle$ and eigenvalues $\epsilon_n = (n + 1/2)\omega_0$ of H_0 are be known from the basic quantum mechanics lectures. In real space the eigenstates are given by

$$\phi_n(q) = \frac{1}{\sqrt{2^n n! \sqrt{\pi}}} \exp\left(-\frac{1}{2}q^2\right) H_n(q), \tag{8.29}$$

where the H_n are the Hermite polynomials. Using these eigenstates as a basis set, the operator H_0 becomes a diagonal matrix. The position operator becomes

$$q = \frac{1}{\sqrt{2}}(a^\dagger + a), \tag{8.30}$$

where the raising and lowering operators $a^\dagger$ and a only have the following nonzero matrix elements:

$$\langle n+1|a^\dagger|n\rangle = \langle n|a|n+1\rangle = \sqrt{n+1}. \tag{8.31}$$

The matrix representation of the anharmonic term λq^4 is a banded matrix. Aftwer truncation of the basis set to a finite number of states N, a sparse eigensolver such as the Lanczos algorithm can again be used to calculate the spectrum. Note that since we use the orthonormal eigenstates of H_0 as basis elements, the overlap matrix S here is the identity matrix and we have to deal only with a standard eigenvalue problem. A solution to this problem is provided in a Mathematica notebook on the web page.

8.3 The time-dependent Schrödinger equation

Finally we will reintroduce the time dependence to study dynamics in non-stationary quantum systems.

8.3.1 Spectral methods

By introducing a basis and solving for the complete spectrum of energy eigenstates we can directly dolve the time-dependent problem in the case of a stationary Hamiltonian. This is a consequence of the linearity of the Schrödinger equation.

To calculate the time evolution of a state $|\psi(t_0)\rangle$ from time t_0 to t we first solve the stationary eigenvalue problem $H|\phi\rangle = E|\phi\rangle$ and calculate the eigenvectors $|\phi_n\rangle$ and eigenvaluesϵ_n. Next we represent the initial wave function $|\psi\rangle$ by a spectral decomposition

$$|\psi(t_0)\rangle = \sum_n c_n |\phi_n\rangle. \tag{8.32}$$

Since each of the $|\phi_n\rangle$ is an eigenvector of H, the time evolution $e^{-i\hbar H(t-t_0)}$ is trivial and we obtain at time t:

$$|\psi(t)\rangle = \sum_n c_n e^{-i\hbar\epsilon_n(t-t_0)} |\phi_n\rangle. \tag{8.33}$$

8.3.2 Direct numerical integration

If the number of basis states is too large to perform a complete diagonalization of the Hamiltonian, or if the Hamiltonian changes over time we need to perform a direct integration of the Schrödinger equation.

The main novelty, compared to the integration of classical wave equations is that the exact quantum mechanical time evolution conserves the normalization

$$\int |\psi(x,t)|^2 dx = 1 \tag{8.34}$$

of the wave function and the numerical algorithm should also have this property: the approximate time evolution needs to be unitary.

We first approximate the time evolution

$$\psi(x, t + \Delta t) = e^{-i\hbar H \Delta t} \psi(x, t). \tag{8.35}$$

by a forward Euler approximation

$$e^{-i\hbar H \Delta t} \approx 1 - i\hbar H \Delta t + \mathrm{O}(\Delta t^2). \tag{8.36}$$

This is neither unitary nor stable and we need a better integrator. The simplest stable and unitary integrator can be obtained by the following reformulation

$$e^{-i\hbar H \Delta t} = \left(e^{i\hbar H \Delta t/2}\right)^{-1} e^{-i\hbar H \Delta t/2} \approx \left(1 + i\hbar H \frac{\Delta t}{2}\right)^{-1} \left(1 - i\hbar H \frac{\Delta t}{2}\right) + \mathrm{O}(\Delta t^3). \tag{8.37}$$

This gives an algorithm

$$\psi(x, t + \Delta t) = \left(1 + i\hbar H \frac{\Delta t}{2}\right)^{-1} \left(1 - i\hbar H \frac{\Delta t}{2}\right) \psi(x, t) \tag{8.38}$$

or equivalently

$$\left(1 + i\hbar H \frac{\Delta t}{2}\right) \psi(x, t + \Delta t) = \left(1 - i\hbar H \frac{\Delta t}{2}\right) \psi(x, t). \tag{8.39}$$

After introducing a basis as above, we realize that we need to solve a linear system of equations. For one-dimensional problems the matrix representation of H is often tridiagonal and a tridiagonal solver can be used.

In higher dimensions the matrix H will no longer be simply tridiagonal but still very sparse and we can use iterative algorithms, similar to the Lanczos algorithm for the eigenvalue problem. For details about these algorithms we refer to the nice summary at `http://mathworld.wolfram.com/topics/Templates.html` and especially the biconjugate gradient (BiCG) algorithm. Implementations of this algorithm are available, e.g. in the Iterative Template Library (ITL).

Chapter 9

The quantum N body problem: quantum chemistry methods

After treating classical problems in the first chapters, we will turn to quantum problems in the last two chapters. In the winter semester we saw that the solution of the classical one-body problem reduced to an ordinary differential equation, while that of quantum one-body problem was a partial differential equation.

Many body problems are intrinsically harder to simulate. In the last chapter we saw how the computational complexity of the classical N-body problem can be reduced from $O(N^2)$ to $O(N \ln N)$, thus making even large systems with 10^8 particles accessible to simulations.

The quantum many body problem is much harder than the classical one. While the dimension of the classical phase space grew linearly with the number of particles, that of the quantum problem grows exponentially. Thus the complexity is usually $O(\exp(N))$. Exponential algorithms are the nightmare of any computational scientist, but they naturally appear in quantum many body problems. In this chapter we will discuss approximations used in quantum chemistry that reduce the problem to an polynomial one, typically scaling like $O(N^4)$. These methods map the problem to a single-particle problem and work only as long as correlations between electrons are weak. In the next chapter we will thus discuss exact methods - without any approximation - which are needed for the simulation of strongly correlated systems, such as high temperature superconductors, heavy electron materials and quantum magnets.

9.1 Basis functions

All approaches to solutions of the quantum mechanical N-body problem start by choosing a suitable basis set for the wave functions. The Schrödinger equation then maps to an eigenvalue equations, which is solved using standard linear algebra methods. Before attempting to solve the many body problem we will discuss basis sets for single particle wave functions.

9.1.1 The electron gas

For the free electron gas with Hamilton operator

$$H = -\sum_{i=1}^{N} \frac{\hbar^2}{2m}\nabla^2 + e^2 \sum_{i<j} v_{ee}(\vec{r}_i, \vec{r}_j) \tag{9.1}$$

$$v_{ee}(\vec{r}, \vec{r}\,') = \frac{1}{|\vec{r} - \vec{r}\,'|} \tag{9.2}$$

the ideal choice for basis functions are plane waves

$$\psi_{\vec{k}}(\vec{r}) = \exp(-i\vec{k}\vec{r}). \tag{9.3}$$

At low temperatures the electron gas forms a Wigner crystal. Then a better choice of basis functions are eigenfunctions of harmonic oscillators centered around the classical equilibrium positions.

9.1.2 Electronic structure of molecules and atoms

The Hamilton operator for molecules and atoms contains extra terms due to the atomic nuclei:

$$H = \sum_{i=1}^{N} \left(-\frac{\hbar^2}{2m}\nabla^2 + V(\vec{r}_i) \right) + e^2 \sum_{i<j} v_{ee}(\vec{r}_i, \vec{r}_j) \tag{9.4}$$

where the potential of the M atomic nuclei with charges Z_i at the locations $\vec{R}_i$ is given by

$$V(\vec{r}) = -e^2 \sum_{i=1}^{M} \frac{Z_i}{|\vec{R}_i - \vec{r}|}. \tag{9.5}$$

Here we use the Born-Oppenheimer approximation and consider the nuclei as stationary classical particles. This approximation is valid since the nuclei are many orders of magnitude heavier than the electrons. The Car-Parinello method for molecular dynamics, which we will discuss later, moves the nuclei classically according to electronic forces that are calculated quantum mechanically.

As single particle basis functions we choose L atomic basis functions f_i, centered on the nuclei. In general these functions are not orthogonal. Their scalar products define a matrix

$$S_{ij} = \int d^3\vec{r} f_i^*(\vec{r}) f_j(\vec{r}), \tag{9.6}$$

which is in general *not* the identity matrix. The associated annihilation operators $a_{i\sigma}$ are defined formally as scalar products

$$a_{i\sigma} = \sum_j (S^{-1})_{ij} \int d^3\vec{r} f_j^*(\vec{r}) \psi_\sigma(\vec{r}), \tag{9.7}$$

where $\sigma = \uparrow, \downarrow$ is the spin index and $\psi_\sigma(\vec{r})$ the fermion field operator.

The non-orthogonality causes the commutation relations of these operators to differ from those of normal fermion creation- and annihilation operators:

$$\begin{aligned} \{a_{i\sigma}^\dagger, a_{j\sigma'}\} &= \delta_{\sigma\sigma'} (S^{-1})_{ij} \\ \{a_{i\sigma}, a_{j\sigma'}\} = \{a_{i\sigma}^\dagger, a_{j\sigma'}^\dagger\} &= 0. \end{aligned} \tag{9.8}$$

Due to the non-orthogonality the adjoint $a^\dagger_{i\sigma}$ does *not* create a state with wave function f_i. This is done by the operator $\hat{a}^\dagger_{i\sigma}$, defined through:

$$\hat{a}^\dagger_{i\sigma} = \sum_j S_{ji} a^\dagger_{i\sigma}, \tag{9.9}$$

which has the following simple commutation relation with $a_{j\sigma}$:

$$\{\hat{a}^\dagger_{i\sigma}, a_{j\sigma}\} = \delta_{ij}. \tag{9.10}$$

The commutation relations of the $\hat{a}^\dagger_{i\sigma}$ and the $\hat{a}_{j\sigma'}$ are:

$$\begin{aligned} \{\hat{a}^\dagger_{i\sigma} \hat{a}_{j\sigma'}\} &= \delta_{\sigma\sigma'} S_{ij} \\ \{\hat{a}_{i\sigma}, \hat{a}_{j\sigma'}\} = \{\hat{a}^\dagger_{i\sigma}, \hat{a}^\dagger_{j\sigma'}\} &= 0. \end{aligned} \tag{9.11}$$

When performing calculations with these basis functions extra care must be taken to account for this non-orthogonality of the basis.

In this basis the Hamilton operator (9.4) is written as

$$H = \sum_{ij\sigma} t_{ij} a^\dagger_{i\sigma} a_{j\sigma} + \frac{1}{2} \sum_{ijkl\sigma\sigma'} V_{ijkl} a^\dagger_{i\sigma} a^\dagger_{k\sigma'} a_{l\sigma'} a_{j\sigma}. \tag{9.12}$$

The matrix elements are

$$t_{ij} = \int d^3\vec{r} f_i^*(\vec{r}) \left(\frac{\hbar^2}{2m} \nabla^2 + V(\vec{r}_i) \right) f_j(\vec{r}) \tag{9.13}$$

$$V_{ijkl} = e^2 \int d^3\vec{r} \int d^3\vec{r}' f_i^*(\vec{r}) f_j(\vec{r}) \frac{1}{|\vec{r} - \vec{r}'|} f_k^*(\vec{r}') f_l(\vec{r}') \tag{9.14}$$

Which functions should be used as basis functions? Slater proposed the **Slater-Type-Orbitals** (STO):

$$f_i(r, \theta, \phi) \propto r^{n-1} e^{-\zeta_i r} Y_{lm}(\theta, \phi). \tag{9.15}$$

The values ζ_i are optimized so that the eigenstates of isolated atoms are reproduced as good as possible. The main advantage of STOs is that they exhibit the correct asymptotic behavior for small distances r. Their main disadvantage is in the evaluation of the matrix elements in equation (9.14).

The **Gaussian-Type-Orbitals** (GTO)

$$f_i(\vec{r}) \propto x^l y^m z^n e^{-\zeta_i r^2} \tag{9.16}$$

simplify the evaluation of matrix elements, as products of Gaussian functions centered at two different nuclei are again Gaussian functions and can be integrated easily. In addition, the term $\frac{1}{|\vec{r}-\vec{r}'|}$ can also be rewritten as an integral over a Gaussian function

$$\frac{1}{|\vec{r} - \vec{r}'|} = \frac{2}{\sqrt{\pi}} \int_0^\infty dt e^{-t^2(\vec{r}-\vec{r}')^2}. \tag{9.17}$$

Then the six-dimensional integral (9.14) is changed to a seven-dimensional one, albeit with purely Gaussian terms and this can be carried out analytically as we have seen in the exercises.

As there are $O(L^4)$ integrals of the type (9.14), quantum chemistry calculations typically scale as $O(N^4)$. Modern methods

9.2 Pseudo-potentials

The electrons in inner, fully occupied shells do not contribute in the chemical bindings. To simplify the calculations they can be replaced by pseudo-potentials, modeling the inner shells. Only the outer shells (including the valence shells) are then modeled using basis functions. The pseudo-potentials are chosen such that calculations for isolated atoms are as accurate as possible.

9.3 Hartree Fock

The Hartree-Fock approximation is based on the assumption of independent electrons. It starts from an ansatz for the N-particle wave function as a Slater determinant of N single-particle wave functions:

$$\Phi(\vec{r}_1, \sigma_1; \ldots; \vec{r}_N, \sigma_N) = \frac{1}{\sqrt{N}} \begin{vmatrix} \phi_1(\vec{r}_1, \sigma_1) & \cdots & \phi_N(\vec{r}_1, \sigma_1) \\ \vdots & & \vdots \\ \phi_1(\vec{r}_N, \sigma_N) & \cdots & \phi_N(\vec{r}_N, \sigma_N) \end{vmatrix}. \tag{9.18}$$

The single particle wave functions ϕ_μ are orthogonal. For numerical calculations a finite basis has to be introduced, as discussed in the previous section. Quantum chemists distinguish between the self-consistent-field (SCF) approximation in a finite basis set and the Hartree-Fock (HF) limit, working in a complete basis. In physics both are known as Hartree-Fock approximation.

The functions ϕ_μ are chosen so that the energy The ground state energy is:

$$\langle\Phi|H|\Phi\rangle = \sum_{\mu=1}^{N} \langle\phi_\mu|\frac{\hbar^2}{2m}\nabla^2 + V|\phi_\mu\rangle + \frac{1}{2}e^2 \sum_{\nu,\mu=1}^{N} \left(\langle\phi_\mu\phi_\nu|v_{ee}|\phi_\mu\phi_\nu\rangle - \langle\phi_\mu\phi_\nu|v_{ee}|\phi_\nu\phi_\mu\rangle\right). \tag{9.19}$$

is minimized, constrained by the normalization of the wave functions. Using Lagrange multipliers ϵ_μ we obtain the variational condition

$$\delta\left(\langle\Phi|H|\Phi\rangle - \sum_\mu \epsilon_\mu \langle\phi_\mu|\phi_\mu\rangle\right) = 0. \tag{9.20}$$

Performing the variation we end up with the Hartree-Fock equation

$$F|\phi_\mu\rangle = \epsilon_\mu|\phi_\mu\rangle, \tag{9.21}$$

where the matrix elements $f_{\nu\mu}$ of the Fock matrix F are given by

$$f_{\nu\mu} = \langle\phi_\nu|\frac{\hbar^2}{2m}\nabla^2 + V|\phi_\mu\rangle + e^2 \sum_{\tau=1}^{N} \left(\langle\phi_\nu\phi_\tau|v_{ee}|\phi_\mu\phi_\tau\rangle - \langle\phi_\nu\phi_\tau|v_{ee}|\phi_\tau\phi_\mu\rangle\right). \tag{9.22}$$

The Hartree-Fock equation looks like a one-particle Schrödinger equation. However, the potential depends on the solution. The equation is used iteratively, always using the new solution for the potential, until convergence to a fixed point is achieved.

The eigenvalues ϵ_μ of F do not directly correspond to energies of the orbitals, as the Fock operator counts the v-terms twice. Thus we obtain the total ground state energy from the Fock operator eigenvalues by subtracting the double counted part:

$$E_0 = \sum_{\mu=1}^{N} \epsilon_\mu - \frac{1}{2} e^2 \sum_{\nu,\mu=1}^{N} \left(\langle \phi_\mu \phi_\nu | v_{ee} | \phi_\mu \phi_\nu \rangle - \langle \phi_\mu \phi_\nu | v_{ee} | \phi_\nu \phi_\mu \rangle \right). \tag{9.23}$$

How do we calcuate the functions ϕ_n in our finite basis sets, introduced in the previous section? To simplify the discussion we assume closed-shell conditions, where each orbital is occupied by both an electron with spin $\uparrow$ and one with spin $\downarrow$.

We start by writing the Hartree Fock wave function (9.18) is second quantized form:

$$|\Phi\rangle = \prod_{\mu,\sigma} c^\dagger_{\mu\sigma} |0\rangle, \tag{9.24}$$

where $|\phi_{\mu\sigma}\rangle$ creates an electron in the orbital $\phi_\mu(\mathbf{r}, \sigma)$. As these wave functions are orthogonal the $c^\dagger_{\mu\sigma}$ satsify the usual fermion anticommutation relations.

Next we expand the $c^\dagger_{\mu\sigma}$ in terms of our finite basis set:

$$c^\dagger_{\mu\sigma} = \sum_{n=1}^{L} d_{\mu n} \hat{a}^\dagger_{n\sigma} \tag{9.25}$$

and find that

$$a_{j\sigma} |\Phi\rangle = a_{j\sigma} \prod_{\mu,\sigma'} c^\dagger_{\mu\sigma'} |0\rangle = \sum_\nu d_{\nu j} \prod_{\mu\sigma' \neq \nu\sigma} c^\dagger_{\mu\sigma'} |0\rangle. \tag{9.26}$$

In order to evaluate the matrix elements $\langle \Phi | H | \Phi \rangle$ of the Hamiltonian (9.12) we introduce the bond-order matrix

$$P_{ij} = \sum_\sigma \langle \Phi | a^\dagger_{i\sigma} a_{j\sigma} | \Phi \rangle = 2 \sum_\nu d^*_{\nu i} d_{\nu j}, \tag{9.27}$$

where we have made use of the closed-shell conditions to sum over the spin degrees of freedom. The kinetic term of H is now simply $\sum_{ij} P_{ij} t_{ij}$. Next we rewrite the interaction part $\langle \Phi | a^\dagger_{i\sigma} a^\dagger_{k\sigma'} a_{l\sigma'} a_{j\sigma} | \Phi \rangle$ in terms of the P_{ij}. We find that if $\sigma = \sigma'$

$$\langle \Phi | a^\dagger_{i\sigma} a^\dagger_{k\sigma} a_{l\sigma} a_{j\sigma} | \Phi \rangle = \langle \Phi | a^\dagger_{i\sigma} a_{j\sigma} | \Phi \rangle \langle \Phi | a^\dagger_{k\sigma} a_{l\sigma} | \Phi \rangle - \langle \Phi | a^\dagger_{i\sigma} a_{l\sigma} | \Phi \rangle \langle \Phi | a^\dagger_{k\sigma} a_{j\sigma} | \Phi \rangle \tag{9.28}$$

and if $\sigma \neq \sigma'$:

$$\langle \Phi | a^\dagger_{i\sigma} a^\dagger_{k\sigma'} a_{l\sigma'} a_{j\sigma} | \Phi \rangle = \langle \Phi | a^\dagger_{i\sigma} a_{j\sigma} | \Phi \rangle \langle \Phi | a^\dagger_{k\sigma'} a_{l\sigma'} | \Phi \rangle \tag{9.29}$$

Then the energy is (again summing over the spin degrees of freedom):

$$E_0 = \sum_{ij} t_{ij} P_{ij} + \frac{1}{2} \sum_{ijkl} \left(V_{ijkl} - \frac{1}{2} V_{ilkj} \right) P_{ij} P_{kl}. \tag{9.30}$$

We can now repeat the variational arguments, minimizing E_0 under the condition that the $|\phi_\mu\rangle$ are normalized:

$$1 = \langle \phi_\mu | \phi_\mu \rangle = \sum_{i,j} d^*_{\mu i} d_{\mu j} S_{ij}. \tag{9.31}$$

Using Lagrange multipliers to enforce this constraint we finally end up with the Hartree-Fock equations for a finite basis set:

$$\sum_{j=1}^{L}(f_{ij} - \epsilon_\mu S_{ij})d_{\mu j} = 0, \tag{9.32}$$

where

$$f_{ij} = t_{ij}P_{ij} + \sum_{kl}\left(V_{ijkl} - \frac{1}{2}V_{ilkj}\right)P_{kl}. \tag{9.33}$$

This is a generalized eigenavlue problem of the form $Ax = \lambda Bx$ for which library routines exist. They first diagonalize B and then solve a standard eigenvalue problem in the basis which diagonalizes B.

9.4 Density functional theory

Another commonly used method, for which the Nobel prize in chemistry was awarded to Walter Kohn, is the density functional theory. It is based on two fundamental theorems by Hohenberg and Kohn. The first theorem states that the ground state energy E_0 of an electronic system in an external potential V is a functional of the electron density $\rho(\vec{r})$:

$$E_0 = E[\rho] = \int d^3\vec{r}V(\vec{r})\rho(\vec{r}) + F[\rho], \tag{9.34}$$

with a universal functional F. The second theorem states that the density of the ground state wave function minimizes this functional. I will prove both theorems in the lecture.

Until now everything is exact. The problem is that, while the functional F is universal, it is also unknown! Thus we need to find good approximations for the functional. One usually starts from the ansatz:

$$F[\rho] = E_h[\rho] + E_k[\rho] + E_{xc}[\rho]. \tag{9.35}$$

The Hartree-term E_h given by the Coulomb repulsion between two electrons:

$$E_h[\rho] = \frac{e^2}{2}\int d^3\vec{r}d^3\vec{r}'\frac{\rho(\vec{r})\rho(\vec{r}')}{|\vec{r}-\vec{r}'|}. \tag{9.36}$$

The kinetic energy $E_k[\rho]$ is that of a non-interacting electron gas with the same density. The exchange- and correlation term $E_{xc}[\rho]$ contains the remaining unknown contribution.

To determine the ground state wave function we again make an ansatz for the wave function, using $N/2$ single-electron wave function, which we occupy with spin $\uparrow$ and spin $\downarrow$ electrons:

$$\rho(\vec{r}) = 2\sum_{\mu=1}^{N/2}|\chi_\mu(\vec{r})|^2. \tag{9.37}$$

The single-electron wave functions are again normalized. Variation of the functional, taking into account this normalization gives an equation, which looks like a single-electron Schrödinger equation

$$\left(-\frac{\hbar^2}{2m}\nabla^2 + V_{eff}(\vec{r})\right)\chi_\mu(\vec{r}) = \epsilon_\mu\chi_\mu(\vec{r}), \tag{9.38}$$

with an effective potential

$$V_{eff}(\vec{r}) = U(\vec{r}) + e^2 \int d^3\vec{r}' \frac{\rho(\vec{r}')}{|\vec{r} - \vec{r}'|} + v_{xc}(\vec{r}), \tag{9.39}$$

and an exchange-correlation potential defined by

$$v_{xc}(\vec{r}) = \frac{\delta E_{xc}[\rho]}{\delta \rho(\vec{r})}. \tag{9.40}$$

The form (9.38) arises because we have separated the kinetic energy of the non-interacting electron system from the functional. The variation of this kinetic energy just gives the kinetic term of this Schrödinger-like equation.

This non-linear equation is again solved iteratively, where in the ansatz (9.37) the χ_μ with the lowest eigenvalues ϵ_μ are chosen.

In finite (and non-orthogonal) basis sets the same techniques as for the Hartree-Fock method are used to derive a finite eigenvalue problem.

9.4.1 Local Density Approximation

Apart from the restricted basis set everything was exact up to this point. As the functional $E_{xc}[\rho]$ and thus the potential $v_{xc}(\vec{r})$ is not known, we need to introduce approximations.

The simplest approximation is the "local density approximation" (LDA), which replaces v_{xc} by that of a uniform electron gas with the same density:

$$\begin{aligned} v_{xc} &= -\frac{0.611}{r_s}\beta(r_s)\ [a.u.] \\ \beta(r_s) &= 1 + 0.0545 r_s \ln(1 + 11.4/r_s) \\ r_s^{-1} &= a_B \left(\frac{4\pi}{3}\rho\right)^{1/3} \end{aligned} \tag{9.41}$$

9.4.2 Improved approximations

Improvements over the LDA have been an intense field of research in quantum chemistry. I will just mention two improvements. The "local spin density approximation" (LSDA) uses separate densities for electrons with spin $\uparrow$ and $\downarrow$. The "generalized gradient approximation" (GGA) and its variants use functionals depending not only on the density, but also on its derivatives.

9.5 Car-Parinello method

Roberto Car and Michele Parinello have combined density functional theory with molecular dynamics to improve the approximations involved in using only Lennard-Jones potentials. This method allows much better simulations of molecular vibration spectra and of chemical reactions.

The atomic nuclei are propagated using classical molecular dynamics, but the electronic forces which move them are estimated using density functional theory:

$$M_n \frac{d^2\vec{R}_n}{dt^2} = -\frac{\partial E[\rho(\vec{r},t),\vec{R}_n]}{\partial \vec{R}_n}. \tag{9.42}$$

Here M_n and $\vec{R}_n$ are the masses and locations of the atomic nuclei.

As the solution of the full electronic problem at every time step is a very time consuming task it is performed only once at the start. The electronic degrees of freedoms are then updated using an artificial dynamics:

$$m\frac{d^2\chi_\mu(\vec{r},t)}{dt^2} = -\frac{1}{2}\frac{\delta E[\rho(\vec{r},t),\vec{R}_n]}{\delta\chi_\mu^\dagger(\vec{r},t)} + \sum_\nu \Lambda_{\mu\nu}\chi_\nu(\vec{r},t), \tag{9.43}$$

The Lagrange multipliers $\Lambda_{\mu\nu}$ ensure proper orthonormalization of the wave functions.

9.6 Configuration-Interaction

The approximations used in Hartree-Fock and density functional methods are based on non-interacting electron pictures. They do not treat correlations and interactions between electrons correctly. To improve these methods, and to allow the calculation of excited states, often the "configuration-interaction" (CI) method is used.

Starting from the Hartree-Fock ground state

$$|\psi_{HF}\rangle = \prod_{\mu=1}^{N} c_\mu^\dagger |0\rangle \tag{9.44}$$

one or two of the $c_\mu^\dagger$ are replaced by other orbitals $c_i^\dagger$:

$$|\psi_0\rangle = \left(1 + \sum_{i,\mu}\alpha_\mu^i c_i\dagger c_\mu + \sum_{i<j,\mu<\nu}\alpha_{\mu\nu}^{ij} c_i\dagger c_j\dagger c_\mu c_\nu\right)|\psi_{HF}\rangle. \tag{9.45}$$

The energies are then minimized using this variational ansatz. In a problem with N occupied and M empty orbitals this leads to a matrix eigenvalue problem with dimension $1 + NM + N^2M^2$. Using the Lanczos algorithm the low lying eigenstates can then be calculated in $\mathrm{O}((N+M)^2)$ steps.

Further improvements are possible by allowing more than only double-substitutions. The optimal method treats the full quantum problem of dimension $(N+M)!/N!M!$. Quantum chemists call this method "full-CI". Physicists simplify the Hamilton operator slightly to obtain simpler models with fewer matrix elements, and call that method "exact diagonalization". This method will be discussed in the final chapter.

9.7 Program packages

As the model Hamiltonian and the types of basis sets are essentially the same for all quantum chemistry applications flexible program packages have been written. There is thus usually no need to write your own programs – unless you want to implement a new algorithm.

Chapter 10

The quantum N body problem: exact algorithms

The quantum chemical approaches discussed in the previous chapter simplify the problem of solving a quantum many body problem. The complexity of a problem with N particles in $M = O(N)$ orbitals is then only $O(N^4)$ or often better, instead of $O(\exp(N))$.

This enormous reduction in complexity is however paid for by a crude approximation of electron correlation effects. This is acceptable for normal metals, band insulators and semi-conductors but fails in materials with strong electron correlations, such as almost all transition metal ceramics.

In the last category many new materials have been synthesized over the past twenty years, including the famous high temperature superconductors. In these materials the electron correlations play an essential role and lead to interesting new properties that are not completely understood yet. Here we leave quantum chemistry and enter quantum physics again.

10.1 Models

To understand the properties of these materials the Hamilton operator of the full quantum chemical problem (9.4) is usually simplified to generic models, which still contain the same important features, but which are easier to investigate. They can be used to understand the physics of these materials, but not directly to quantitatively fit experiments.

10.1.1 The tight-binding model

The simplest model is the tight-binding model, which concentrates on the valence bands. All matrix elements t_{ij} in equation (9.13), apart from the ones between nearest neighbor atoms are set to zero. The others are simplified, as in:

$$H = \sum_{\langle i,j\rangle,\sigma} (t_{ij} c^\dagger_{i,\sigma} c_{j,\sigma} + \text{H.c.}). \qquad (10.1)$$

This model is easily solvable by Fourier transforming it, as there are no interactions.

10.1.2 The Hubbard model

To include effects of electron correlations, the Hubbard model includes only the often dominant intra-orbital repulsion V_{iiii} of the V_{ijkl} in equation (9.14):

$$H = \sum_{\langle i,j\rangle,\sigma} (t_{ij} c^\dagger_{i,\sigma} c_{j,\sigma} + \text{H.c.})) + \sum_i U_i n_{i,\uparrow} n_{i,\downarrow}. \tag{10.2}$$

The Hubbard model is a long-studied, but except for the 1D case still not completely understood model for correlated electron systems.

In contrast to band insulators, which are insulators because all bands are either completely filled or empty, the Hubbard model at large U is insulating at half filling, when there is one electron per orbital. The reason is the strong Coulomb repulsion U between the electrons, which prohibit any electron movement in the half filled case at low temperatures.

10.1.3 The Heisenberg model

In this insulating state the Hubbard model can be simplified to a *quantum* Heisenberg model, containing exactly one spin per site.

$$H = \sum_{\langle i,j\rangle} J_{ij} \vec{S}_i \vec{S}_j \tag{10.3}$$

For large U/t the perturbation expansion gives $J_{ij} = 2t_{ij}^2(1/U_i + 1/U_j)$. The Heisenberg model is the relevant effective models at temperatures $T \ll t_{ij}, U$ (10^4 K in copper oxides).

10.1.4 The t-J model

The t-J model is the effective model for large U at low temperatures away from half-filling. Its Hamiltonian is

$$H = \sum_{\langle i,j\rangle,\sigma} \left[(1 - n_{i,-\sigma}) t_{ij} c^\dagger_{i,\sigma} c_{j,\sigma} (1 - n_{j,-\sigma}) + \text{H.c.}\right] + \sum_{\langle i,j\rangle} J_{ij} (\vec{S}_i \vec{S}_j - n_i n_j/4). \tag{10.4}$$

As double-occupancy is prohibited in the t-J model there are only three instead of four states per orbital, greatly reducing the Hilbert space size.

10.2 Algorithms for quantum lattice models

10.2.1 Exact diagonalization

The most accurate method is exact diagonalization of the Hamiltonian matrix using the Lanczos algorithm which was discussed in section 7.10. The size of the Hilbert space of an N-site system [4^N for a Hubbard model , 3^N for a t-J model and $(2S+1)^N$ for a spin-S model] can be reduced by making use of symmetries. Translational symmetries can be employed by using Bloch waves with fixed momentum as basis states. Conservation of particle number and spin allows to restrict a calculation to subspaces of fixed particle number and magnetization.

As an example I will sketch how to implement exact diagonalization for a simple one-dimensional spinless fermion model with nearest neighbor hopping t and nearest neighbor repulsion V:

$$H = -t \sum_{i=1}^{L-1} (c_i^\dagger c_{i+1} + \text{H.c.}) + V \sum_{i=1}^{L-1} n_i n_{i+1}. \tag{10.5}$$

The first step is to construct a basis set. We describe a basis state as an unsigned integer where bit i set to one corresponds to an occupied site i. As the Hamiltonian conserves the total particle number we thus want to construct a basis of all states with N particles on L sites (or N bits set to one in L bits). The function `state(i)` returns the state corresponding to the i-th basis state, and the function `index(s)` returns the number of a basis state `s`.

```
#include <vector>
#include <alps/bitops.h>
#include <limits>
#include <valarray>

class FermionBasis {
public:
  typedef unsigned int state_type;
  typedef unsigned int index_type;
  FermionBasis (int L, int N);

  state_type state(index_type i) const {return states_[i];}
  index_type index(state_type s) const {return index_[s];}
  unsigned int dimension() const { return states_.size();}

private:
  std::vector<state_type> states_;
  std::vector<index_type> index_;
};

FermionBasis::FermionBasis(int L, int N)
{
  index_.resize(1<<L); // 2^L entries
  for (state_type s=0;s<index_.size();++s)
    if(alps::popcnt(s)==N) {
      // correct number of particles
          states_.push_back(s);
          index_[s]=states_.size()-1;
    }
    else
```

```
      // invalid state
      index_[s]=std::numeric_limits<index_type>::max();
}
```

Next we have to implement a matrix-vector multiplication $v = Hw$ for the Hamiltonian:

```
#include <cassert>

class HamiltonianMatrix : public FermionBasis {
public:
  HamiltonianMatrix(int L, int N, double t, double V)
    : FermionBasis(L,N), t_(t), V_(V), L_(L) {}

  void multiply(std::valarray<double>& v, const std::valarray<double>& w);

private:
  double t_, V_;
  int L_;
};

void HamiltonianMatrix::multiply(std::valarray<double>& v,
               const std::valarray<double>& w)
{
  // check dimensions
  assert(v.size()==dimension());
  assert(w.size()==dimension());

  // do the V-term
  for (int i=0;i<dimension();++i)
  {
    state_type s = state(i);
    // count number of neighboring fermion pairs
    v[i]=w[i]*V_*alps::popcnt(s&(s>>1));
  }

  // do the t-term
  for (int i=0;i<dimension();++i)
  {
    state_type s = state(i);
    for (int r=0;r<L_-1;++r) {
      state_type shop = s^(3<<r);  // exchange two particles
      index_type idx = index(shop); // get the index
      if(idx!=std::numeric_limits<index_type>::max())
        v[idx]+=w[i]*t_;
    }
  }
}
```

This class can now be used together with the Lanczos algorithm to calculate the energies and wave functions of the low lying states of the Hamiltonian.

In production codes one uses all symmetries to reduce the dimension of the Hilbert space as much as possible. In this example translational symmetry can be used if periodic boundary conditions are applied. The implementation gets much harder then.

In order to make the implementation of exact diagonalization much easier we have generalized the expression templates technique developed by Todd Veldhuizen for array expression to expressions including quantum operators. Using this expression template library we can write a multiplication

$$|\psi\rangle = H|\phi\rangle = (-t\sum_{i=1}^{L-1}(c_i^\dagger c_{i+1} + \text{H.c.}) + V\sum_{i}^{L-1} n_i n_{i+1})|\phi\rangle \tag{10.6}$$

simply as:

```
Range i(1,L-1);
psi = sum(i,(-t*(cdag(i)*c(i+1)+HC)+V*n(i)*n(i+1))*phi);
```

The advantage of the above on-the-fly calculation of the matrix in the multiplication routine is that the matrix need not be stored in memory, which is an advantage for the biggest systems where just a few vectors of the Hilbert space will fit into memory. If one is not as demanding and wants to simulate a slightly smaller system, where the (sparse) matrix can be stored in memory, then a less efficient but more flexible function can be used to create the matrix, since it will be called only once at the start of the program. Such a program is available through the ALPS project at `http://alps.comp-phys.org/`.

10.2.2 Quantum Monte Carlo

Path-integral representation in terms of world lines

All quantum Monte Carlo algorithms are based on a mapping of a d-dimensional quantum system to a $(d+1)$-dimensional classical system using a path-integral formulation. We then perform classical Monte Carlo updates on the world lines of the particles. I will introduce one modern algorithm for lattice models, the “loop-algorithm” which is a generalization of the cluster algorithms to lattice models. Other (non-cluster) algorithms are avaliable also for continuum models. The interested student will find descriptions of those algorithms in many text books on computational physics.

I will discuss the loop algorithm for a spin-1/2 quantum XXZ model with the Hamiltonian

$$\begin{aligned} H &= -\sum_{\langle i,j\rangle}\left(J_z S_i^z S_j^z + J_{xy}(S_i^x S_j^x + S_i^y S_j^y)\right) \\ &= -\sum_{\langle i,j\rangle}\left(J_z S_i^z S_j^z + \frac{J_{xy}}{2}(S_i^+ S_j^- + S_i^- S_j^+)\right). \end{aligned} \tag{10.7}$$

For $J \equiv J_z = J_{xy}$ we have the Heisenberg model ($J > 0$ is ferromagnetic, $J < 0$ antiferromagnetic). $J_{xy} = 0$ is the (classical) Ising model and $J_z = 0$ the quantum XY model.

In the quantum Monte Carlo simulation we want to evaluate thermodynamic averages such as

$$\langle A \rangle = \frac{\mathrm{Tr} A e^{-\beta H}}{\mathrm{Tr} e^{-\beta H}}. \tag{10.8}$$

The main problem is the calculation of the exponential $e^{-\beta H}$. The straightforward calculation would require a complete diagonalization, which is just what we want to avoid. We thus discretize the imaginary time (inverse temperature) direction[1] and subdivide $\beta = M\Delta\tau$:

$$e^{-\beta H} = \left(e^{-\Delta\tau H}\right)^M = (1 - \Delta\tau H)^M + O(\Delta\tau) \tag{10.9}$$

In the limit $M \to \infty$ ($\Delta\tau \to 0$) this becomes exact. We will take the limit later, but stay at finite $\Delta\tau$ for now.

The next step is to insert the identity matrix, represented by a sum over all basis states $1 = \sum_i |i\rangle\langle i|$ between all operators $(1 - \Delta\tau H)$:

$$\begin{aligned} Z &= \mathrm{Tr} e^{-\beta H} = \mathrm{Tr}\,(1 - \Delta\tau H)^M + O(\Delta\tau) \\ &= \sum_{i_1,\ldots,i_M} \langle i_1|1 - \Delta\tau H|i_2\rangle\langle i_2|1 - \Delta\tau H|i_3\rangle \cdots \langle i_M|1 - \Delta\tau H|i_1\rangle + O(\Delta\tau) \\ &=: P_{i_1,\ldots,i_M} \end{aligned} \tag{10.10}$$

and similarly for the measurement, obtaining

$$\langle A \rangle = \sum_{i_1,\ldots,i_M} \frac{\langle i_1|A(1 - \Delta\tau H)|i_2\rangle}{\langle i_1|1 - \Delta\tau H|i_2\rangle} P_{i_1,\ldots,i_M} + O(\Delta\tau). \tag{10.11}$$

If we choose the basis states $|i\rangle$ to be eigenstates of the local S^z operators we end up with an Ising-like spin system in one higher dimension. Each choice $i_1, \ldots, i_M$ corresponds to one of the possible configurations of this classical spin system. The trace is mapped to periodic boundary conditions in the imaginary time direction of this classical spin system. The probabilities are given by matrix elements $\langle i_n|1 - \Delta\tau H|i_{n+1}\rangle$. We can now sample this classical system using classical Monte Carlo methods.

However, most of the matrix elements $\langle i_n|1 - \Delta\tau H|i_{n+1}\rangle$ are zero, and thus nearly all configurations have vanishing weight. The only non-zero configurations are those where neighboring states $|i_n\rangle$ and $|i_{n+1}\rangle$ are either equal or differ by one of the off-diagonal matrix elements in H, which are nearest neighbor exchanges by two opposite spins. We can thus uniquely connect spins on neighboring "time slices" and end up with world lines of the spins, sketched in Fig. 10.1. Instead of sampling over all configurations of local spins we thus have to sample only over all world line configurations (the others have vanishing weight). Our update moves are not allowed to break world lines but have to lead to new valid world line configurations.

[1]Time evolution in quantum mechanics is e^{-itH}. The Boltzman factor $e^{-\beta H}$ thus corresponds to an evolution in imaginary time $t = -i\beta$

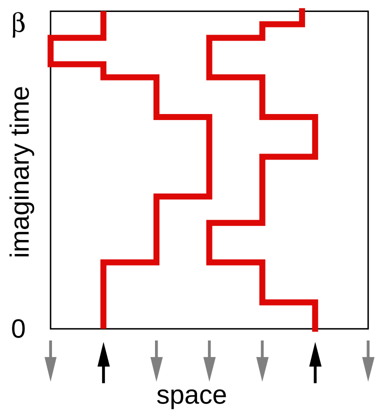


Figure 10.1: Example of a world line configuration for a spin-1/2 quantum Heisenberg model. Drawn are the world lines for up-spins only. Down spin world lines occupy the rest of the configuration.

Table 10.1: The six local configurations for an XXZ model and their weights.

configuration		weight
$s_i(\tau+d\tau)\uparrow \quad \uparrow s_j(\tau+d\tau)$ $s_i(\tau)\uparrow \quad \uparrow s_j(\tau)$,	$s_i(\tau+d\tau)\downarrow \quad \downarrow s_j(\tau+d\tau)$ $s_i(\tau)\downarrow \quad \downarrow s_j(\tau)$	$1+\frac{J_z}{4}d\tau$
$s_i(\tau+d\tau)\uparrow \quad \downarrow s_j(\tau+d\tau)$ $s_i(\tau)\uparrow \quad \downarrow s_j(\tau)$,	$s_i(\tau+d\tau)\downarrow \quad \uparrow s_j(\tau+d\tau)$ $s_i(\tau)\downarrow \quad \uparrow s_j(\tau)$	$1-\frac{J_z}{4}d\tau$
$s_i(\tau+d\tau)\downarrow \quad \uparrow s_j(\tau+d\tau)$ $s_i(\tau)\uparrow \quad \downarrow s_j(\tau)$,	$s_i(\tau+d\tau)\uparrow \quad \downarrow s_j(\tau+d\tau)$ $s_i(\tau)\downarrow \quad \uparrow s_j(\tau)$	$\frac{J_{xy}}{2}d\tau$

The loop algorithm

Until 1993 only local updates were used, which suffered from a slowing down like in the classical case. The solution came as a generalization of the cluster algorithms to quantum systems.[2]

This algorithm is best described by first taking the continuous time limit $M \to \infty$ ($\Delta\tau \to d\tau$) and by working with infinitesimals. Similar to the Ising model we look at two spins on neigboring sites i and j at two neighboring times τ and $\tau + d\tau$, as sketched in Tab. 10.1. There are a total of six possible configurations, having three different probabilities. The total probabilities are the products of all local probabilities, like in the classical case. This is obvious for different time slices. For the same time slice it is also true since, denoting by H_{ij} the term in the Hamiltonian H acting on the bond between sites i and j we have $\prod_{\langle i,j\rangle}(1 - d\tau H_{ij}) = 1 - d\tau \sum_{\langle i,j\rangle} H_{ij} = 1 - d\tau H$. In the following we focus only on such local four-spin plaquettes. Next we again use the

[2]H. G. Evertz et al., Phys. Rev. Lett. **70**, 875 (1993); B. B. Beard and U.-J. Wiese, Phys. Rev. Lett. **77**, 5130 (1996); B. Ammon, H. G. Evertz, N. Kawashima, M. Troyer and B. Frischmuth, Phys. Rev. B **58**, 4304 (1998).

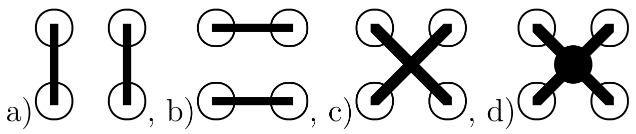
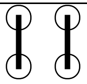
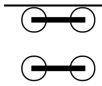
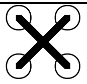
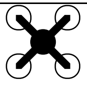


Figure 10.2: The four local graphs: a) vertical, b) horizontal c) crossing and d) freezing (connects all four corners).

Table 10.2: The graph weights for the quantum-XY model and the Δ function specifying whether the graph is allowed. The dash – denotes a graph that is not possible for a configuration because of spin conservation and has to be zero.

G	$\Delta\left(\begin{smallmatrix}\uparrow & \uparrow\\ \uparrow & \uparrow\end{smallmatrix}, G\right) = \Delta\left(\begin{smallmatrix}\downarrow & \downarrow\\ \downarrow & \downarrow\end{smallmatrix}, G\right)$	$\Delta\left(\begin{smallmatrix}\uparrow & \downarrow\\ \uparrow & \downarrow\end{smallmatrix}, G\right) = \Delta\left(\begin{smallmatrix}\downarrow & \uparrow\\ \downarrow & \uparrow\end{smallmatrix}, G\right)$	$\Delta\left(\begin{smallmatrix}\downarrow & \uparrow\\ \uparrow & \downarrow\end{smallmatrix}, G\right) = \Delta\left(\begin{smallmatrix}\uparrow & \downarrow\\ \downarrow & \uparrow\end{smallmatrix}, G\right)$	graph weight
	1	1	–	$1 - \frac{J_{xy}}{4}d\tau$
	–	1	1	$\frac{J_{xy}}{4}d\tau$
	1	–	1	$\frac{J_{xy}}{4}d\tau$
	0	0	0	0
total weight	1	1	$\frac{J_{xy}}{2}d\tau$	

Kandel-Domany framework and assign graphs. As the updates are not allowed to break world lines only four graphs, sketched in Fig. 10.2 are allowed. Finally we have to find Δ functions and graph weights that give the correct probabilities. The solution for the XY-model, ferromagnetic and antiferromagnetic Heisenberg model and the Ising model is shown in Tables 10.2 - 10.5.

Let us first look at the special case of the Ising model. As the exchange term is absent in the Ising model all world lines run straight and can be replaced by classical spins. The only non-trivial graph is the "freezing", connecting two neighboring world lines. Integrating the probability that two neighboring sites are *nowhere* connected along the time direction we obtain: times:

$$\prod_{\tau=0}^{\beta}(1 - d\tau J/2) = \lim_{M\to\infty}(1 - \Delta\tau J/2)^M = \exp(-\beta J/2) \tag{10.12}$$

Taking into account that the spin is $S = 1/2$ and the corresponding classical coupling $J_{cl} = S^2 J = J/4$ we find for the probability that two spins are *connected*: $1 - \exp(-2\beta J_{cl})$. We end up exactly with the cluster algorithm for the classical Ising model!

Table 10.3: The graph weights for the ferromagnetic quantum Heisenberg model and the Δ function specifying whether the graph is allowed. The dash – denotes a graph that is not possible for a configuration because of spin conservation and has to be zero.

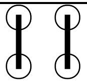
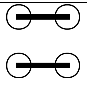
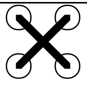
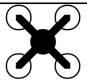
G	$\Delta\left(\begin{smallmatrix}\uparrow & \uparrow\\ \uparrow & \uparrow\end{smallmatrix}, G\right) = \Delta\left(\begin{smallmatrix}\downarrow & \downarrow\\ \downarrow & \downarrow\end{smallmatrix}, G\right)$	$\Delta\left(\begin{smallmatrix}\uparrow & \downarrow\\ \uparrow & \downarrow\end{smallmatrix}, G\right) = \Delta\left(\begin{smallmatrix}\downarrow & \uparrow\\ \downarrow & \uparrow\end{smallmatrix}, G\right)$	$\Delta\left(\begin{smallmatrix}\downarrow & \uparrow\\ \uparrow & \downarrow\end{smallmatrix}, G\right) = \Delta\left(\begin{smallmatrix}\uparrow & \downarrow\\ \downarrow & \uparrow\end{smallmatrix}, G\right)$	graph weight
	1	1	–	$1 - \frac{J}{4}d\tau$
	–	0	0	0
	1	–	1	$\frac{J}{2}d\tau$
	0	0	0	0
total weight	$1 + \frac{J}{4}d\tau$	$1 - \frac{J}{4}d\tau$	$\frac{J}{2}d\tau$	

Table 10.4: The graph weights for the antiferromagnetic quantum Heisenberg model and the Δ function specifying whether the graph is allowed. The dash – denotes a graph that is not possible for a configuration because of spin conservation and has to be zero. To avoid the sign problem (see next subsection) we change the sign of J_{xy}, which is allowed only on bipartite lattices.

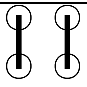
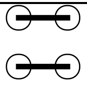
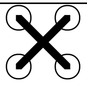
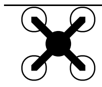
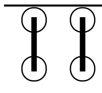
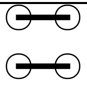
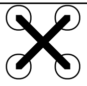
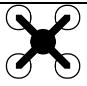
G	$\Delta\left(\begin{smallmatrix}\uparrow & \uparrow\\ \uparrow & \uparrow\end{smallmatrix}, G\right) = \Delta\left(\begin{smallmatrix}\downarrow & \downarrow\\ \downarrow & \downarrow\end{smallmatrix}, G\right)$	$\Delta\left(\begin{smallmatrix}\uparrow & \downarrow\\ \uparrow & \downarrow\end{smallmatrix}, G\right) = \Delta\left(\begin{smallmatrix}\downarrow & \uparrow\\ \downarrow & \uparrow\end{smallmatrix}, G\right)$	$\Delta\left(\begin{smallmatrix}\downarrow & \uparrow\\ \uparrow & \downarrow\end{smallmatrix}, G\right) = \Delta\left(\begin{smallmatrix}\uparrow & \downarrow\\ \downarrow & \uparrow\end{smallmatrix}, G\right)$	graph weight
	1	1	–	$1 - \frac{\lvert J\rvert}{4}d\tau$
	–	1	1	$\frac{\lvert J\rvert}{2}d\tau$
	0	–	0	0
	0	0	0	0
total weight	$1 - \frac{\lvert J\rvert}{4}d\tau$	$1 + \frac{\lvert J\rvert}{4}d\tau$	$\frac{\lvert J\rvert}{2}d\tau$	

Table 10.5: The graph weights for the ferromagnetic Ising model and the Δ function specifying whether the graph is allowed. The dash – denotes a graph that is not possible for a configuration because of spin conservation and has to be zero.

G	$\Delta(\substack{\uparrow\ \uparrow\\ \uparrow\ \uparrow}, G) = \Delta(\substack{\downarrow\ \downarrow\\ \downarrow\ \downarrow}, G)$	$\Delta(\substack{\uparrow\ \downarrow\\ \uparrow\ \downarrow}, G) = \Delta(\substack{\downarrow\ \uparrow\\ \downarrow\ \uparrow}, G)$	$\Delta(\substack{\downarrow\ \uparrow\\ \uparrow\ \downarrow}, G) = \Delta(\substack{\uparrow\ \downarrow\\ \downarrow\ \uparrow}, G)$	graph weight
	1	1	–	$1 - \frac{J_z}{4} d\tau$
	–	0	0	0
	0	–	0	0
	1	0	0	$\frac{J_z}{2} d\tau$
total weight	$1 + \frac{J_z}{4} d\tau$	$1 - \frac{J_z}{4} d\tau$	0	

The other cases are special. Here each graph connects two spins. As each of these spins is again connected to only one other, all spins connected by a cluster form a closed loop, hence the name "loop algorithm". Only one issue remains to be explained: how do we assign a horizontal or crossing graph with infinitesimal probability, such as $(J/2)d\tau$. This is easily done by comparing the assignment process with radioactive decay. For each segment the graph runs vertical, except for occasional decay processes occuring with probability $(J/2)d\tau$. Instead of asking at every infinitesimal time step whether a decay occurs we simply calculate an exponentially distributed decay time t using an exponential distribution with decay constant $J/2$. Looking up the equation in the lecture notes of the winter semester we have $t = -(2/J)\ln(1-u)$ where u is a uniformly distributed random number.

The algorithm now proceeds as follows (see Fig. 10.3): for each bond we start at time 0 and calculate a decay time. If the spins at that time are oriented properly and an exchange graph is possible we insert one. Next we advance by another randomly chosen decay time along the same bond and repeat the procedure until we have reached the extent β. This assigns graphs to all infinitesimal time steps where spins do not change. Next we assign a graph to all of the (finite number of) time steps where two spins are exchanged. In the case of the Heisenberg models there is always only one possible graph to assign and this is very easy. In the next step we identify the loop-clusters and then flip them each with probability 1/2. Alternatively a Wolff-like algorithm can be constructed that only builds one loop-cluster.

Improved estimators for measurements can be constructed like in classical models. The derivation is similar to the classical models. I will just mention two simple ones

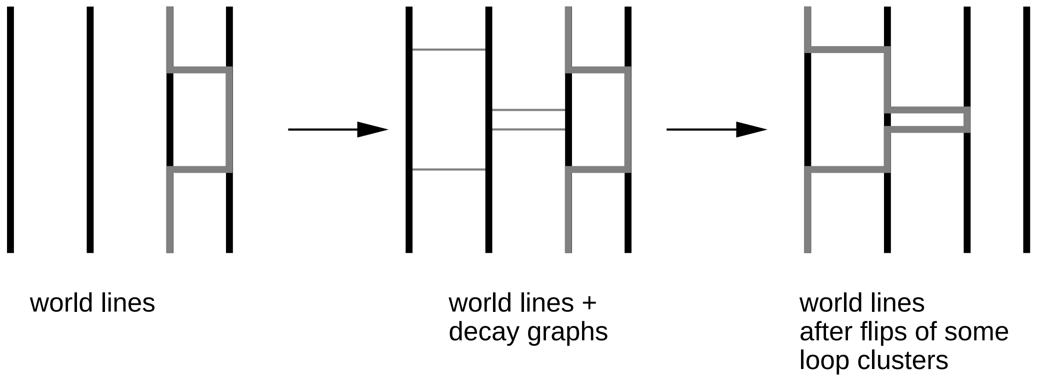


Figure 10.3: Example of a loop update. In a first step decay paths are inserted where possible at positions drawn randomly according to an exponential distribution and graphs are assigned to all exchange terms (hoppings of world lines). In a second stage (not shown) the loop clusters are identified. Finally each loop cluster is flipped with probability 1/2 and one ends up with a new configuration.

for the ferromagnetic Heisenberg model. The spin-spin corelation is

$$S_i^z(\tau)S_j^z(\tau') = \begin{cases} 1 & \text{if } (i,\tau) \text{ und } (j,\tau') \text{ are on the same cluster} \\ 0 & \text{otherwise} \end{cases} \tag{10.13}$$

and the uniform susceptibilty is

$$\chi = \frac{1}{N\beta}\sum_c S(c)^2, \tag{10.14}$$

where the sum goes over all loop clusters and $S(c)$ is the length of all the loop segments in the loop cluster c.

The negative sign problem

Now that we have an algorithm with no critical slowing down we could think that we have completely solved the problem of quantum many body problems. Indeed the scaling of the loop algorithm is $O(N\beta)$ where N is the number of lattice sites and β the inverse temperature – this is optimum scaling.

There is however the *negative sign problem* which destroys our dreams. We need to interpret the matrix elements $\langle i_n|1-\Delta\tau H|i_{n+1}\rangle$ as probablities, which requires them to be positive. However all off-diagonal positive matrix elements of H give rise to a negative probability!

The simplest example is the exchange term $-(J_{xy}/2)(S_i^+S_j^- + S_i^-S_j^+)$ in the Hamiltonian (10.7) in the case of an antiferromagnet with $J_{xy} < 0$. For any bipartite lattice, such as chains, square lattices or cubic lattices with there is always an even number of such exchanges and we get rescued once more. For non-bipartite lattices (such as a triangular lattice), on which the antiferromagnetic order is frustrated there is no way around the sign problem. Similarly a minus sign occurs in all configurations where two fermions are exchanged.

Even when there is a sign problem we can still do a simulation. Instead of sampling

$$\langle A \rangle_p := \frac{\int A(x)p(x)dx}{\int p(x)dx} \tag{10.15}$$

we rewrite this equation as

$$\langle A \rangle_p = \frac{\frac{\int A(x)\mathrm{sign}(p(x))|p(x)|dx}{\int |p(x)|dx}}{\frac{\int \mathrm{sign}(p(x))|p(x)|dx}{\int |p(x)|dx}} = \frac{\langle A \cdot \mathrm{sign}p \rangle_{|p|}}{\langle \mathrm{sign}p \rangle_{|p|}}. \tag{10.16}$$

We sample with the absolute values $|p|$ and include the sign in the observable. The "sign problem" is the fact that the errors get blown up by an additional factor $1/\langle \mathrm{sign}p \rangle_{|p|}$, which grows exponentially with volume and inverse temperature β, as $\langle \mathrm{sign}p \rangle_{|p|} \propto \exp(-\mathrm{const} \times \beta N)$. Then we are unfortunately back to exponential scaling.

The sign problem occurs not only for frustrated magnets, but for any fermionic quantum system in moer than one dimension: the wave function changes sign when two fermions are exchanged and hence any world line configuration where two fermions exchange their positions during the propagation from imaginary time 0 to β will contribute with a negative weight. Many people have tried to solve the sign problem using basis changes or clever reformulations, but – except for special cases – nobody has succeeded yet. If you want you can try your luck: the person who finds a general solution to the sign problem will surely get a nobel prize. Unfortunately it was recently shown that the negative sign problem is NP-hard and thus almost certainly unsolvable in the general case. [3]

10.2.3 Density Matrix Renormalization Group methods

The density matrix renormalization group (DMRG) method uses a clever trick of reducing the Hilbert space size by selecting important states. The idea is to grow the lattice by just a few sites in each renormalization step. If all basis states are kept, this leads to the well-known exponential increase in size. Instead of keeping all basis functions, the DMRG method selects a number of m "important" states according to their density matrix eigenvalue. A good reference is the paper by S.R. White in Phys. Rev. B **48**, 10345 (1993), as well as the doctoral thesis of B. Ammon.

10.3 Lattice field theories

We did not talk much about field theories since a discussion of algorithms for lattice field theories requires a good knowledge of analytical approaches for field theories first. Here I just want to sketch how a classical or quantum field theory can be simulated.

[3] M. Troyer and U.J. Wiese, Phys. Rev. Lett. **94**, 170201 (2005).

10.3.1 Classical field theories

As an example I choose the classical $O(N)$ nonlinear sigma model in d dimensions, with an action:

$$S = g \int d^d x |\nabla \vec{\Omega}(x)|^2 \qquad \text{with} \qquad |\vec{\Omega}(x)| = 1 \tag{10.17}$$

living in a finite box of size L^d. To simulate this field theory we introduce a finite lattice spacing a and obtain a grid of dimension M^d with $M = L/a$. Let us denote the value of the field $\vec{\Omega}$ on the grid points by $\vec{\Omega}_i$. Discretizing the action S by replacing derivatives by differences we obtain, making use of $|\vec{\Omega}(x)| = 1$

$$S_a = \frac{2g}{a^2} \sum_{\langle i,j \rangle} \vec{\Omega}_i \vec{\Omega}_j. \tag{10.18}$$

This is nothing but the action of a claccisal $O(N)$ lattice model in d dimension with $\beta J = 2g/a^2$ and we can again use the cluster algorithms. $N = 1$ is the Ising model, $N = 2$ the XY-model and $N = 3$ the Heisenberg model. One calls the model (10.17) the “effective field theory” of the $O(N)$ lattice models.

There is however a subtle but important difference. In statistical mechanics we had a fixed lattice spacing a and let $L \to \infty$ to approach the thermodynamic limit. In the field theory we keep L fixed and let $a \to 0$, while scaling βJ like $2g/a^2$. This leads to different interpretations of the results.

10.3.2 Quantum field theories

Quantum field theories can be treated similarly. Using a path integral formulation like we introduced it for quantum lattice models the d-dimensional quantum field theory is mapped to a $(d+1)$-dimensional classical one: Let us consider the quantum nonlinear sigma model, a Lorentz invariant field theory with an effective action:

$$S = g \int_0^\beta d\tau \int d^d x \left(|\nabla \vec{\Omega}(x,\tau)|^2 + \frac{1}{c^2} |\partial \vec{\Omega}(x,\tau)/\partial \tau|^2 \right) \qquad \text{with} \qquad |\vec{\Omega}(x,\tau)| = 1, \tag{10.19}$$

where β is the inverse temperature and the velocity c relates the spatial and temporal scales. Introducing a lattice spacing a in the spatial direction and a lattice spacing a/c in the temporal direction we obtain a classical $O(N)$ lattice model with couplings $\beta_{cl} J = 2g/a^2$. The extent in the space direction is L/a and in the temporal direction $\beta c/a$. Thus we see that the *coupling constant* of the quantum model gets mapped to the *classical temperature* and the *temperature* gets mapped to the *extent in imaginary time.* Again we can use the classical cluster algorithms to study this quantum field theory.

For details and simulation algorithms for other field theories the interested student is refered to the book by J.M. Thijssen.

Enjoy your holidays!