Computational Physics

Spring 1996

Martin Siegert

Department of Physics

Simon Fraser University

Computational Physics

Contents

Reference Books

(i) W. H. Press, S. A. Teukolski, W. T. Vetterling, B. P. Flannery, *Numerical Recipes*, Second Edition (Fortran and C Version available), Cambridge University Press, Cambridge 1994.
This is an excellent book discussing numerical algorithm for many different problems with good explanations of the mathematical background. However, it does not contain any applications to physics. In particular, nothing can be found about Monte-Carlo or molecular dynamics simulations.

(ii) H. Gould, J. Tobochnik, *An Introduction to Computer Simulation Methods* (Parts 1 and 2), Addison-Wesley, Reading 1988.
These two volumes contain many applications to physical systems, however the computational methods used to solve these problems are very simplistic, e.g., in the first half of part 1 the Euler method is used to integrate differential equations.

(iii) S. E. Koonin, D. C. Meredith, *Computational Physics*, Addison-Wesley, Reading 1990.
This book also contains numerical solutions to many different problems in physics. However, the discussion of the algorithms is rather disappointing: an error analysis is generally missing. Although the book has around 600 pages, it is rather short: the last 2/3 of the book contains source code of example programs.

(iv) E. W. Schmid, G. Spitz, W. Lösch, *Theoretical Physics on the Personal Computer*, Springer, Berlin 1990.
Discusses almost exclusively the numerical integration of differential equations.

(v) D. W. Heermann, *Computer Simulation Methods*, Springer, Berlin 1990.
This book discusses exclusively molecular dynamics simulations and Monte-Carlo simulations. The book is rather short and therefore many details that are important for the actual simulation are missing. Furthermore, several important topics are missing as well, e.g., finite-size scaling is not explained. The source code of the Kirkpatrick-Stoll random-number generator given in the appendix uses nonstandard language elements and is therefore useless for most readers.

(vi) M. P. Allen, D. J. Tildesley, *Computer Simulations of Liquids*, Oxford University Press, Oxford 1990. This book discusses molecular dynamics simulations and Monte-Carlo simulations with the emphasis on simulation of liquids. It gives probably the best introduction to molecular dynamics simulations.

(vii) D. C. Rapaport, *The Art of Molecular Dynamics Simulation*, Cambridge University Press, Cambridge 1995.
Fairly extensive description of molecular dynamics simulations; example programs in C.

Introduction

This course is intended to serve several purposes, e.g., the students should

(i) become accustomed to a modern computer environment of UNIX workstations,
(ii) learn to use a higher programming language such as Fortran90 or C/C++,
(iii) learn numerical methods, algorithms to solve problems in physics,
(iv) learn how to evaluate measured data, either experimental data or from a numerical simulation.

There are several fields in physics that could be included in a course of Computational Physics:

1. <u>Numerical Analysis</u>

The physical principles to solve a problem are known and lead to equations for the relevant quantities. If these equations cannot be solved analytically, they may be solved numerically.
Examples: Calculate the oscillation frequencies of a complicated molecule; calculate the trajectory of a rocket flying from the earth to the moon, etc.

2. <u>Symbolic Formula Manipulation</u>

This deals mostly with the conversion, simplification, and graphical representation of mathematical formula. In a limited way also the numerical solution of equations is possible. Common programs for this kind of computer-algebra are: Maple, Mathematica, or Reduce.

3. <u>Numerical Simulations</u>

Natural processes are modeled in a computer experiment with the aim to understand the physical origin of observed phenomena and/or to make predictions for real experiments.

Example: Is an alloy consisting of 50% iron and 50% copper still magnetic? How does the magnetism depend on temperature, concentration?
Advantage over a laboratory experiment: It is possible to look at ideal systems, to turn off interfering side effects such as friction, and to tune parameters that may not be accessible in an experiment. In this way it is possible to make a connection with the real world and simplified systems that can be analyzed theoretically.

4. <u>Control of Experiments</u>

An experiment sends signals to a computer that in turn adjusts the experimental parameters and analyses the data.

Comparison:

Laboratory Experiment	Computer Simulation
sample	model
measuring apparatus	computer program
calibration	test of the program for special cases
measurement	simulation
data analysis	data analysis

In this course only problems that belong to the categories 1. and 3. will be discussed.

1. Preliminaries

1.1 Introduction to Computer Hardware

a) Computer Types

Computer Power ↓

1) PC (IBM or Apple)
2) Workstation
3) Mainframe

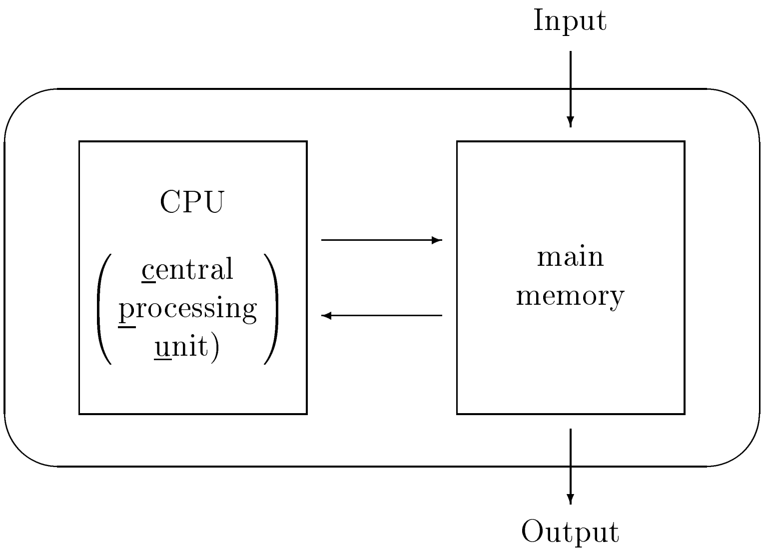
b) Usage

1) PC: usually single user ("personal" computer). Therefore, no special access protection for files. If the computer "hangs up" you may turn it off and on again and the PC reboots.

2) Workstation: Multi-user system. You do not have to sit in front of the workstation in order to use it, e.g., you can connect over a network. You also can start programs running in the background and disconnect and the programs will continue to run.
Therefore: **Never ever switch off a workstation !**
You can destroy many hours of work of your fellow students and, without a proper shutdown, it can be difficult to restart the system. Even if the system does not appear to work properly it may be possible to resolve the problems from a remote machine without a shutdown. A shutdown and restart can only be performed by the system manager.
Workstations are today's most modern computers at scientist's workplaces.

3) Mainframes: Large multi-user machines that serve a university, a whole region or even a whole country. You usually have to write a proposal before you get an account on these machines. You may also have to pay for the computer time.

c) Computer Structure and Operation



Input occurs usually over the keyboard or the mouse, but also over data or command files. Output is either displayed on the monitor or sent to the printer, but may also stored in files on the hard disk or on a diskette.

Mode of Operation: All components of a computer respond to two states of electrical voltage (called 0 and 1). Input and output are coded in a sequence of zeros and ones (binary code). The amount of information that can be obtained by a yes-no question is called a *bit*. The commands, i.e., the program, that the computer is to execute as well as the data the computer has to read in, compute, and output are stored as binary code at distinct addresses in the memory. The size of the memory is given in *bytes*, 1 *byte* = 8 *bits*. A PC has usually 4 to 16 MB, the Workstations in the Physics Computing Laboratory have 64 MB ($64\,MB = 64 \times (2^{10})^2$ *byte*, $2^{10} = 1024 \simeq 10^3$). The CPU reads the commands out of the memory and executes them one after the other.

1.2 Basic Software Components

"Software" is nothing else than a program. However, in order to write and execute your own programs, a minimum of software components must already be present on the computer.

a) The Operating System
This is a program that controls all basic features of the computer: It controls input and output, e.g., sends your keyboard input to the correct memory addresses, etc., it controls the disk drives (hard disk and diskettes), it controls the reading of commands and data out of the main memory, etc. Without an operating system the CPU is "dead".

Common operating systems:

DOS, OS/2 and Apple	for PCs
UNIX	for workstations and mainframes (UNIX is also available for IBM-PCs, and it is even public domain: "Linux")
⋮	

Today, most operation systems include a window manager that enables you to work with several windows on the screen, to use the mouse, etc. In the UNIX environment this graphical interface is called "X-Windows".

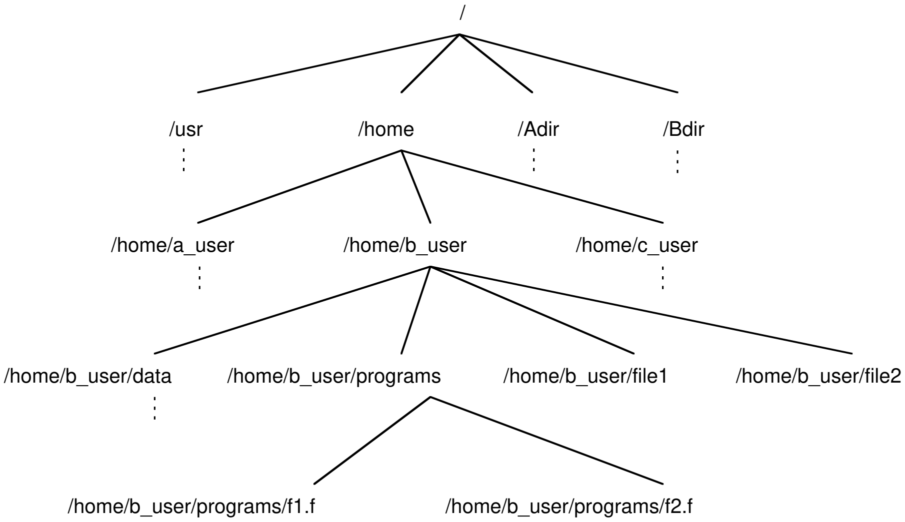
b) Editor
To write the source code of a program you need an editor. This program allows you to enter text over the keyboard and save the text as a file on the hard disk. There are many different editors, under UNIX two common editors are "vi" and "emacs" (emacs also available for OS/2).

c) Compiler
This program is needed to translate the source code of a program, which is a text file, into an executable file: Depending on the programming language there are F77, F90, C, C++, etc., compilers.

2. Introduction to UNIX

Filesystem

Similar to DOS, Apple UNIX uses a hierarchical file system: files can be combined into directories, which in turn can be combined again into directories, etc. The directory on the highest level is simply called “/”. Any directory may contain files and subdirectories.
Example:



The directory /usr and its subdirectories contains most of the system software, programs, etc. Every user has a “home” directory, where she/he can create own files/subdirectories. This directory should not be confused with the directory /home, in fact, on most UNIX machines the home directory of the user xy is /home/xy. This can be abbreviated as ~xy, your own home directory is simply ~.

Commands

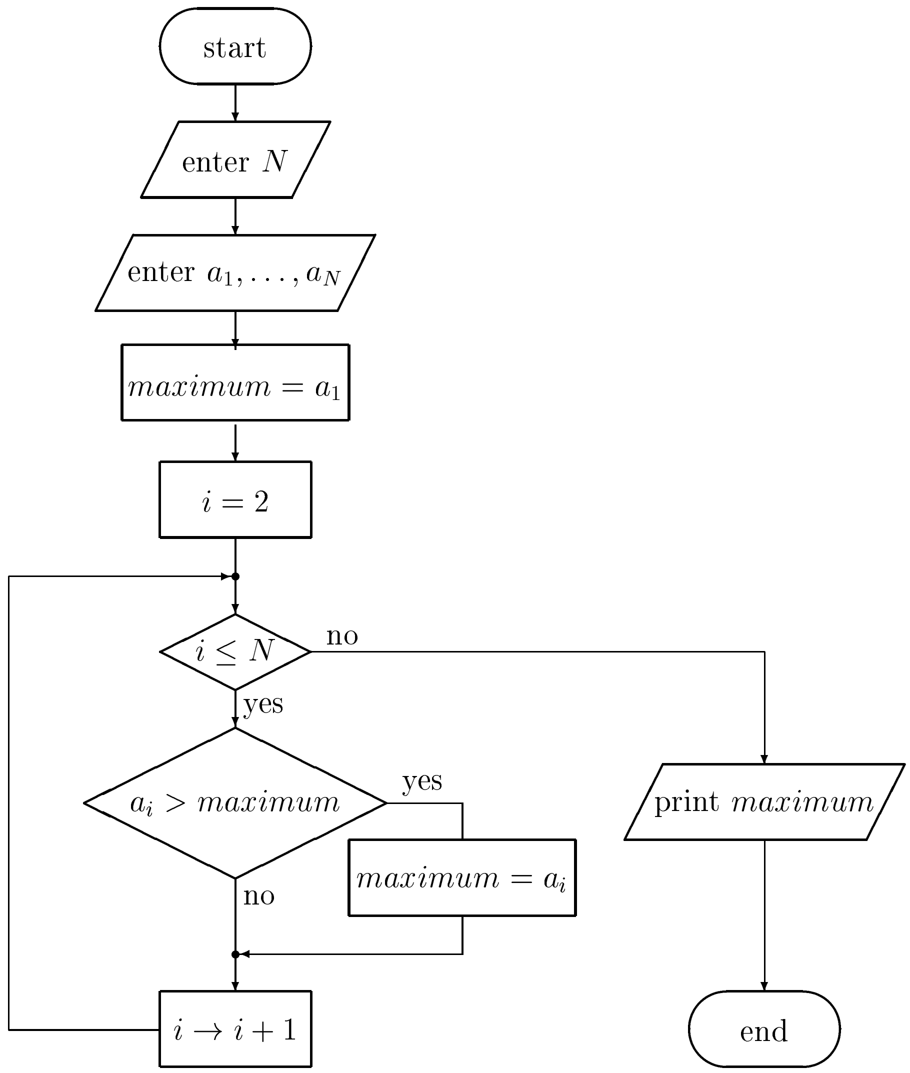
- man <command> online manual of the UNIX system, explains the command syntax of command, e.g., man man explains the syntax of the man command.
- cd <directory> change directory, e.g., assuming that you are in the directory ~b_user the command cd programs changes to the directory ~b_user/programs. It is not necessary to type the full path name, the relative path name suffices.
- ls lists the contents of the current directory. This command has many options, see man ls.
- cp <file1> <file2> or cp <file1> <dir1> copies file1 to file2, resp. copies file1 into the directory dir1 (with the same name).
- mv <file1> <file2> or mv <file1> <dir1> moves files, same as cp, but the original file1 is deleted after the process.
- mkdir <directory> creates a directory
- rmdir <directory> removes a directory (works only if the directory is empty).
- rm <file> removes a file. Be careful, especially if you use wildcard characters such as *. UNIX does not ask, whether you really want to do this, e.g., rm * deletes all files in the current directory and there is no undelete on the UNIX system.

3. Introduction to Programming

3.1 From Algorithms to Machine Code

An algorithm is a method to solve a particular problem. A program is an implementation of an algorithm according to the rules of a programming language.
Example: Find the largest of N numbers $a_1, a_2, \ldots, a_N$.
Algorithm: Find the larger number of a_1 and a_2. Compare this number with a_3, etc.

Flowdiagram:



Program:

```
program maximum
allocatable a(:)              !  declare a as an one-dimensional array
read*,n                       !  read number of elements
allocate (a(n))               !  allocate storage for array elements
read*,(a(i),i=1,n)
rmax=a(1)                     !  set initial value for maximum
do i=2,n                      !  do loop
  if (a(i) > rmax) rmax=a(i)  !  if a(i) > maximum set maximum=a(i)
end do
print*,rmax                   !  output
end
```

Finally the compiler (here F90) will generate from the source code an executable program.

3.2 Number Representation, Rounding Errors

In any programming language there are different types of variables, such as integer, real, double precision, complex, character, etc. To estimate the rounding errors of a computation one has to know how these variables, in particular those that represent numbers, are stored on a computer. Although there is no general standard, most workstations and PCs use the same number representation, i.e., most machines that work with 32 bit words (number of bits used for a single precision real number or for an integer) comply with the IEEE convention that will be explained in the following.

<u>Integer</u>

Any integer variable occupies 32 bits of storage. Because of that an integer i can take values $-2^{31} = -2147483648 \leq i \leq 2^{31} - 1 = 2147483647$

$$\underset{31}{\boxed{\ }}\underbrace{\boxed{\qquad\qquad\qquad\cdots\qquad\qquad\qquad}\underset{0}{\boxed{\ }}}_{31\ \text{bits}}$$

Examples:

$$1 = 2^0 = \boxed{0 \qquad\qquad \cdots \qquad\qquad 0\ 1}$$

$$0 = \boxed{0 \qquad\qquad \cdots \qquad\qquad 0}$$

$$57 = 2^5 + 2^4 + 2^3 + 2^0 = \boxed{0 \qquad\qquad \cdots \qquad\qquad 0\ 1\ 1\ 1\ 0\ 0\ 1}$$

$$2^{31} - 1 = \boxed{0\ 1 \qquad\qquad \cdots \qquad\qquad 1}$$

For negative integers the highest, i.e., the 31st bit is 1. But you do not obtain the negative value of an integer by just changing the 31st bit from 0 to 1. Rather the negative value is obtained by flipping all bits and then adding 1. Thus:

$$-1 = \boxed{1 \qquad\qquad \cdots \qquad\qquad 1}$$

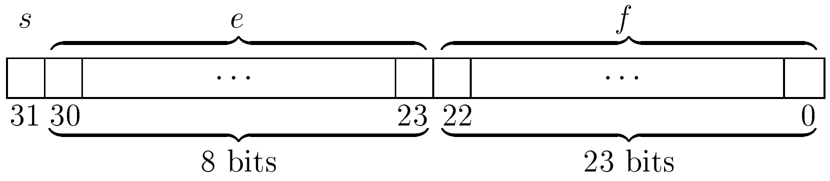
$$-2^{31} = \boxed{1\ 0 \qquad\qquad \cdots \qquad\qquad 0}$$

There is no integer overflow: by subtracting, resp. adding, multiples of 2^{32} the result of any integer operation is converted into an integer within the interval $[-2^{31}, 2^{31}-1]$. This is simply done by discarding the highest bits.

$$2147483647 + 1 = -2147483648 = -2^{31} \text{ or } -1 + 1 = 0\ .$$

Real (single precision)

A single precision real number also occupies 32 bits of storage.



s sign bit, e exponent, f mantissa
The value of the real number is determined as

$$x = (-1)^s \times 2^{e-127} \times 1.f_{22}f_{21}\cdots f_0\ .$$

Here, e is given in binary form, e.g., $e = 0111110 \to e = 126$ and $1.f_{22}f_{21}\cdots f_0$ means $1 + f_{22}2^{-1} + f_{21}2^{-2} + \cdots + f_0 2^{-23} = 1 + \sum_{i=1}^{23} f_{23-i}2^{-i}$. (If e contains only zeros the meaning is slightly different: $x = (-1)^s \times 2^{-127} \times [f_{22}2^0 + f_{21}2^{-1} + \cdots + f_0 2^{-22}]$.)

Examples: (i) $1.0 = \boxed{0011111110\cdots 0}$; $s = 0$, $e = 127$, $f = 0$.

(ii) The largest real number is $r_{\max} = \boxed{0111101\cdots 1} \simeq 3.4 \times 10^{38}$ not $\boxed{01\cdots 1}$! Infinity is represented by $\boxed{0111110\cdots 0}$, all "larger" numbers represent so-called floating point exceptions. These are results of invalid mathematical operations such as division by zero.

(iii) The smallest positive number is $\boxed{0\cdots 01} = 2^{-149} \simeq 1.4 \times 10^{-45}$

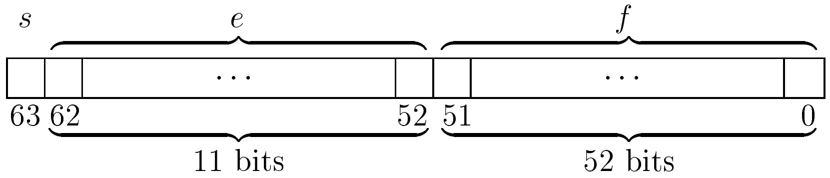
(iv) Negative numbers have $s = 1$.

(v) What is the smallest number $1 + \epsilon$ that is larger than 1? Answer: $1 + \epsilon = \boxed{001111110\cdots 01}$; $\epsilon = 2^{-23} \simeq 1.192 \times 10^{-7}$ is the computer accuracy for single precision real numbers: Any single precision real number has 7 significant digits. Thus, $5.5 + 10^{-7}$ is equivalent to 5.5. The relative rounding error of a real number is therefore

$$\frac{\Delta x}{x} \simeq 10^{-7}\ . \tag{1}$$

Double Precision

Double precision numbers are composed of two computer words, thus occupy 64 bits of storage. The representation is very similar to that of real numbers:



and

$$x = (-1)^s \times 2^{e-1023} \times 1.f_{51}f_{50}\cdots f_0 \ .$$

Infinity corresponds to $e = 1\cdots 1$ and $f = 0\cdots 0$. The largest number is $r_{\max} \simeq 1.8 \times 10^{308}$. The computer accuracy ϵ (again $1+\epsilon$ is the smallest number larger than 1) for double precision numbers is $\epsilon = 2^{-52} \simeq 2.22 \times 10^{-16}$, thus the relative rounding error for double precision real numbers is

$$\frac{\Delta x}{x} \simeq 10^{-16} \ , \tag{2}$$

thus there are 16 significant digits.

Complex variables are pairs of either real (F77, F90) or double precision (F90) variables.

Unsigned integers (C, C++) can take values within a range of 0 to $2^{32}-1$.

Care must be taken if different types of variables are mixed in a computation. If a real/double precision variable or expression is assigned to an integer variable the noninteger part is simply *cut off*. If a real variable that is larger than 2^{31} is assigned to an integer the result is unpredictable! If a double precision variable is converted to single precision the variable is *rounded* to the closest single precision number.

4. Numerical Differentiation

Mathematically the derivative of a function f is defined as

$$\frac{df}{dx} \equiv f'(x) = \lim_{h\to 0} \frac{f(x+h)-f(x)}{h} \ .$$

If possible, we will calculate derivatives analytically and use that function $f'(x)$ in the computer program. Only under rare circumstances, e.g., if f is very complicated and the formula for the derivative is difficult to calculate analytically, will we calculate the derivative numerically using finite differences of the function f. In any case this will need more computer time than the analytical method, thus if CPU-time is a problem, the time you need to calculate a derivative analytically is well spent. Furthermore, numerical accuracy is a real problem when a derivative is calculated numerically as we will see below.

To derive an approximation for the derivative of an arbitrary function f let us first look at the Taylor expansion of that function:

$$f(x+h) = f(x) + f'(x)h + \tfrac{1}{2}f''(x)h^2 + \tfrac{1}{6}f'''(x)h^3 + \ldots = \sum_{\nu=0}^{\infty} \tfrac{1}{\nu!} f^{(\nu)}(x)h^\nu \ . \tag{3}$$

From (3) we get $f(x+h) - f(x-h) = 2hf'(x) + \mathcal{O}(h^3)$. The symbol $\mathcal{O}(\epsilon^n)$ means that this term behaves as ϵ^n as ϵ approaches zero, more accurately, $\mathcal{O}(\epsilon^n)/\epsilon^n$ is finite in the limit $\epsilon \to 0$. Thus we obtain

$$f'(x) = \frac{f(x+h)-f(x-h)}{2h} + \mathcal{O}(h^2) \ . \tag{4}$$

This *symmetric* difference formula is better than $f'(x) = [f(x+h) - f(x)]/h + \mathcal{O}(h)$ or $f'(x) = [f(x) - f(x-h)]/h + \mathcal{O}(h)$. The term $\mathcal{O}(h^2)$ specifies the *systematic* error we make when using Eq. (4). Because of the finite accuracy of real numbers on the computer there will be an additional *rounding* error: Using (1) we obtain

$$\Delta(a-b) = |a|\frac{\Delta a}{|a|} + |b|\frac{\Delta b}{|b|} \simeq (|a| + |b|) \times 10^{-7} \ .$$

Consequently, the relative error $\Delta(a-b)/|a-b|$ can be enormous, if $|a-b| \ll |a| + |b|$. Therefore it is not a good idea when calculating derivatives numerically to make h as small as possible. Instead, there exist an optimal value for h that depends on the function $f(x)$ that minimizes the systematic error in (4) and the rounding error.
Estimation of the rounding error:

$$\epsilon_r = \Delta\left[\frac{f(x+h)}{2h} - \frac{f(x-h)}{2h}\right] \simeq \epsilon\frac{|f(x)|}{h} \xrightarrow[h\to 0]{} \infty \ . \tag{5}$$

Here, ϵ is the accuracy with which f is calculated. This may be comparable to machine precision, i.e., $\epsilon \simeq 10^{-7}$ for single and $\epsilon \simeq 10^{-16}$ for double precision, but it can be larger, if f is very complicated. The truncation error of the Taylor series (3) after neglecting all terms after the quadratic is

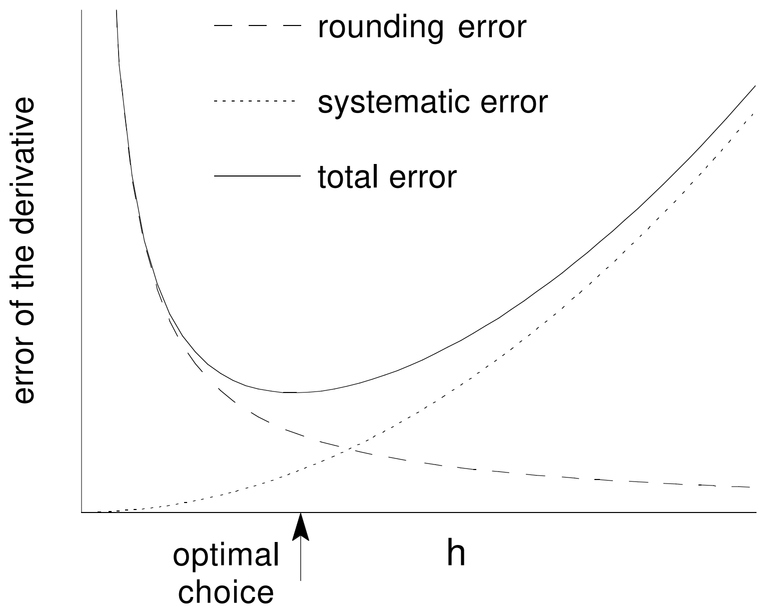
$$R_2 = \tfrac{1}{6}f'''(x+\alpha h)h^3 \ ,$$

where α lies between 0 and 1. Thus the systematic error made by using (4) is approximately

$$\epsilon_s \simeq \tfrac{1}{3}|f'''(x)|h^2 \ . \tag{6}$$

The optimal value for h is obtained by minimizing the sum $\epsilon_s + \epsilon_r$ with respect to h with the result

$$h \simeq \epsilon^{1/3}x_c \quad \text{with} \quad x_c = |f(x)/f'''(x)|^{1/3} \ . \tag{7}$$



x_c is a typical length scale that specifies how fast f changes in the neighborhood of x. It can be hard to estimate, especially if you do not know very much about the function f. In such cases you may take x_c of the same magnitude as x itself. In any case it is a good idea to check the result for the derivative by using a different value for h, e.g., $2h$ or $h/2$. The relative error of the derivative (4) for the value of h given by (7) is

$$(\epsilon_s + \epsilon_r)|f'(x)| \sim \epsilon^{2/3} \; .$$

Thus, it is impossible to reach the machine precision ϵ when calculating a derivative numerically using finite differences. If you need a higher accuracy, it is possible to calculate the derivative for several different values of h and then extrapolate to $h \to 0$, see Numerical Recipes, Chap. 5.7.

Higher order derivatives such as $f''(x)$ or $\partial^2 f(x,y)/\partial x \partial y$ can be calculated in a similar way, also starting from the Taylor expansion. Quite generally, symmetric formulas should be used because of the higher accuracy, e.g.,

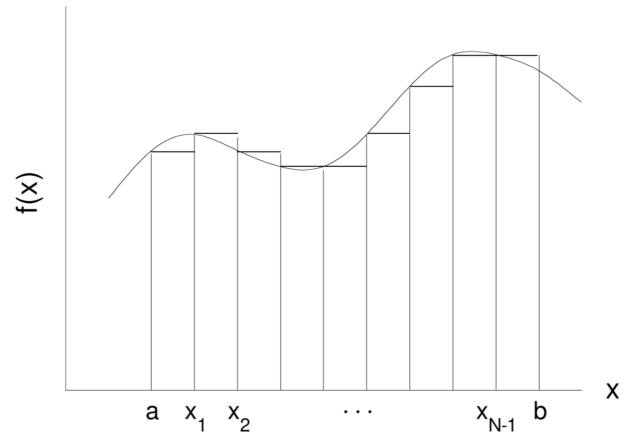
$$f''(x) = [f(x+h) - 2f(x) + f(x-h)]/h^2 + \mathcal{O}(h^2) \; .$$

5. Numerical Integration

5.1 One-Dimensional Integrals

The (Riemann) integral of a function f over the interval $[a,b]$ is defined as

$$I = \int_a^b dx\, f(x) = \lim_{h \to 0} \sum_{j=0}^{N-1} h f(x_j) \quad \text{with} \quad N = (b-a)/h \quad \text{and} \quad x_j = a + jh \; . \qquad (8)$$



The error that is made when (8) is used with some finite value for h can be estimated again using the Taylor expansion for f:

$$f(x) = f(x_j) + (x-x_j)f'(x_j) + \tfrac{1}{2}(x-x_j)^2 f''(x_j) + \tfrac{1}{6}(x-x_j)^3 f^{(3)}(x_j) + \mathcal{O}((x-x_j)^4) \; . \quad (9)$$

Therefore $\int_{x_j}^{x_{j+1}} dx\, f(x) = hf(x_j) + \frac{1}{2}h^2 f'(x_j) + \mathcal{O}(h^3)$, and the error made in the interval is $\simeq \frac{1}{2}h^2 f'(x_j)$. Since there are $N = (b-a)/h$ intervals the total error is of the order $\mathcal{O}(h)$.

We get a much better approximation, if linear interpolation in each interval is used to approximate the function f, $f(x) \simeq f_j + (x-x_j)(f_{j+1}-f_j)/(x_{j+1}-x_j)$, where the abbreviation $f_j \equiv f(x_j)$ has been introduced. This leads to the trapezoidal rule

$$I \simeq I_T(h) = h\left[\tfrac{1}{2}f_0 + f_1 + f_2 + \cdots + f_{N-1} + \tfrac{1}{2}f_N\right] \ . \tag{10}$$

The error is estimated using again Eq. (9) and the corresponding formulae for the derivatives, i.e.,

$$f'(x_j) = [f_{j+1} - f_j]/h - \tfrac{1}{2}hf''(x_j) - \tfrac{1}{6}h^2 f^{(3)}(x_j) + \mathcal{O}(h^3)$$

and

$$f''(x_j) = [f'(x_{j+1}) - f'(x_j)]/h - \tfrac{1}{2}hf^{(3)}(x_j) + \mathcal{O}(h^2) \ .$$

Thus, by integrating Eq. (9) we find

$$\begin{aligned}
\int_{x_j}^{x_{j+1}} dx\, f(x) &= hf_j + \tfrac{1}{2}h^2 f'(x_j) + \tfrac{1}{6}h^3 f''(x_j) + \tfrac{1}{24}h^4 f^{(3)}(x_j) + \mathcal{O}(h^5) \\
&= hf_j + \tfrac{1}{2}h(f_{j+1} - f_j) - \tfrac{1}{12}h^3 f''(x_j) - \tfrac{1}{24}h^4 f^{(3)}(x_j) + \mathcal{O}(h^5) \\
&= hf_j + \tfrac{1}{2}h(f_{j+1} - f_j) - \tfrac{1}{12}h^2[f'(x_{j+1}) - f'(x_j)] + \mathcal{O}(h^5)
\end{aligned}$$

and therefore

$$I = I_T(h) - \tfrac{1}{12}h^2[f'(x_N) - f'(x_0)] + \mathcal{O}(h^4) \ . \tag{11}$$

Surprisingly, the term proportional to h^3 cancels! In fact, it can be shown that the correction terms in Eq. (11) contain only even powers of h. Indeed, (11) shows nothing but the first terms of the Euler-Maclaurin summation formula (see Abramowitz, Stegun, p. 886).

We can use (11) to obtain a formula with even higher accuracy by combining $I_T(h)$ and $I_T(2h)$ so that the term $\sim h^2$ drops out as well:

$$I = I_S(h) + \mathcal{O}(h^4) \quad \text{with} \quad I_S(h) = \tfrac{4}{3}I_T(h) - \tfrac{1}{3}I_T(2h) \ . \tag{12}$$

This approximation is known as Simpson's rule,

$$I_S(h) = \tfrac{1}{3}h\left[f_0 + 4f_1 + 2f_2 + 4f_3 + 2f_4 + \cdots + 2f_{N-2} + 4f_{N-1} + f_N\right] \ . \tag{13}$$

Some remarks:

(i) Do not calculate (13) directly, use (10) and (12).

(ii) Start with a fairly large h (small N) and then iterate (12) by reducing h by a factor of 2 in each iteration until some specified degree of accuracy has been achieved, i.e., until the difference between $I_S(h)$ and $I_S(2h)$ is smaller than some desired accuracy ϵ. Since in each iteration you can use the results of the previous iteration, the number of function evaluations is kept to a minimum:

$$I_T(h) = \tfrac{1}{2}I_T(2h) + h(f_1 + f_3 + f_5 + \ldots + f_{N-3} + f_{N-1}) \ .$$

(iii) Sometimes it is a good idea to make a substitution of the integration variable before doing the numerical integration in order to reduce roundoff errors, e.g., the transformation

$$W = \int_1^{10000} dx\, f(x) \stackrel{z=1/x}{=} \int_{1/10000}^{1} dz\, f(1/z)/z^2$$

reduces the interval of integration and, depending on the function f, may lead to much faster convergence. This transformation also works if $a = -\infty$ or $b = \infty$.

(iv) If the function f has an integrable singularity at one of the endpoints a or b (13) can not be used since f_0 or f_N can not be calculated. In these cases other (so-called open) formulas have to be used that are very similar to (13) but avoid the function evaluations at the endpoints (see Numerical Recipes, Chap. 4.4). However, quite often the singularity can be removed analytically, e.g., consider the following integral with a function g that is non-singular at 1:

$$W = \int_0^1 dx\, \frac{g(x)}{\sqrt{1-x^2}} \ .$$

At $x = 1$ the factor $1/\sqrt{1-x^2} = 1/\sqrt{1+x} \cdot 1/\sqrt{1-x}$ diverges, but this singularity is integrable: $\int_0^1 dx/\sqrt{1-x} = -2\sqrt{1-x}|_0^1 = 2$. The substitution $z = \sqrt{1-x}$ converts the integral into a form that can be evaluated using (13):

$$W = \int_0^1 dz\, \frac{2}{\sqrt{2-z^2}} g(1-z^2) \ .$$

(v) If the integrand f is strongly peaked in some part of the integration interval it may be worth-while to split the interval into several parts and evaluate them separately.

(vi) Under certain circumstances, e.g., if the integrand f is very irregular, has singularities within the interval of integration, etc., the problem can be reformulated into the problem of solving a differential equation: $y(x) = \int_a^x d\tilde{x}\, f(\tilde{x}) \Rightarrow y' = f(x)$ and solve this differential equation with a method that uses a variable step size. This will be discussed later in this course.

5.2 Multidimensional Integrals

There are several problems connected with the evaluation of multidimensional integrals. First, if you need M function evaluations to calculate a one-dimensional integral, you will need of the order of M^d evaluations to do a d-dimensional integral with the same accuracy. Second, the region of integration can be complicated, i.e., the boundaries of the inner integrals may depend on the variables of the outer integrals:

$$I = \int_{x_1}^{x_2} dx \int_{y_1(x)}^{y_2(x)} dy \int_{z_1(x,y)}^{z_2(x,y)} f(x,y,x) \ .$$

Such integrals can be evaluated by repeatedly doing one-dimensional integration, but there is another possibility, namely Monte-Carlo integration. The idea is based on the fact that the integral over some function f is equal to the mean value of that function averaged over the region of integration times the volume of integration,

$$I = \int_V d^d x \, f(\mathbf{x}) = V\langle f \rangle \ . \tag{14}$$

The idea of the Monte-Carlo integration is to choose points $\mathbf{x}_i$ randomly within the volume V and compute the average as $\langle f \rangle = \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_i)$. The error can be estimated by the statistical error $\Delta I = V[\langle f^2 \rangle - \langle f \rangle^2]^{1/2}/N$. If the integral is to be calculated over some complicated region W, define a region V that can be easily calculated and encloses W as closely as possible. Furthermore, define $f(\mathbf{x}) = 0$ for $\mathbf{x}$ outside W. Now Eq. (14) can be applied to this modified function f within the volume V.

The problem remains how to choose the points $\mathbf{x}_i$ in a random way. For this we need random numbers, the generation of which is discussed in Sec. 10 later in this course.

6. Minimization and Root Finding

6.1 Root Finding in one Dimension

6.1.1 Bisection

We want to find the solution of the equation $f(x) = 0$. Assume that we know two points x_1 and x_2 so that $f(x_1) > 0$ and $f(x_2) < 0$ or $f(x_1) < 0$ and $f(x_2) > 0$. Thus, if f is continuous (we will require this for all of this section) there must be a zero between x_1 and x_2. Under these preconditions the zero can be found by bisection: Let us assume that $x_1 < x_2$. Then calculate f at the midpoint of the interval $x_3 = (x_1 + x_2)/2$. If $f(x_3)$ has the same sign as $f(x_1)$ make x_3 your new x_1, otherwise make x_3 your new x_2. Then iterate the process until $x_2 - x_1$ is smaller than some desired accuracy or until $f(x_3)$ is of the order of machine precision. Bisection will always converge, there are methods that converge faster under certain conditions, in other cases they may also be slower. For example, you can choose x_3 by interpolating linearly between x_1 and x_2, $x_3 = [f(x_1)x_2 - f(x_2)x_1]/[f(x_2) - f(x_1)]$, and choose the new interval depending on the sign of $f(x_3)$ as above ("regula falsi").

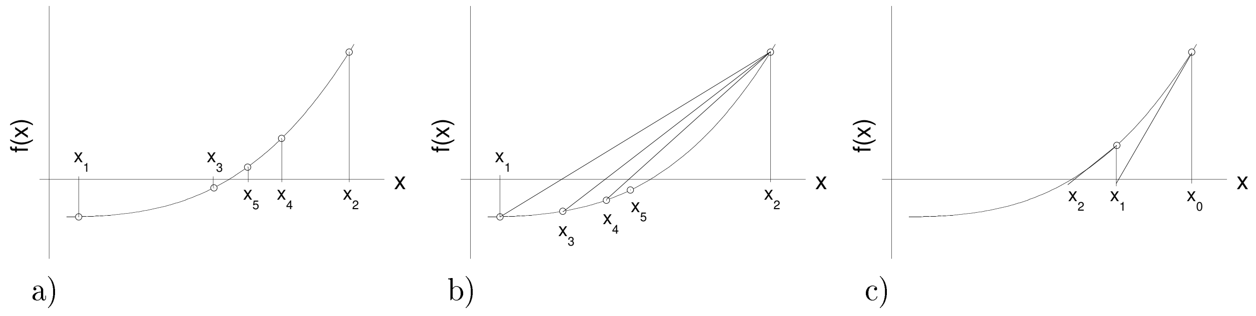


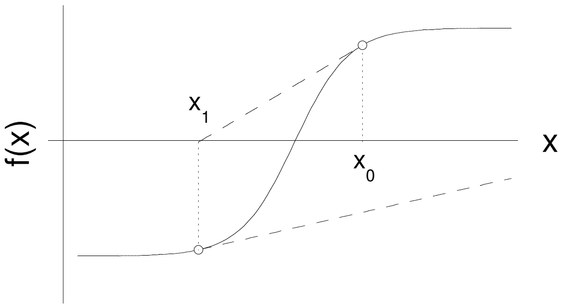
Illustration of different methods of root finding: a) bisection, b) regula falsi c) Newton's method

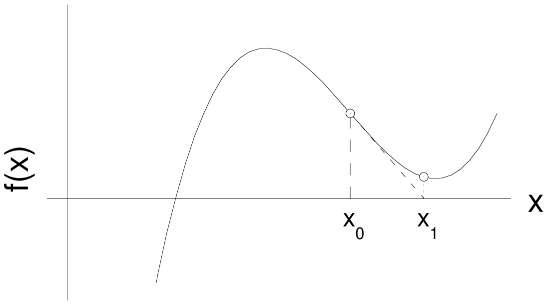
6.1.2 Newton's Method

Newton's method uses only information at one point x in order to approximate the zero of the function f by extrapolating to the zero along the tangent to f at x. The first two terms of the Taylor expansion of f are $f(\overline{x}) \simeq f(x) + (\overline{x} - x)f'(x)$. Solving for $\overline{x}$ with $f(\overline{x}) = 0$ leads to the iteration scheme

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \; . \tag{15}$$

Newton's method converges much faster than the methods mentioned in Sec. 6.1.1, *if your starting point x_0 is close enough to the zero.* Otherwise the method may fail completely. Thus, it is important that you know in which regime the zero can be found, and it is essential that you check after each iteration whether $|f(x_n)|$ actually has decreased. If not, discard the step and switch to a different method, e.g., to bisection, and switch back to Newton afterwards.





Two situations in which Newton's method fails.

6.2 Minimization in one Dimension

Finding the minimum of a nonlinear function of a single variable is very similar to root finding in one dimension, since the position of the minimum of a function $f(x)$ is determined by the equation $f'(x) = 0$. Thus, Newton's method reads

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)} \; . \tag{16}$$

In case $f(x_{n+1}) > f(x_n)$ we again discard the step and switch to a different method. Bisection, however, is not such a good idea for minimizing a function, at least not if you do not want to calculate the derivative f'. The reason is the following: assume we have three points $x_1 < x_2 < x_3$ with $f(x_2) < f(x_1), f(x_3)$. In order to proceed we have to choose a new point either $(x_1 + x_2)/2$ or $(x_2 + x_3)/2$. Approximately half of the time we will choose the wrong interval. Therefore, too many function evaluations will be done. The better alternative is the so-called *golden section search.* Let us assume we are looking for the minimum of the function f in the interval [a,b]. For the moment we will set $a = 0$ and $b = 1$, the generalization to an arbitrary interval is straightforward. We start by calculating the function f at two inner points of the interval $0 < \tau_1 < \tau_2 < 1$. We will choose τ_1 and τ_2 so that the distances

τ_1 and $1 - \tau_2$ from the interval boundaries are the same. If $f(\tau_1) < f(\tau_2)$ the minimum has to be in the interval $[0, \tau_2]$. In order to proceed we have to know the function f at *two* inner points of this new interval $[0, \tau_2]$. However, we already know $f(\tau_1)$. We now choose τ_1^* and $\tau_2^* = \tau_1$ and require that $\tau_1 = \tau_1^*/\tau_2$ and $\tau_2 = \tau_2^*/\tau_2$ in order to keep the points, where the function is to be evaluated, always at the same fractions of the current interval. Thus, we must have $\tau_1 = \tau_2^2$. The symmetry requirement $\tau_1 = 1 - \tau_2$ guarantees that in the case of $f(\tau_1) > f(\tau_2)$ we can choose $[\tau_1, 1]$ as the new interval and $\tau_1^* = \tau_2$ in exactly the same way. The two conditions, $\tau_1 = \tau_2^2$ and $\tau_1 = 1 - \tau_2$, lead to a quadratic equation with the result

$$\tau_2 = \tfrac{1}{2}(\sqrt{5} - 1) \qquad \text{and} \qquad \tau_1 = \tau_2^2 \ . \tag{17}$$

The number $\frac{1}{2}(\sqrt{5} - 1)$ is known as the *golden mean*, hence the name golden section search. Note that in a golden section search the function f is never evaluated at the endpoints of the interval.

For an arbitrary interval $[a, b]$ the points at which the function has to be calculated are $x_1 = a + \tau_1(a - b)$ and $x_2 = a + \tau_2(a - b)$. If $f(x_1) < f(x_2)$ set $b \to x_2$, $x_2 \to x_1$, and $x_1 \to a + \tau_1(b - a)$, otherwise set $a \to x_1$, $x_1 \to x_2$, and $x_2 \to a + \tau_2(b - a)$. These steps are to be iterated until $b - a < \epsilon_1$, or the reduction of f in two consecutive iterations is smaller than ϵ_2 for some predescribed values of ϵ_1, ϵ_2.

6.3 Minimization of a Function of Several Variables

There are several methods of minimizing a nonlinear function of several variables (steepest descent, conjugate gradient methods, etc., see Numerical Recipes, Chaps. 10.4-10.6). Here, we only discuss the Newton-Raphson method which is the generalization of Newton's method to higher dimensions. It has the advantage that it can be generalized to solving sets of nonlinear equations as well.

An extremum of the function $f(\mathbf{x})$, $\mathbf{x} = (x_1, \ldots, x_n)$, is characterized by the condition

$$\nabla f = 0 \qquad \text{or} \qquad \frac{\partial f}{\partial x_i} = 0, \ i = 1, \ldots, n \ . \tag{18}$$

The Taylor expansion of f around some point $\mathbf{x}$ reads

$$f(\mathbf{x} + \mathbf{s}) \ = \ f(\mathbf{x}) + \mathbf{s} \cdot \nabla f(\mathbf{x}) + \tfrac{1}{2}\mathbf{s}\underset{\sim}{A}\mathbf{s} + \ldots \tag{19a}$$

and

$$\nabla f(\mathbf{x} + \mathbf{s}) \ = \ \nabla f(\mathbf{x}) + \underset{\sim}{A}\mathbf{s} + \ldots \ , \tag{19b}$$

where the symmetric matrix $\underset{\sim}{A}$ with elements

$$[\underset{\sim}{A}]_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j} \tag{20}$$

has been introduced. Setting $\nabla f(\mathbf{x}) = 0$ in (19b) leads to the Newton-Raphson method

$$\underset{\sim}{A}\mathbf{s} = -\nabla f \tag{21}$$

and the iteration scheme

$$\mathbf{x}_{\nu+1} = \mathbf{x}_\nu + \mathbf{s} \ . \tag{22}$$

Eq. (21) defines a set of *linear* equations that have to be solved for $\mathbf{s}$. The solution of linear equations will not be discussed in this course. This is in fact a very difficult numerical problem, however almost every numerical library contains routines for the solution of linear equations. Thus, it is a waste of time to program it yourself. Instead, we will learn how to use these libraries, in particular the NAG library, on the UNIX system. The NAG library contains FORTRAN routines of all kinds, thus C programmers have to learn how to use these programs. This is almost essential since there are basically no libraries for C available.

In the neighborhood of a minimum $\underset{\sim}{A}$ is positive definite, i.e., $\mathbf{s}\underset{\sim}{A}\mathbf{s} = -\nabla f \mathbf{s} > 0$. Therefore, the vector $\mathbf{s}$ is in a direction that reduces f. However, if we are still far away from the minimum there is no guarantee that $\underset{\sim}{A}$ is positive definite, thus the Newton-Raphson method in its pure form (21,22) will probably not converge for quite similar reasons as in the one-dimensional case. Hence, again, we have to check after each iteration whether $f(\mathbf{x}_{\nu+1}) < f(\mathbf{x}_\nu)$. If this is the case, the step is accepted, if not we replace Eq. (22) by

$$\mathbf{x}_{\nu+1} = \mathbf{x}_\nu + \lambda \mathbf{s} \tag{23}$$

and minimize $f(\mathbf{x}_\nu + \lambda\mathbf{s})$ with respect to λ. Since this is now a problem of finding the minimum in *one* dimension we can use a golden section search that was discussed in the previous section. The question remains, within which interval should we look for the minimum? The upper limit is clear: $b = 1$ which is the full Newton-Raphson step (22). If we choose $a = 0$ we will run into trouble if $\mathbf{s}\nabla f > 0$, since all values for λ will lead to an increase of f. This can be avoided by choosing $\lambda_1 = a + \tau_1(1-a) = 0$ and determine the lower boundary of the interval as $a = -\tau_1/(1-\tau_1) = -\tau_2$. Since we already know f for $\lambda = 0$ we only have to calculate $f(\mathbf{x}_\nu + \lambda_2\mathbf{s})$ with $\lambda_2 = -\tau_2 + \tau_2(1+\tau_2) = \tau_1$ in order to start the golden section search. The golden section search will be performed until two consecutive values of λ differ by less than ϵ_λ. Do not choose ϵ_λ too small; the golden section search is only needed to approach the minimum, close to the minimum the pure Newton-Raphson step (22) will always be accepted. $\epsilon_\lambda = 10^{-2}$, e.g., is a reasonable choice.

Under very rare circumstances the golden section search will yield a result $\lambda = 0$ or $\lambda = \pm\epsilon_\lambda$. This is extremely unlikely since it means that by chance a direction $\mathbf{s}$ was found that is orthogonal to the gradient. In these cases it may be worthwhile to take a step in the direction of the gradient, $\mathbf{x}_{\nu+1} = \mathbf{x}_\nu - \lambda\nabla f$, again by minimizing f with respect to λ. However, result $\lambda = 0$ obtained from the golden section search is by far more likely to occur, if there is a programming mistake in the routines that calculate the function f and/or the derivatives and the step $\mathbf{s}$. Thus, in the case of $\lambda = 0$ one should carefully check these routines for errors.

The combination of the Newton-Raphson method with the golden section search also allows to treat problems, where some or all of the parameters are confined within certain bounds, e.g., if the function f contains terms like $\ln x_i$ for some value of i, the iteration procedure must stay within a regime $x_i > 0$. If the Newton-Raphson step violates these conditions the golden section search is used and the search interval is limited so that the restrictions on the parameters are satisfied. This can be done by treating all points $\mathbf{x}_\nu$ that violate the constraints as if they correspond to points with $f(\mathbf{x}_\nu) = +\infty$. The same approach is used if one wants to restrict the search for the minimum within some limited area in parameter space. To take into account constraints in this way is a good method, if *the location of the minimum* $\mathbf{x}_{\min}$ *does not lie on the boundary of the area* D *for which* f *is defined*. If $\mathbf{x}_{\min}$ does lie on the boundary of D the method probably will still work, but is very inefficient: In these case the minimum of f is located at a point with $\nabla f(\mathbf{x}_{\min}) \neq \mathbf{0}$, therefore the Newton-Raphson step will almost certainly fail to reduce f even in the vicinity of the minimum and the minimization is done exclusively by the golden section search. For such cases methods that proceed along the gradient of f, like the steepest descent and conjugate gradient methods, are likely to be more efficient.

The Frenkel-Kontorova Model

In this section an application of the minimization method described above will be discussed. Consider the surface of some crystalline material, e.g., iron. We will assume that the surface is perfect, i.e., there are no defects, no steps, no holes, etc., and that the atoms at the surface form a perfect periodic lattice with a lattice constant a. We now deposit a small amount of a *different* material onto this surface with a different lattice constant b. The surface atoms provide a potential for the deposited atoms, also called adatoms, that has the periodicity a of the underlying lattice. Hence, if all adatoms will be positioned at the minimum of this periodic potential they cannot maintain their preferred distance b, but will be compressed (if $b > a$). If the strength of the periodic potential is very large and the misfit $b - a$ is not too big this will be nevertheless the preferred state. However, for larger misfits and/or weaker potentials a preferable configuration of the adatoms consists of regimes in which the adatoms are compressed, but these regimes are separated by defects, so-called dislocations. These are lattice sites, where no adatom is located at a site that corresponds to a minimum of the periodic potential (in the case of $b < a$ dislocations correspond to sites, where there are two adatoms located close to a minimum of the periodic potential), see the figure below. The question that has to be answered is: What is the distance between these dislocations depending on the misfit $b - a$ and the strength u of the periodic potential?

The Frenkel-Kontorova model studies the one-dimensional version of this problem. The interaction between the adatoms is modeled by harmonic springs with an equilibrium length b and a spring constant κ. The periodic potential is taken to be a simple cosine function. Hence, the total energy of the system can be expressed in the following way:

$$V = \sum_j \left[\tfrac{1}{2}\kappa \left(x_{j+1} - x_j - b\right)^2 - \tfrac{1}{2}b^2 + u\left(1 - \cos(2\pi x_j/a)\right)\right] , \qquad \text{(FK 1)}$$

where x_j is the position of the j-th adatom and the sum runs over all adatoms. The zero of the energy scale in (FK 1) is set so that the state $x_j = ja$, where all particles sit in the minimum of the periodic potential, has energy $V = 0$. The model (FK 1) contains 4 parameters a, b, κ, u. However, two parameters, a and κ, can be eliminated by changing the length and energy scales. It is also advantageous to introduce new variables $\varphi_j = 2\pi(x_j/a - j)$ that measure the position of the j-th particle relative to the j-th minimum of the periodic potential. After these changes the total energy depends only on two parameters, the strength of the potential u and the misfit $\Delta = \frac{2\pi}{a}(b-a)$,

$$V = \sum_j \left[\tfrac{1}{2}\left(\varphi_{j+1} - \varphi_j - \Delta\right)^2 - \tfrac{1}{2}\Delta^2 + u\left(1 - \cos\varphi_j\right)\right] , \qquad \text{(FK 2)}$$

The stable configurations of the Frenkel-Kontorova model correspond to sets of variables φ_j that minimize the total energy V (FK 2). These are the ground states of the model. If u and Δ are changed, in general the ground state will change as well. The *phase diagram* specifies the ground state of the Frenkel-Kontorova model for each pair of values (u, Δ). Two extreme cases can be specified immediately: For $u = 0$ the distances between the particles will be equal to the length of the springs, i.e., $\varphi_{j+1} - \varphi_j = \Delta$. On the other hand, for large strengths u of the potential all particles will sit at the minimum of the potential, $\varphi_j = 0$. For other values of u and Δ the ground states have to be calculated numerically by minimizing the total energy V.

In a numerical calculations it is impossible to study infinite lattice sizes. For finite lattices, however, boundary conditions have to be specified. Two cases are of interest:

a) free boundary conditions: The two particles at the ends of the chain have only one neighbor, thus there are N particles, but only $N-1$ springs. The total energy in this case is

$$V = \sum_{j=1}^{N-1} \left[\tfrac{1}{2}\left(\varphi_{j+1} - \varphi_j - \Delta\right)^2 - \tfrac{1}{2}\Delta^2\right] + \sum_{j=1}^{N} \left[u\left(1 - \cos\varphi_j\right)\right] . \qquad \text{(FK 3)}$$

This kind of boundary conditions model an island of adatoms on the surface of a crystal.

b) periodic boundary conditions: We assume that after a finite number N of adatoms the whole sequence of positions φ_j will repeat itself, e.g., if particle 1 sits exactly in the minimum of the periodic potential, then particle $N+1$ sits at the minimum as well. In this case we can identify particle 1 and particle $N+1$, thus we close the chain by connecting particle 1 and particle N with a spring as well. However, we must specify how many minima $N+p$ of the periodic potential are spanned by a chain of length N, since this will determine the boundary conditions: $\varphi_{N+1} = \varphi_1 + 2\pi p$, p being an integer. The total energy for periodic boundary condition is therefore given by

$$\begin{aligned} V = &\sum_{j=1}^{N-1} \left[\tfrac{1}{2}\left(\varphi_{j+1} - \varphi_j - \Delta\right)^2 - \tfrac{1}{2}\Delta^2 + u\left(1 - \cos\varphi_j\right)\right] \\ &+ \tfrac{1}{2}\left(\varphi_1 + 2\pi p - \varphi_N - \Delta\right) - \tfrac{1}{2}\Delta^2 + u\left(1 - \cos\varphi_N\right) . \end{aligned} \qquad \text{(FK 4)}$$

The only solution for $p = 0$ is $\varphi_j = 0$, $j = 1, \ldots, N$, i.e., all particles sit at adjacent minima of the periodic potential. Periodic boundary conditions model a situation, where the surface is completely covered with adatoms. For a fixed pair of parameters (u, Δ) the minimization of (FK 4) will give a result that depends on the ratio p/N with $2\pi p/N$ being the average value for $\varphi_{j+1} - \varphi_j$ for that particular solution. To determine the absolute minimum of V one has to calculate V for several values of p/N and plot the energy per particle V/N as a function of p/N. The minimum of that curve gives the preferred value for p/N that characterizes the ground state of the model for the given values of u and Δ.

The so determined states for periodic boundary conditions all have rational wavenumbers p/N, these states are called *commensurate* solutions, since the periodicity of the solution is commensurate with the periodicity of the lattice. In a numerical calculation no other states can be determined. However, one may ask whether there exist so-called incommensurate ground states, i.e., ground states with an irrational wave number p/N. For the Frenkel-Kontorova model it has been proven analytically that below a certain critical strength $u_c(\Delta)$ of the potential there do exist incommensurate ground states, and furthermore the commensurate ground states do not fill up the phase diagram (the number of points (u, Δ) for which the ground states are incommensurate, form a set of measure larger than zero). It is clear that such states must exist for $u = 0$, however, it is nontrivial that incommensurate states still exist for $u > 0$. For $u > u_c(\Delta)$ these incommensurate solutions cease to exist, or, more precisely, the number of points (u, Δ), where the ground state is incommensurate, form a set of measure zero. It is possible to approximate incommensurate states by commensurate solutions with a large periodicity N. Such a typical solutions is shown in the figure below: For fairly large regimes the values of φ_j are close to multiples of 2π. These regimes are separated by defects or dislocations, where the values of φ_j changes by 2π over only a few values of j. For $u < u_c(\Delta)$ the distance of these dislocations is determined by the wave number p/N, and there are no preferred locations for the defects on the lattice. However, for $u > u_c(\Delta)$ the dislocations get pinned to the lattice, i.e., the center of the dislocation always is at a lattice site or exactly between two lattice sites. Consequently, the periodicity of such a solution must be commensurate with the periodicity of the lattice. For these reasons the transition at $u_c(\Delta)$ is called a *pinning transition*. One may ask, whether there exist any other solutions. In particular, are there ground states for which the positions of the dislocations form a chaotic sequence? This question can be answered again analytically: there are no such ground states; all ground states of the Frenkel Kontorova model are either commensurate or incommensurate (the spectrum of the fourier transform is discrete).

For small values of Δ the ground state is the trivial solution $\varphi_j = 0$, $j = 1, \ldots, N$. At some value Δ for a given fixed value of u this state will no longer be the ground state of the Frenkel-Kontorova model. What is the wave number p/N of the first commensurate state with $\varphi_j \neq 0$ for at least some j between 1 and N? Answer: As Δ is increased the wave number that characterizes the ground state runs through *all* rational numbers p/N ! This behavior is called "devil's staircase". For $u > u_c(\Delta)$ the staircase is called complete, otherwise incomplete.

6.4 Solution of a Set of Nonlinear Equations

Finding a solution to a set of nonlinear equations

$$\mathbf{F}(\mathbf{x}) = 0 \qquad \text{with} \qquad \mathbf{F} = (F_1, \ldots, F_n) \ , \quad \mathbf{x} = (x_1, \ldots, x_n) \tag{24}$$

is a much harder problem than finding the minimum of a nonlinear function. In case the Newton-Raphson step (22) was not successful we always could do a line search in order to *reduce the function* f. When looking for the solution of a nonlinear set of equation such criteria do not exist. We can collapse the set of equations (24) to a single nonlinear function

$$f = \tfrac{1}{2}\mathbf{F} \cdot \mathbf{F} \ , \tag{25}$$

but then the solutions $\mathbf{F} = 0$ correspond to the *absolute* minimum of f and not just some local minimum usual minimization routines will converge to. Thus, there is no guarantee that we will actually find solutions of (24) by minimizing (25). In practice the best way is to solve (24) by the Newton-Raphson method

$$\begin{aligned} \underset{\sim}{A}\mathbf{s} &= -\mathbf{F}(\mathbf{x}) \qquad \text{with} \qquad [\underset{\sim}{A}]_{ij} = \frac{\partial F_i}{\partial x_j} \ , && \text{(26a)} \\ \mathbf{x}_{\nu+1} &= \mathbf{x}_\nu + \mathbf{s} && \text{(26b)} \end{aligned}$$

and use a golden section search $\mathbf{x}_{\nu+1} = \mathbf{x}_\nu + \lambda\mathbf{s}$ aimed at minimizing f only if the Newton-Raphson step (26b) fails to reduce f. Thus, we are still trying to solve (24), we are not using a Newton-Raphson method to minimize f. In fact the golden section search is simpler than in the previous section since the vector $\mathbf{s}$ obtained from (26a) will *always* point in a direction along which f is reduced: $\nabla f \cdot \mathbf{s} = (\mathbf{F}\underset{\sim}{A}) \cdot (-\underset{\sim}{A}^{-1}\mathbf{F}) = -\mathbf{F}^2 < 0$. Thus, we can limit the golden section search to the interval $[0, 1]$. Nevertheless, the resulting program is very similar to that of the previous section.

6.5 Other Methods

There are many different methods that can be used to find the minimum of a nonlinear function of several variables, in particular, methods that do not require the calculation of second derivatives, e.g., steepest descent and conjugate gradient methods. We also did not discuss the problem of linear optimization, where the function f is a linear function of the variables, the values of which are restricted by several constraints. See Numerical Recipes, Ch. 10.8 for details on these methods. A totally different method called "simulated annealing" uses a relaxation scheme that is similar to methods used in Monte-Carlo simulations that are discussed later in this course. In short, the function to be minimized corresponds to the energy in a Monte-Carlo simulation, and one introduces 'temperature' as an artificial parameter. At higher 'temperatures' the system can explore the full parameter space by jumping over the energy barriers that separate different minima. Then the temperature is reduced step by step so that less and less transitions between minima occur until one finally at zero temperature ends up (hopefully) in the absolute minimum of the energy landscape.

7. Data Analysis

In this section we will be concerned with the problem of extracting information from a set of data points regardless of whether the data were obtained in a real experiment or a computer simulation. Thus, we assume that somebody gave us a set of data $x_1, x_2, \ldots, x_N$. The probability density $p(x)$ describes the occurrence of a particular value x in such a measurement: $p(x)dx$ is the probability of finding a value $\overline{x} \in [x, x+dx]$. From this probability density all quantities of interest can be calculated, e.g., the mean value $\mu = \langle x_i \rangle = \int_{-\infty}^{\infty} dx\, xp(x)$ or the variance $\sigma^2 = \langle x_i^2 \rangle - \langle x_i \rangle^2 = \int_{-\infty}^{\infty} dx\, (x^2 - \mu^2)p(x)$; in general $\langle f(x_i) \rangle = \int_{-\infty}^{\infty} dx\, f(x)p(x)$. Since the probability density is usually unknown, these quantities have to be estimated from the measured data. There are two mathematical theorems, the law of large numbers and the central limit theorem, that provide the necessary information.

7.1 Central Limit Theorem

Let $x_1, x_2, \ldots, x_N$ be a sequence of independent random variables with a common probability distribution $p(x)$ and expectation values $\langle x_i \rangle = \mu$. Let $S_N = \sum_{i=1}^{N} x_i$.
Law of large numbers: For all $\epsilon > 0$ the probability of $S_N/N - \mu$ being larger than ϵ approaches 0 in the limit $N \to \infty$.
In other words: The average S_N/N converges to the mean μ of the probability distribution $p(x)$ for large N.

An even stronger statement can be made, if we additionally assume that the variance σ^2 is finite:
Central Limit Theorem: In the limit $N \to \infty$ the probability

$$\Pr\left[\frac{S_N - N\mu}{\sigma\sqrt{N}} < X\right] \longrightarrow \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{X} dy\, e^{-y^2/2} \equiv P(X)$$

converges for all X to the normal or Gaussian distribution $P(X)$.

This theorem gives an estimate for the probability that the discrepancy $|S_N/N - \mu|$ is larger than $\sigma/\sqrt{N}$. Furthermore it states that for large N the mean μ and the variance σ^2 fully determine the probability density of the average $\overline{x} = S_N/N$

$$\overline{p}(\overline{x}) \xrightarrow{N\to\infty} \frac{\sqrt{N}}{\sigma\sqrt{2\pi}} e^{-\frac{N(\overline{x}-\mu)^2}{2\sigma^2}} , \qquad (27)$$

i.e., regardless of the probability density $p(x)$ of the individual variables x_i, the average $\overline{x} = S_N/N$ follows in the limit of large N a Gaussian probability distribution with mean $\mu = \overline{x}$ and variance σ^2/N.
Examples:
(a) Rolling the die: $x_i = i$, $i = 1, \ldots, 6$

$$\Longrightarrow \langle x_i \rangle = \mu = 3.5 \ , \ \sigma^2 = (1^2 + \ldots + 6^2)/6 - (3.5)^2 = \tfrac{35}{12}.$$

Central limit theorem:

$$\Pr\left[|S_N - 3.5N| < \alpha\sqrt{35N/12}\right] \simeq P(\alpha) - P(-\alpha) = \frac{1}{2\pi} \int_{-\alpha}^{\alpha} dy\, e^{-y^2/2} \ .$$

$N = 1000$, $\alpha = 1$: $\Pr[3446 < S_N < 3554] \simeq 0.68$; $\alpha = 2$: $\Pr[3392 < S_N < 3608] \simeq 0.95$ (in other words: the measured value of $\overline{x} = S_N/N$ will fall within $\pm 2\sigma$ of the true value μ 95% of the time).

(b) Sampling: Suppose that in a population of M families there are M_k families with exactly k children, $k = 0, 1, \ldots$ and $\sum_k M_k = M$. For a family chosen at random, the number of children is a random variable which assumes the value ν with probability $p_\nu = M_\nu/M$. We now make a measurement by selecting randomly N families. Then the sample average S_N/N is likely to be near $\mu = \sum_{\nu=1}^N \nu p_\nu = \sum_{\nu=1}^N \nu M_\nu/M$, namely the population average. If it is desired that there be a probability of 0.99 or better that the sample average S_N/N differs from the unknown mean μ by less that 1/10, then the sample size N should be such that $\Pr[|(S_N - N\mu)/N| < 1/10] \geq 0.99$. From $P(x) - P(-x) = 0.99$ we obtain $x \simeq 2.58$, hence N should satisfy $2.58\sigma/\sqrt{N} \leq 1/10$ or $N \geq 666\sigma^2$. Thus, this result depends crucially on an estimate of the variance σ^2.

How do we estimate μ and σ^2, and what is the error of the estimate of μ?
Result:

$$\overline{x} \simeq \mu \ , \tag{28a}$$

$$\langle \overline{x}^2 \rangle - \langle \overline{x} \rangle^2 = \frac{1}{N(N-1)} \sum_{i=1}^N \left\langle x_i^2 - \overline{x}^2 \right\rangle = \frac{1}{N}\sigma^2 \ . \tag{28b}$$

The first equation follows from the law of large numbers. The variance of $\overline{x}$ is obtained from

$$\begin{aligned}
\langle \overline{x}^2 \rangle &= \int_{-\infty}^{\infty} dx_1 \cdots \int_{-\infty}^{\infty} dx_N p(x_1) \times \cdots \times p(x_N) \left(\frac{1}{N} \sum_{i=1}^N x_i \right)^2 \\
&= \int_{-\infty}^{\infty} dx_1 \cdots \int_{-\infty}^{\infty} dx_N p(x_1) \times \cdots \times p(x_N) \left[\frac{1}{N^2} \sum_{i \neq j} x_i x_j + \frac{1}{N^2} \sum_i x_i^2 \right] \\
&= \frac{N(N-1)}{N^2} \langle x \rangle^2 + \frac{N}{N^2} \langle x^2 \rangle \\
&= \frac{N-1}{N}\mu^2 + \frac{1}{N}(\sigma^2 + \mu^2) = \mu^2 + \frac{1}{N}\sigma^2 \ ,
\end{aligned}$$

where we have used the identity $\langle x^2 \rangle = \langle (x - \langle x \rangle)^2 \rangle + \langle x \rangle^2$. Now we have

$$\sum_{i=1}^N \left\langle x_i^2 - \overline{x}^2 \right\rangle = N(\sigma^2 + \mu^2) - N\mu^2 - \sigma^2 = (N-1)\sigma^2 \quad \Rightarrow \quad \sigma^2 = \frac{1}{N-1} \sum_{i=1}^N \left\langle x_i^2 - \overline{x}^2 \right\rangle \ ,$$

and therefore

$$\mathrm{Var}(\overline{x}) = \langle \overline{x}^2 \rangle - \langle \overline{x} \rangle^2 = \mu^2 + \frac{1}{N}\sigma^2 - \mu^2 = \frac{1}{N}\sigma^2 = \frac{1}{N(N-1)} \sum_{i=1}^N \left\langle x_i^2 - \overline{x}^2 \right\rangle \ .$$

(Quote from Numerical Recipes: "If the difference between N and $N-1$ ever matters to you, then you are probably up to no good anyway — e.g., trying to substantiate a questionable hypothesis with marginal data."). The derivation above did not make use of the central limit theorem and is therefore valid for all probabilities $p(x)$ with mean μ and variance σ^2.

7.2 Least-Squares Fits

In this chapter we are dealing with the problem that we made a measurement (experiment or computer simulation) and we want to compare the data with a theory or model that depends on some parameters. We want to verify whether the theory applies to our experiment and, if that is the case, what are the parameters? Since the data are not exact because of "noise" they will never fit the theory exactly. Thus, we need some criteria for answering the question "How good is the fit?". Hence, the fitting procedure should provide (i) the parameters, (ii) error estimates on the parameters, and (iii) a statistical measure of the goodness of the fit.

Suppose we are fitting N data points (x_i, y_i), $i = 1, \ldots, N$, to a model with M parameters, $a_1, \ldots, a_M$. Thus the model provides some function $f(x; a_1, \ldots, a_M)$ that predicts, in which way the measured data y_i should behave as a function of x. Now, suppose that each data point y_i has a measurement error that is independently random and is Gaussian distributed with variance σ_i^2 around the true model $f(x; \hat{\mathbf{a}})$, where all the parameters have the correct values $\hat{\mathbf{a}}$. Then the probability or likelihood of measuring this data set is the product of the probabilities of measuring each data point,

$$L = \prod_{i=1}^{N} p(y_i) = \prod_{i=1}^{N} \frac{1}{\sigma_i \sqrt{2\pi}} \exp\left[-\frac{(y_i - f(x_i; \mathbf{a}))^2}{2\sigma_i^2}\right] = c_\sigma \exp\left[-\frac{1}{2}\sum_{i=1}^{N}\left(\frac{y_i - f(x_i; \mathbf{a})}{\sigma_i}\right)^2\right] \quad (29)$$

with $c_\sigma = \prod_{i=1}^{N} \frac{1}{\sigma_i\sqrt{2\pi}}$. By maximizing the probability (29) we obtain the most likely set of parameters. Since c_σ does not depend on $\mathbf{a}$, maximizing L is equivalent to minimizing the argument of the exponential,

$$\chi^2 = \sum_{i=1}^{N}\left[\frac{y_i - f(x_i; a_1, \ldots, a_M)}{\sigma_i}\right]^2 , \quad (30)$$

with respect to the parameters $a_1, \ldots, a_M$. The minimization of (30) is called a *least-squares fit*. From the previous section we know that any probability distribution of an average will converge to a gaussian distribution, *if the number of measurements is large*. However, in real life the assumption of a Gaussian distribution may not be justified, e.g., sometimes you have outliers in your experimental data that are way off the theoretical curve. These outliers will completely spoil the least-squares fit. This must be kept in mind when estimating the size of errorbars for parameters obtained from a least-squares fit.

The method of minimizing (30) is very similar to the Newton-Raphson method discussed in Sec. 6.3, but because of the special form of the function χ^2 and the random contributions to χ^2 stemming from measurement errors the algorithm used for least-squares fits, called Levenberg-Marquardt method, differs in some points from the Newton-Raphson method. The gradient of χ^2 with respect to the parameters $\mathbf{a} = (a_1, \ldots, a_M)$ is given by

$$\frac{\partial \chi^2}{\partial a_n} = -2\sum_{i=1}^{N} \frac{[y_i - f(x_i; \mathbf{a})]}{\sigma_i^2}\frac{\partial f(x_i; \mathbf{a})}{\partial a_n} , \qquad n = 1, \ldots, M , \quad (31)$$

and the matrix containing the second derivatives is

$$\frac{\partial \chi^2}{\partial a_m \partial a_n} = 2\sum_{i=1}^{N} \frac{1}{\sigma_i^2}\left[\frac{\partial f(x_i;\mathbf{a})}{\partial a_m}\frac{\partial f(x_i;\mathbf{a})}{\partial a_n} - [y_i - f(x_i;\mathbf{a})]\frac{\partial^2 f(x_i;\mathbf{a})}{\partial a_m \partial a_n}\right] . \tag{32}$$

It is convenient to remove the factors of 2 by defining

$$\beta_n = -\frac{1}{2}\frac{\partial \chi^2}{\partial a_n} \qquad \text{and} \qquad \alpha_{mn} = \frac{1}{2}\frac{\partial^2 \chi^2}{\partial a_m \partial a_m} . \tag{33}$$

The matrix α is called the curvature matrix. Thus, the change $\delta\mathbf{a}$ of the parameters in an iteration of the Newton-Raphson method is obtained by solving the set of linear equations

$$\sum_{m=1}^{M} \alpha_{nm}\delta a_m = \beta_n . \tag{34}$$

In the Levenberg-Marquardt algorithm the second derivatives in Eq. (32) are neglected in the definition of the matrix (α_{nm}). The reason is that these terms are multiplied by $y_i - y(x_i;\mathbf{a})$, i.e., for a reasonable model this is just the measurement error of the i-th data point. Since these errors should be random, the terms containing the second derivatives should cancel when summed over i. Thus, we will use the simpler form

$$\alpha_{mn} = \sum_{i=1}^{N} \frac{1}{\sigma_i^2}\frac{\partial f(x_i;\mathbf{a})}{\partial a_m}\frac{\partial f(x_i;\mathbf{a})}{\partial a_n} . \tag{35}$$

Furthermore, the Newton-Raphson step (34) has to be replaced by something else as long as we are far away for the minimum. The problem is that, e.g., the gradient used in the steepest-decent methods

$$\delta a_n = const. \times \beta_n \tag{36}$$

does not provide information about the factor *const.* that determines the size of the step. The Levenberg-Marquardt algorithm combines the steepest-descent method and the Newton-Raphson method to achieve a better convergence. It estimates the constant in (36) using the diagonal elements α_{nn} and some overall constant λ,

$$\delta\alpha_n = \frac{1}{\lambda a_{nn}}\beta_n . \tag{37}$$

We now are able to combine the two methods (34) and (37) by defining a modified matrix (α'_{mn}) with

$$\alpha'_{mn} = \begin{cases} (1+\lambda)\alpha_{nn} & \text{for } n = m \\ \alpha_{mn} & \text{for } n \neq m \end{cases} \tag{38}$$

and then use

$$\sum_{m=1}^{M} \alpha'_{nm}\delta a_m = \beta_n . \tag{39}$$

For $\lambda \gg 1$ this corresponds to the steepest-descent method (37), whereas for small λ it crosses over to the Newton-Raphson method (34). In principle you can fiddle around with the matrix (α_{mn}), since the final set of parameters is determined by the condition $\beta_n = 0$ independently of the definition of α_{mn}. The iteration now proceeds in the following way:

(i) make an initial guess of the parameters $\mathbf{a}$ and calculate $\chi^2(\mathbf{a})$,

(ii) set λ to 10^{-3} (or something like that),

(iii) solve the linear equations (39) and compute $\chi^2(\mathbf{a} + \delta\mathbf{a})$,

(iv) if $\chi^2(\mathbf{a} + \delta\mathbf{a}) \geq \chi^2(\mathbf{a})$, increase λ by a factor of 10 and go back to (iii); if $\chi^2(\mathbf{a} + \delta\mathbf{a}) < \chi^2(\mathbf{a})$, decrease λ by 10, update the parameters $\mathbf{a} + \delta\mathbf{a} \to \mathbf{a}$, and go back to (iii).

(v) iterate (iii), (iv) until $|\delta\chi^2|/\chi^2(\mathbf{a}) < \epsilon_\chi$ and $|\boldsymbol{\beta}| < \epsilon_\beta$ for some desired accuracies ϵ_χ, ϵ_β and $\delta\chi^2 = \chi^2(\mathbf{a} + \delta\mathbf{a}) - \chi^2(\mathbf{a})$. Numerical Recipes tells you that you should not terminate the iteration after a step with $\delta\chi^2 \geq 0$. Following that advice is probably not a good idea. If you happen to choose ϵ_χ too small the iteration will converge to the minimum of χ^2 without satisfying the condition $|\delta\chi^2|/\chi^2 < \epsilon_\chi$. In the following iterations the algorithm will then always increase the value of λ since χ^2 can no longer be reduced. This results in a huge value for λ that seems to indicate that the algorithm did not converge. Therefore, even in the case of $\delta\chi^2 > 0$ check whether $|\boldsymbol{\beta}| < \epsilon_\beta$. If it is, terminate the iteration and return the old values $\mathbf{a}$ and $\chi^2(\mathbf{a})$ as the solution.

(vi) Set $\lambda = 0$ and compute the covariance matrix c in order to calculate standard deviations of the parameters, correlations, etc., see below. The covariance matrix c is the inverse of the curvature matrix α.

It does not make sense to iterate these equations until machine precision is reached since χ^2 is not determined to that accuracy because of measurement errors. Thus you may well stop as soon as χ^2 does not decrease significantly anymore. Usually, relative changes $|\chi^2(\mathbf{a}+\delta\mathbf{a}) - \chi^2(\mathbf{a})|/\chi^2(\mathbf{a})$ that are less than $\epsilon_\chi \simeq 10^{-3}$ are hardly significant.

This brings us to the questions "what is a good fit?" and "what are the errorbars on the calculated parameters?" If each of the data points y_i is distributed according to a Gaussian probability distribution with variances σ_i^2, it can be shown (see section 7.2.2) that the probability of obtaining a result χ^2_{min} from the least-squares fit with $\chi^2_{\text{min}} < \chi^2$ is given by the so-called χ^2-probability function with ν degrees of freedom

$$P(\chi^2|\nu) = \left[2^{\nu/2}\Gamma(\nu/2)\right]^{-1} \int_0^{\chi^2} dy\, y^{\frac{\nu}{2}-1} e^{-y/2} \;, \tag{40}$$

with $\nu = N - M$. This statement is correct under the assumption that the measurement errors are sufficiently small so that at the minimum of χ^2 the deviations of the parameters $\mathbf{a}$ from the true values $\hat{\mathbf{a}}$ are so small that the fitting function can be replaced by the linear approximation

$$f(x_i; \mathbf{a}) = f(x_i; \hat{\mathbf{a}}) + \sum_{n=1}^{M} (a_n - \hat{a}_n) \left. \frac{\partial f(x_i; \mathbf{a})}{\partial a_n} \right|_{\mathbf{a}=\hat{\mathbf{a}}}$$

that neglects higher order terms in the Taylor expansion of f. The probability function $P(\chi^2|\nu)$ depends on only one parameter ν. In particular, the mean of this distribution is ν itself and the variance is 2ν; in the limit of large ν $P(\chi^2|\nu)$ becomes the Gaussian distribution: $P(\chi^2|\nu) \approx P(x)$ with $x = (\chi^2 - \nu)/\sqrt{2\nu}$. Since the mean of the χ^2 distribution function is

$\nu = N - M$ this fact can be used as a criterion for the goodness of the fit: a reasonable fit should have $\chi^2_{\text{min}} \simeq N - M$. However, the validity of this criterion depends crucially on the estimates of the variances σ_i^2. If the σ_i^2 are poorly estimated or even arbitrarily set to 1, the value of χ^2_{min} is meaningless. In these cases you can use the relation $\chi^2_{\text{min}} = N - M$ to determine an overall factor of the σ_i^2: If the least-squares fit using variances $\tilde{\sigma}_i^2$ yielded a result $\tilde{\chi}^2$, the variances $\sigma_i^2 = [\tilde{\chi}^2/(N-M)]\tilde{\sigma}_i^2$ would give a result of $\chi^2 = N - M$. If these values for σ_i^2 turn out to be unreasonable, this is an indication that the quality of the fit is not very good. Clearly, this method prohibits a direct evaluation of the goodness of the fit.

However, quite often you will be able to obtain a good fit to your data, but the parameters $\mathbf{a}$ still fluctuate quite strongly without changing the value of χ^2 significantly. This occurs if the minimum of χ^2 sits in a very flat valley, so that by changing the parameters in a particular way you wander around in that valley without changing χ^2. This kind of degeneracy is an indication that the model contains too many parameters, or, conversely, your data set is not sufficient to determine all the parameters independently. A trivial example: Assume that somebody tries to fit a set of data y_i to the function $f(x_i; a_1, a_2, a_3) = a_1 \exp[-a_2 x_i + a_3]$. Naturally, he will not be able to determine a_1 and a_3 independently, since all fits with the same value for $a_1 e^{a_3}$ will give the same χ^2. Unfortunately, these degeneracies will occur even if they are not built into the fitting function $f(x; \mathbf{a})$ in such an obvious way. In these cases the number of parameters must be reduced, or you have to concede that the available data do not allow to determine the parameters with the desired accuracy.

Mathematical predictions for the errorbars on the parameters a_i can only be derived if several preconditions apply (see the following subsection). Most importantly, it is assumed that only statistical errors are present. Quite often this is incorrect. Systematic errors appear because the "true fitting function" is unknown. For example the chosen fitting function f is not a perfect model for your data y_i over the full range of the x_i. So-called crossover effects are a typical example that cause such systematic errors. Also the assumption that the measurement data y_i are Gaussian distributed may not be valid. If systematic errors are more important than statistical errors, the standard deviations of the parameters that are derived in the following subsection are meaningless. Indications for systematic errors can be obtained from the residuals $R_i = y_i - f(x_i; \mathbf{a})$. If only statistical errors are present, the R_i are randomly distributed around zero. If systematic errors are dominant, there are intervals in x, where *all* R_i are positive, followed by an interval, where all R_i are negative, i.e., the R_i have an oscillatory behavior as a function of x.

For practical purposes you can estimate the errorbars on your parameters by changing the initial guesses for $\mathbf{a}$ and compare the results after the changes in χ^2 are smaller than ϵ_χ. You can also fix some of the parameters and minimize χ^2 only with respect to the remaining parameters. To test how sensitively the parameters depend on the model one can make "supposedly insignificant" changes to the function f and repeat the least-squares fit. The obtained differences in the parameter values yield in many cases by far better (most often larger!) error estimates of the parameters than the formulae presented in Sec. 7.2.1. In fact, if this leads to large changes in the parameters your model is either ill-defined or your data set is too small. In any case it is a good idea to perform the least-squares fit under various conditions in order to get a feeling for the accuracy of the parameter estimates.

Although the formulae for the variances of the parameters derived in the following subsection are not valid, if systematic errors are present, the results for the correlation coefficients

$$r_{kl} = \frac{c_{kl}}{\sqrt{c_{kk}c_{ll}}} \tag{41}$$

are still helpful to judge the quality of your fit. Here the matrix c is the inverse of the curvature matrix α. If only statistical errors are present, c_{nn} turns out to be the variance of the parameter a_n, whereas c_{kl} are the covariances of the parameters a_k and a_l, see Sec. 7.2.1. The correlation coefficients r_{kl} take values between -1 and 1 and indicate correlations between the parameters a_k and a_l: $|r_{kl}| = 1$ implies a linear relation between the parameters a_l and a_k. If $r_{kl} = 1$ a_l increases with a_k, whereas for $r_{kl} = -1$ it decreases. Hence large values of $|r_{kl}| \gtrsim 0.9$ indicated degeneracies in your model as discussed above. In this case you cannot fit all parameters independently and therefore the number of parameters should be reduced.

If your data y_i are spread out over a large interval of several orders of magnitude it may be a good idea to fit the logarithms $\ln y_i$ to the fitting function $\ln f(x; \mathbf{a})$. A direct fit of y_i to f would be dominated by the largest values of y_i since already small *relative* errors $|(y_i - f(x_i; \mathbf{a}))/y_i|$ will give large contributions to χ^2. Typical examples are power law fits with $f(x; a_1, a_2, a_3) = a_1 x^{a_2} + a_3$. Thus, if the absolute errors Δy_i are all of the same order of magnitude then the y_i should be directly fitted to the fitting function f, if the relative errors $\Delta y_i / y_i$ are of the same order of magnitude you may consider fitting $\ln y_i$ to $\ln f$. These considerations also elucidate that it is very important to have good estimates of the variances σ_i^2 that serve as weight factors in order to get a reliable fit to your data.

7.2.1 Estimating the Uncertainties of Least-Squares Fit Parameters

(see, J. R. Wolberg, Prediction Analysis, Van Nostrand Company, Princeton, 1967, ch. 3.10)

In this section we assume that the following preconditions apply:

(1) The measured values y_i are Gaussian distributed around the true values $\hat{y}_i$ with variances σ_i^2.

(2) The measurement errors $\Delta y_i = y_i - \hat{y}_i$ are uncorrelated.

(3) The data are fitted to a model $y = f(x; \mathbf{a})$. The true model obeys $\hat{y}_i - f(x_i; \hat{\mathbf{a}}) = 0$. It is assumed that $f(x_i; \mathbf{a})$ can be approximated around $\hat{\mathbf{a}}$ by the following expansion:

$$y_i - f(x_i; \mathbf{a}) = y_i - \hat{y}_i - \sum_{k=1}^{M} \frac{\partial f_i}{\partial a_k}(a_k - \hat{a}_k) + \mathcal{O}((a_k - \hat{a}_k)^2) \ , \tag{42}$$

where the notation $\partial f_i / \partial a_k$ has been used as a short hand for $\partial f(x_i; \mathbf{a}) / \partial a_k |_{\mathbf{a}=\hat{\mathbf{a}}}$.

(4) Only statistical errors are present.

At the minimum of χ^2 we have $\beta_k = 0$. Thus, setting the left-hand side of Eq. (42) to zero, multiplying the equation by $\frac{1}{\sigma_i^2} \frac{\partial f_i}{a_l}$ and summing over all data points we obtain

$$\sum_{i=1}^{N} \frac{1}{\sigma_i^2} \frac{\partial f_i}{\partial a_l} \Delta y_i = \sum_{i=1}^{N} \sum_{k=1}^{M} \frac{1}{\sigma_i^2} \frac{\partial f_i}{\partial a_l} \frac{\partial f_i}{\partial a_k} (a_k - \hat{a}_k) = \sum_{k=1}^{M} \alpha_{lk} (a_k - \hat{a}_k) \ .$$

By introducing the inverse $c = \alpha^{-1}$ of the matrix α so that $\sum_{m=1}^{M} \alpha_{km} c_{ml} = \delta_{ij}$, we can solve for the errors of the parameters

$$a_k - \widehat{a}_k = \sum_{l=1}^{M} c_{kl} \sum_{i=1}^{N} \frac{1}{\sigma_i^2} \frac{\partial f_i}{\partial a_l} \Delta y_i \ .$$

From this equation we obtain

$$(a_k - \widehat{a}_k)(a_l - \widehat{a}_l) = \sum_{i=1}^{N} \left(\frac{\Delta y_i}{\sigma_i^2} \right)^2 \left[\sum_{m=1}^{M} c_{km} \frac{\partial f_i}{\partial a_m} \right] \left[\sum_{n=1}^{M} c_{ln} \frac{\partial f_i}{\partial a_n} \right] + \begin{array}{c} \text{terms containing} \\ \Delta y_i \Delta y_j \text{ with } i \neq j \end{array} .$$

Since the measurement errors are assumed to be uncorrelated the terms $\sim \Delta y_i \Delta y_j$ with $i \neq j$ will be zero on average. Thus, if the measurement would be repeated many times the "mean value" or, more accurately, the most probable value

$$\langle (a_k - \widehat{a}_k)(a_l - \widehat{a}_l) \rangle = \sum_{i=1}^{N} \left\{ \left\langle \left(\frac{\Delta y_i}{\sigma_i} \right)^2 \right\rangle \sum_{n,m=1}^{M} c_{km} c_{kn} \frac{1}{\sigma_i^2} \frac{\partial f_i}{\partial a_k} \frac{\partial f_i}{\partial a_l} \right\} \tag{43}$$

is obtained. If σ_i^2 is the variance of the measurement of y_i then the ratio $\langle (\Delta y_i / \sigma_i)^2 \rangle \equiv 1$. However, in practice σ_i can be estimated at most, quite often the factors σ_i^2 are nothing but weight factors that have no direct relation to the variances. Thus, the expectation value $\langle (\Delta y_i / \sigma_i)^2 \rangle$ may not be a constant but depend on i. In order to proceed we approximate this expectation value by its average over i,

$$\left\langle \left(\frac{\Delta y_i}{\sigma_i} \right)^2 \right\rangle \approx \left\langle \frac{1}{N} \sum_{i=1}^{N} \left(\frac{\Delta y_i}{\sigma_i} \right)^2 \right\rangle =: Q \ .$$

Since we allow $\langle (\Delta y_i)^2 \rangle$ to differ from σ_i^2 the average Q must be estimated in a different way later on. Using the definition of the matrix α (35) we obtain the following result for the correlations of the a_k

$$\langle (a_k - \widehat{a}_k)(a_l - \widehat{a}_l) \rangle = Q c_{kl} \tag{44}$$

and, specifically, the estimated variances of the parameters

$$\left\langle (a_k - \widehat{a}_k)^2 \right\rangle = Q c_{kk} \ . \tag{45}$$

It remains to estimate Q. Using Eq. (42) and the definition $R_i = y_i - f(x_i; \mathbf{a})$ of the residuals we find

$$(\Delta y_i)^2 = R_i^2 + 2R_i \sum_{k=1}^{M} \frac{\partial f_i}{\partial a_k} (a_k - \widehat{a}_k) + \sum_{k,l=1}^{M} \frac{\partial f_i}{\partial a_k} \frac{\partial f_i}{\partial a_l} (a_k - \widehat{a}_k)(a_l - \widehat{a}_l) \ .$$

The term linear in $a_k - \widehat{a}_k$ will be zero on average. By summing over i we obtain

$$NQ = \sum_{i=1}^{N} \left\langle \frac{R_i^2}{\sigma_i^2} \right\rangle + \sum_{i=1}^{N} \left\langle \frac{1}{\sigma_i^2} \sum_{k,l=1}^{M} \frac{\partial f_i}{\partial a_k} \frac{\partial f_i}{\partial a_l} (a_k - \widehat{a}_k)(a_l - \widehat{a}_l) \right\rangle \ .$$

The first term on the right-hand side is nothing but the minimized χ^2, the second term can be evaluated using (35) and (44) with the result MQ. Hence, we finally obtain

$$Q \equiv \left\langle \frac{1}{N} \sum_{i=1}^{N} \left(\frac{\Delta y_i}{\sigma_i} \right)^2 \right\rangle = \frac{\chi^2}{N-M} \tag{46}$$

and the variances

$$\left\langle (a_k - \widehat{a}_k)^2 \right\rangle = \frac{\chi^2}{N-M} c_{kk} \tag{47}$$

as well as the covariances

$$\langle (a_k - \widehat{a}_k)(a_l - \widehat{a}_l) \rangle = \frac{\chi^2}{N-M} c_{kl} \ . \tag{48}$$

The results (47), (48) explain why the matrix c is called the covariance matrix. From Eq. (47) one obtaines the standard deviation of the parameters $\Delta a_k = \sqrt{c_{kk}\chi^2/(N-M)}$.

Some final remarks: As already mentioned in Sec. 7.2 χ^2 is distributed according to the χ^2-probability function $P(\chi^2|\nu)$ that has a mean value of $\nu = N - M$. Thus, *if σ_i^2 is the correct variance of the measurement of y_i* Eq. (46) gives 1 on both sides of the equation. Otherwise, one can use the condition $\chi^2 \stackrel{!}{=} N - M$ to estimate an overall common factor of the σ_i^2 as already mentioned in the previous section. The final result (47) depends on several assumptions and approximations and for a particular experiment it may not be clear whether these assumptions are valid. Therefore, an independent estimate of the errors along the lines indicated at the end of the previous section is essential in almost every case.

7.2.2 The χ^2-Distribution

In this section the result (40) for the χ^2-distribution function will be derived. It is obtained under the condition that the measured data are independently Gaussian distributed,

$$p(y_j) = \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left[-\frac{(y_j - f(x_j;\mathbf{a}))^2}{2\sigma_j^2} \right] \ , \quad j = 1, \ldots, N \ , \tag{χ^2-1}$$

see (29). Furthermore, χ^2 depends on the y_j through (30) and its distribution is constrained by the gradient conditions

$$\sum_{j=1}^{N} b_{nj} \frac{y_j - f(x_j;\mathbf{a})}{\sigma_j} = 0 \ , \qquad n = 1, \ldots, M \ , \tag{χ^2-2}$$

with $b_{nj} := \frac{1}{\sigma_i} \frac{\partial f(x_j;\mathbf{a})}{\partial a_n}$, $n = 1, \ldots, M$, $j = 1, \ldots, N$. The probability density of χ^2 is therefore given by

$$p(\chi^2|\nu) = c \prod_{j=1}^{N} \left[\int_{-\infty}^{\infty} dy_j p(y_j) \right] \delta\left(\chi^2 - \sum_{j=1}^{N} \left[\frac{y_j - f(x_j;\mathbf{a})}{\sigma_j} \right]^2 \right) \prod_{n=1}^{M} \delta\left(\sum_{j=1}^{N} b_{nj} \frac{y_j - f(x_j;\mathbf{a})}{\sigma_j} \right) \ , \tag{χ^2-3}$$

where the constraints have been taken into account through the products of δ-functions. The constant c is determined by the normalization condition $\int_0^\infty d\chi^2\, p(\chi^2|\nu) = 1$. Substituting $\eta_j = (y_j - f(x_j; \mathbf{a}))/\sigma_j$ and using $\int \mathcal{D}\eta = \prod_{j=1}^N \int_{-\infty}^\infty d\eta_j/\sqrt{2\pi}$ as a short hand, we find that

$$p(\chi^2|\nu) = c \int \mathcal{D}\eta\, e^{-\frac{1}{2}\sum_{j=1}^N \eta_j^2}\, \delta\left(\chi^2 - \sum_{j=1}^N \eta_j^2\right) \prod_{n=1}^M \delta\left(\sum_{j=1}^N b_{nj}\eta_j\right) .$$

Using the representation $\delta(x) = \int_{-\infty}^\infty \frac{dq}{2\pi}\, e^{-iqx}$ the δ-functions can be expressed through integrals as well. Therefore, the probability density function can be written as a product of Gaussian integrals,

$$\begin{aligned} p(\chi^2|\nu) &= c \int \mathcal{D}\eta\, e^{-\frac{1}{2}\sum_{j=1}^N \eta_j^2} \int_{-\infty}^\infty \frac{dq}{2\pi}\, e^{-iq\left(\chi^2 - \sum_{j=1}^N \eta_j^2\right)} \prod_{n=1}^M \int_{-\infty}^\infty \frac{d\zeta_n}{2\pi} e^{-i\zeta_n\left(\sum_{j=1}^N b_{nj}\eta_j\right)} \\ &= c \int \mathcal{D}\eta \int \mathcal{D}\zeta \int_{-\infty}^\infty \frac{dq}{2\pi}\, \exp\left\{-\frac{1}{2}\sum_{j=1}^N \left[(1-2iq)\eta_j^2 + 2i\eta_j \sum_{n=1}^M \left(\frac{1}{\sigma_j} b_{nj}\zeta_n\right)\right]\right\} e^{-iq\chi^2} , \end{aligned}$$

where $\int \mathcal{D}\zeta = \prod_{n=1}^M \int_{-\infty}^\infty \frac{d\zeta_n}{2\pi}$. Now all the integrals can be calculated: the integrals over the η variables are obtained using $\int_{-\infty}^\infty \frac{dx}{\sqrt{2\pi}}\, e^{-(\alpha x^2 - 2\beta x)/2} = e^{-\beta^2/(2\alpha)}/\sqrt{\alpha}$. Thus,

$$\begin{aligned} p(\chi^2|\nu) &= c \int \mathcal{D}\zeta \int_{-\infty}^\infty \frac{dq}{2\pi} \frac{1}{(1-2iq)^{N/2}} \exp\left\{-\frac{1}{2(1-2iq)} \sum_{j=1}^N \left[\sum_{n=1}^M \zeta_n b_{nj}\right]^2\right\} e^{-iq\chi^2} \\ &= c \int \mathcal{D}\zeta \int_{-\infty}^\infty \frac{dq}{2\pi} \frac{1}{(1-2iq)^{N/2}} \exp\left\{-\frac{1}{2(1-2iq)} \sum_{n,m=1}^M \zeta_n \alpha_{nm} \zeta_m\right\} e^{-iq\chi^2} , \end{aligned}$$

where $\alpha_{nm} = \sum_{j=1}^N b_{nj} b_{mj}$ are the elements of the curvature matrix. Since α is real and symmetric there exists an orthogonal transformation D such that the matrix $\mathcal{M} = D^T \alpha D$ is diagonal. Setting $\boldsymbol{\xi} = D\boldsymbol{\zeta}$ the argument of the exponential becomes $-\frac{1}{2}\sum_{n=1}^M \frac{\lambda_n}{1-2iq}\xi_n^2$. Here, λ_n are the eigenvalues of the curvature matrix. Since $\det D = 1$ there is no Jacobian involved in the transformation from $\boldsymbol{\zeta}$ to $\boldsymbol{\xi}$. Thus, the integration over $\boldsymbol{\zeta}$ reduces to a product of usual Gaussian integrals $\int_{-\infty}^\infty \frac{dx}{2\pi} \exp[-\frac{1}{2}\frac{\lambda}{(1-2iq)}x^2] = (1-2iq)^{1/2}/\sqrt{2\pi\lambda}$. Hence,

$$p(\chi^2|\nu) = c \int_{-\infty}^\infty \frac{dq}{2\pi} \frac{1}{(2\pi)^{M/2}\sqrt{\det\alpha}} \frac{1}{(1-2iq)^{\nu/2}}\, e^{-iq\chi^2}$$

with $\det\alpha = \prod_{n=1}^M \lambda_n$ being the determinant of the curvature matrix and $\nu = N - M$ is the number of degrees of freedom. It remains to calculate the Fourier transform $I_\mu(x) := \int_{-\infty}^\infty \frac{dq}{2\pi}(1-2iq)^{-\mu} e^{-iqx}$ of $\widehat{p}(q|\nu) = (1-2iq)^{-\nu/2}$ with $\mu = \nu/2$. Using partial integration it is shown that these integrals obey the recursion relation $I_\mu(x) = \frac{x}{2(\mu-1)} I_{\mu-1}(x)$ for $\mu > 1$. It therefore suffices to calculate $I_1(x)$ and $I_{1/2}(x)$. $I_1(x)$ can be calculated by closing the path of integration in the complex plane and using the law of residues: for $x > 0$ the path of integration is taken to be along the real axis and closed along a half circle in the lower half plane, where $\mathrm{Im}(q) < 0$ so that the contribution of the half circle goes to zero as its radius approaches ∞. Similarly, for $x < 0$ the path must be closed in the upper half plane where

$\mathrm{Im}(q) > 0$. Since the integrand has a pole at $q = -i/2$ we see immediately that $p(\chi^2|\nu) = 0$ for $\chi^2 < 0$ as it has to be. For $x > 0$ we find $I_1(x) = -i\,\mathrm{Res}_{q=-i/2}[e^{-iqx}/(1-2iq)] = \frac{1}{2}e^{-x/2}$, where the leading minus sign accounts for the fact that the path of integration is in the clockwise direction. Consequently, for integer μ the integral $I_\mu(x) = \frac{1}{2}\left(\frac{x}{2}\right)^{\mu-1}\frac{1}{(\mu-1)!}e^{-x/2}$. To calculate $I_{1/2}(x)$ we use the relation $1/\sqrt{z} = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} ds\, \exp[-\frac{z}{2}s^2]$ valid for $\mathrm{Re}\, z > 0$. Hence,

$$\begin{aligned}
I_{1/2}(x) &= \int_{-\infty}^{\infty}\frac{dq}{2\pi}\frac{e^{-iqx}}{(1-2iq)^{1/2}} = \frac{1}{(2\pi)^{3/2}}\int_{-\infty}^{\infty}dq\int_{-\infty}^{\infty}ds\, e^{-\frac{1}{2}(1-2iq)s^2}e^{-iqx} \\
&= \frac{1}{(2\pi)^{3/2}}\int_{-\infty}^{\infty}ds\, e^{-s^2/2}\int_{-\infty}^{\infty}dq\, e^{i(s^2-x)q} = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty}ds\, e^{-s^2/2}\delta(s^2-x)\ .
\end{aligned}$$

Again, for $x < 0$ we find that $I_{1/2}(x) = 0$. For $x > 0$ we use that $\delta(s^2-x) = [\delta(s-\sqrt{x}) + \delta(s+\sqrt{x})]/(2\sqrt{x})$ and arrive at $I_{1/2}(x) = e^{-x/2}/\sqrt{2\pi x}$. Thus,

$$I_\mu(x) = \frac{1}{2^\mu\Gamma(\mu)}x^{\mu-1}e^{-x/2}\ ,\quad x > 0\ ,$$

valid for all $\mu > 0$. Here, $\Gamma(\mu)$ is the Gamma function with $\Gamma(z+1) = z\Gamma(z)$, $\Gamma(1) = 1$, and $\Gamma(1/2) = \sqrt{\pi}$. Putting everything together we now have (for $\chi^2 > 0$)

$$p(\chi^2|\nu) = \tilde{c}\frac{1}{2\Gamma(\nu/2)}\left(\frac{\chi^2}{2}\right)^{\nu/2-1}e^{-\chi^2/2}$$

with $\tilde{c} = c/[(2\pi)^{M/2}\sqrt{\det\alpha}]$. Normalization requires $1 = \int_0^\infty dx\, p(x|\nu) = \tilde{c}\hat{p}(q=0|\nu) = \tilde{c}$. Thus, the final result for the χ^2-probability density function is

$$p(\chi^2|\nu) = \frac{1}{2^{\nu/2}\Gamma(\nu/2)}\left(\chi^2\right)^{\nu/2-1}e^{-\chi^2/2}\qquad \text{for}\quad \chi^2 > 0 \qquad (\chi^2\text{-4})$$

and $p(\chi^2|\nu) = 0$ for $\chi^2 < 0$.

8. Numerical Integration of Differential Equations

8.1 Equation of Motions in Physics

Probably the first example that comes to your mind are the classical equations of motion, i.e., Newton's equations (here for N particles in three dimensions)

$$m_i\ddot{x}_i = F_i(x_1, x_2, \ldots, x_{3N}) \ , \quad i = 1, \ldots, 3N \ , \tag{49}$$

or, using the Lagrange formalism for a system with f degrees of freedom and the generalized coordinates q_i, $i = 1, \ldots, f$,

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}_i} - \frac{\partial \mathcal{L}}{\partial q_i} = 0 \ . \tag{50}$$

These are sets of second order ordinary differential equations: the variables x_i, resp. q_i depend only on time and the highest derivative that appears in these equation is of second order. Hamilton's equations (p_i is the conjugate momentum to q_i)

$$\dot{q}_i = \frac{\partial \mathcal{H}}{\partial p_i} \quad \text{and} \quad \dot{p}_i = -\frac{\partial \mathcal{H}}{\partial q_i} \ , \quad i = 1, \ldots, f$$

instead form a set of $2f$ ordinary first order differential equations. Generally, a set of f second order differential equations for the variables x_i can be written as a set of $2f$ first order differential equations by introducing the derivatives $\dot{x}_i$ as new variables.

Problems in quantum mechanics are described by Schrödinger's equation

$$i\hbar\frac{\partial \psi(\mathbf{r}, t)}{\partial t} = \left[-\frac{\hbar^2}{2m}\nabla^2 + V(\mathbf{r}, t)\right]\psi(\mathbf{r}, t) \ . \tag{51}$$

Here, the wavefunction ψ depends not only on time but also on space $\mathbf{r}$. Schrödinger's equation is a partial differential equation, which are usually much harder to solve than ordinary differential equations. However, Schrödinger's equation is a *linear* equation, which are generally much easier to solve than nonlinear equations.

Another example for a differential equation is the "coffee cooling" problem

$$\frac{dT}{dt} = -\alpha(T - T_r) \ , \tag{52}$$

where T is the temperature of the coffee inside your cup and T_r is the room temperature; α is a constant that describes how fast heat is transferred from the cup to the surrounding air. The same equation describes the decay of atomic nuclei and similar decay phenomena.

An obvious generalization of Eq. (52) leads to the diffusion equation

$$\frac{\partial \rho(\mathbf{r}, t)}{\partial t} = D\nabla^2\rho(\mathbf{r}, t) \tag{53}$$

that describes the spreading of some density ρ in space and time. This is again a partial differential equation that can be quite hard to solve, especially if nonlinearities are added to the right-hand side of the equation (53).

A further generalization leads to stochastic differential equations or Langevin equations that have a stochastic term on the right-hand side that models random forces due to the surrounding environment, e.g., gas atoms, thermal excitations, etc. Such Langevin equations,

$$\frac{\partial \rho(\mathbf{r},t)}{\partial t} = D\nabla^2 \rho(\mathbf{r},t) + f[\rho(\mathbf{r},t)] + \eta(\mathbf{r},t) \ , \tag{54}$$

where $f[\rho]$ is some nonlinear function and η is the random noise, are one of the most difficult differential equations to be solved numerically.

In this course we will mainly discuss ordinary differential equations with some remarks about partial differential equations. Langevin equations will not be discussed because of the difficulties involved in their simulations, although these kind of equations are very important in describing dynamical processes in, e.g., condensed matter physics.

8.2 Euler Method

The simplest method to integrate differential equations is the Euler method. Although it is rarely used in practice (with the exception for Langevin equations, where it is in some cases the only method that works) it is conceptually important because of its simplicity. Consider the differential equation

$$\frac{dx}{dt} = f(x,t) \ . \tag{55}$$

In order to evaluate the derivative time has to be discretized:

$$t_n = t_0 + n\Delta t \ . \tag{56}$$

Assume that we have an approximation x_n for $x(t_n)$. The Euler method simply extrapolates linearly from one time step to the next, $(x_{n+1} - x_n)/\Delta t + \mathcal{O}(\Delta t) = f(x_n, t_n)$, or

$$x_{n+1} = x_n + \Delta t f(x_n, t_n) + \mathcal{O}[(\Delta t)^2] \approx x(t_{n+1}) \ . \tag{57}$$

The term $\mathcal{O}[(\Delta t)^2]$ specifies the error that is may in every time step. The total error after n steps at time t_n is

$$x(t_n) - x_n = n\mathcal{O}[(\Delta t)^2] = \mathcal{O}(\Delta t) \ , \tag{58}$$

since $n = (t_n - t_0)/\Delta t$. An algorithm is called to be of ν-th order if the total error is of order $\mathcal{O}[(\Delta t)^\nu]$. Thus, the Euler algorithm is of first order.

Example: The coffee cooling problem (52): The Euler scheme gives

$$T_n = T_{n-1}(1 - \alpha\Delta t) = T_0(1 - \alpha\Delta t)^n \ .$$

Here, the constant T_r has been eliminated by choosing the temperature difference $T - T_r$ as the new variable T. The exact solution to the coffee cooling problem is

$$T_n = T_0 e^{-\alpha n \Delta t} = T_0 \left[1 - \alpha\Delta t + \tfrac{1}{2}(\alpha\Delta t)^2 - \ldots\right]^n \ .$$

Hence the Euler method is a reasonable approximation only if $\Delta t \ll 1/\alpha$. For $1/\alpha < \Delta t < 2/\alpha$ the Euler method yields an oscillatory decay, whereas for $\Delta t > 2/\alpha$ the oscillations even increase exponentially. Generally, if there are several times scales in the problem, always the shortest time scale limits the step size Δt. It is interesting to look at the results of a modified Euler method that evaluates the right-hand side of (55) not at time t_n but at time t_{n+1} resulting in the scheme

$$x_{n+1} = x_n + \Delta t f(x_{n+1}, t_{n+1}) + \mathcal{O}[(\Delta t)^2] \ . \tag{59}$$

This is now an implicit equation for x_{n+1} that requires the solution of a nonlinear equation, if f is nonlinear. The scheme (59) is called fully implicit because the function f is only evaluated at time t_{n+1}. Since the time derivative $dx/dt \approx (x_{n+1} - x_n)/\Delta t$ is centered around $t_n + \Delta t/2$ it is natural to center the right-hand side of the equation in the same way in order to obtain an algorithm of second order, i.e.,

$$x_{n+1} = x_n + \frac{\Delta t}{2} \left[f(x_{n+1}, t_{n+1}) + f(x_n, t_n) \right] + \mathcal{O}[(\Delta t)^3] \ . \tag{60}$$

This again is an implicit scheme. For the coffee cooling problem the function f is linear, therefore we can again solve for T_n,

$$T_n = T_{n-1} \left(\frac{1 - \frac{1}{2}\alpha\Delta t}{1 + \frac{1}{2}\alpha\Delta t} \right) = \left[\frac{1 - \frac{1}{2}\alpha\Delta t}{1 + \frac{1}{2}\alpha\Delta t} \right]^n T_0 \ .$$

Now we have exponential convergence for *all* time steps Δt, therefore this scheme always converges to the correct asymptotic state, although an accurate description of the time evolution is again obtained only if $\Delta t \ll 1/\alpha$. The property of absolute stability is makes implicit methods superior to explicit ones. This is especially important for partial differential equations, where this property even outweighs the disadvantage of having to solve a set of nonlinear equations at each time step.

Example:

$$\frac{\partial y(x,t)}{\partial t} = D \frac{\partial^2 y(x,t)}{\partial x^2} + f(y; x, t) \tag{61}$$

with some function $f(y; x, t)$ that has a nonlinear dependence on y. The discretization scheme that corresponds to (60) now reads

$$\begin{aligned} y_i(t + \Delta t) = y_i(t) + \frac{\Delta t}{2} \Bigg\{ & \frac{D}{(\Delta x)^2} \Big[y_{i+1}(t) - 2y_i(t) + y_{i-1}(t) \\ & \qquad + y_{i+1}(t + \Delta t) - 2y_i(t + \Delta t) + y_{i-1}(t + \Delta t) \Big] \\ & + f(y_i(t); x_i, t) + f(y_i(t + \Delta t); x_i, t + \Delta t) \Bigg\} , \quad i = 1, \ldots, L \ , \end{aligned} \tag{62}$$

where the spatial variable x has been discretized as well, $x \to x_i$, $i = 1, \ldots, L$, and the abbreviation $y_i(t) \equiv y(x_i, t)$ has been used. Eq. (62) defines a set of nonlinear equations that has to be solved for $y_i(t + \Delta t)$, $i = 1, \ldots, L$ in each time step using, e.g., the method discussed in section 6.4. The scheme (62) is called Crank-Nicholson method and is one of the most important methods for partial differential equations. Although the solution of the nonlinear equations requires much more cpu-time than a simple Euler step, the stability of the method allows the use of a much larger time step Δt so that the Crank-Nicholson method is usually by far superior to the Euler method, in particular, if one is interested in the late-time asymptotics of differential equation. For ordinary differential equations usually other higher order methods are used that avoid the solution of nonlinear equations, see the next sections.

8.3 Leapfrog Algorithm

The Euler method uses the first order approximation $dx/dt \approx (x_{n+1} - x_n)/\Delta t + \mathcal{O}(\Delta t)$. A better, second order algorithm is obtained, if we use the approximation $dx/dt = (x_{n+1} - x_{n-1})/(2\Delta t) + \mathcal{O}[(\Delta t)^2]$ that leads to the algorithm $x_{n+1} = x_{n-1} + 2\Delta t f(x_n, t_n) + \mathcal{O}[(\Delta t)^3]$. This two-point formula (x_{n+1} depends on x_n and x_{n-1}) forms the basic idea for the leapfrog algorithm. This algorithm is mainly used to integrate second order differential equations like Newton's equations

$$\ddot{\mathbf{r}} = \frac{1}{m}\mathbf{F}(\mathbf{r}, \mathbf{v}, t) \; .$$

It calculates the positions $\mathbf{r}$ for $t_0, t_2, \ldots, t_{2n}$ and the velocities $\mathbf{v} = \dot{\mathbf{r}}$ at $t_1, t_3, \ldots, t_{2n-1}$. An initial Euler step is needed to calculate the velocities at t_1: $\mathbf{v}_1 = \mathbf{v}_0 + \frac{\Delta t}{m}\mathbf{F}(\mathbf{r}_0, \mathbf{v}_0, t_0)$. Then the leapfrog algorithm is implemented in the following way

$$\mathbf{r}_n = \mathbf{r}_{n-2} + 2\Delta t \mathbf{v}_{n-1} + \mathcal{O}[(\Delta t)^3] \tag{63a}$$

$$\mathbf{v}_{n+1} = \mathbf{v}_{n-1} + \frac{2\Delta t}{m}\mathbf{F}(\mathbf{r}_n, \mathbf{v}_n, t_n) + \mathcal{O}[(\Delta t)^3] \; . \tag{63b}$$

This algorithm can be readily implemented, if the forces do not depend on the velocities: $\mathbf{F} = \mathbf{F}(\mathbf{r}_n, t_n)$. If they do, however, an additional equation is needed to calculated the velocity $\mathbf{v}_n$ in Eq. (63b). There are several possibilities:

(1) Linear interpolation: $\mathbf{v}_n = \frac{1}{2}(\mathbf{v}_{n+1} + \mathbf{v}_n) + \mathcal{O}((\Delta t)^2)$. This leads to an implicit scheme

$$\mathbf{v}_{n+1} = \mathbf{v}_{n-1} + \frac{2\Delta t}{m}\mathbf{F}(\mathbf{r}_n, \frac{\mathbf{v}_{n+1} + \mathbf{v}_{n-1}}{2}, t_n) + \mathcal{O}((\Delta t)^3)$$

that is a good choice, if the function $\mathbf{F}$ depends linearly on $\mathbf{v}$ so that the equation can be solved for $\mathbf{v}_{n+1}$ analytically.
Example: linear friction: $\frac{1}{m}\mathbf{F} = \mathbf{g} - \alpha\mathbf{v}$ ($m\mathbf{g}$ gravitational force)

$$\begin{aligned} \Rightarrow \mathbf{v}_{n+1} &= \mathbf{v}_{n-1}\frac{1 - \alpha\Delta t}{1 + \alpha\Delta t} + \mathbf{g}\frac{2\Delta t}{1 + \alpha\Delta t} + \mathcal{O}((\Delta t)^3) \\ &= \mathbf{v}_{n-1} - 2\Delta t(1 - \alpha\Delta t)(\mathbf{g} - \alpha\mathbf{v}) + \mathcal{O}((\Delta t)^3) \; . \end{aligned}$$

(2) Use Euler: $\mathbf{v}_n = \mathbf{v}_{n-1} + \frac{\Delta t}{m}\mathbf{F}(\mathbf{r}_n, \mathbf{v}_{n-1}, t_{n-1}) + \mathcal{O}((\Delta t)^2)$; strictly speaking $\mathbf{r}_n$ instead of $\mathbf{r}_{n-1}$ must be used in the argument of $\mathbf{F}$, but the error is of order $\mathcal{O}((\Delta t)^2)$ only.

$$\Rightarrow \mathbf{v}_{n+1} = \mathbf{v}_{n-1} + \frac{\Delta t}{m}\mathbf{F}(\mathbf{r}_n, \mathbf{v}_{n-1} + \tfrac{\Delta t}{m}\mathbf{F}(\mathbf{r}_n, \mathbf{v}_{n-1}, t_{n-1}), t_n) + \mathcal{O}((\Delta t)^3) \; .$$

This method requires two evaluations of the forces for each time step of $2\Delta t$.
(3) Use a Taylor expansion for $\mathbf{F}$:

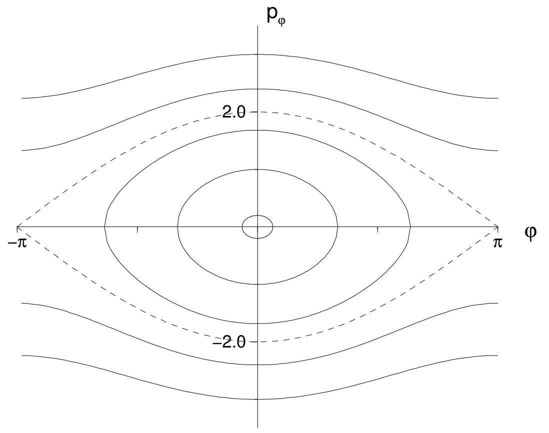
$$\Rightarrow F_i(\mathbf{r}_n, \mathbf{v}_n, t_n) = F_i(\mathbf{r}_n, \mathbf{v}_{n-1}, t_n) + \frac{\Delta t}{m} \sum_j \frac{\partial F_i}{\partial v_j}(\mathbf{r}_n, \mathbf{v}_{n-1}, t_n) F_j(\mathbf{r}_n, \mathbf{v}_{n-1}, t_n) + \mathcal{O}((\Delta t)^2) \ .$$

The accuracy of the leapfrog algorithm is still not very good, thus, if accuracy is your main concern, a higher order algorithm such as the fourth-order Runge-Kutta algorithm (see Sec. 8.5) should be used. However, the leapfrog algorithm has a property that makes it sometimes even superior to these higher order algorithms: At least in its original form (63) the algorithm is *time reversal invariant*. Because of this symmetry the algorithm conserves the total energy of the system. There are still fluctuations of order Δt, but there is no systematic drift in the total energy [for details, see: H. Frauenkron, P. Grassberger, Int. J. Mod. Phys. C **5**, 37 (1994)]. Thus, if energy conservation is more important than accuracy, as it is, e.g., in the calculation of Poincaré maps of Hamiltonian systems (see next section), the leapfrog algorithm may very well be the method of choice.

8.4 Poincaré Maps and Chaotic Systems

The deterministic description of physical systems through nonlinear (!) differential equations may result in irregular or "chaotic" behavior: deterministic chaos (Poincaré 1892). This phenomenon results from an exponential separation of trajectories in phase space as a function of time and *not* from external noise, roundoff errors, infinite number of degrees of freedom (continuum mechanics), quantum mechanical uncertainties, etc. However, the accuracy of the initial conditions is of crucial importance, a typical problem for, e.g., weather forecasts [phase space for a system with f degrees of freedom: generalized coordinates q_i, $i = 1, \ldots, f$; conjugate momenta $p_i = \partial\mathcal{L}/\partial\dot{q}_i$; a point in the $2f$ dimensional phase space is specified through all coordinates q_i and conjugate momenta p_i: $(q_1, \ldots, q_f, p_1, \ldots, p_f)$].

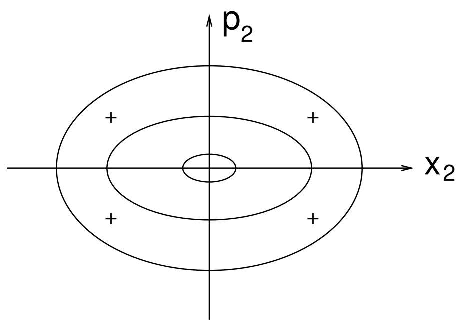
The dynamics of low-dimensional chaotic systems are best studied by plotting their trajectories in phase space. In this course only Hamiltonian systems with conserved total energy are discussed.
Example for trajectories in phase space: pendulum with total energy $\mathcal{H} = \frac{1}{2ml^2}p_\varphi^2 + mgl(1 - \cos\varphi)$, where $p_\varphi = ml^2\dot{\varphi}$ is simply the angular momentum.



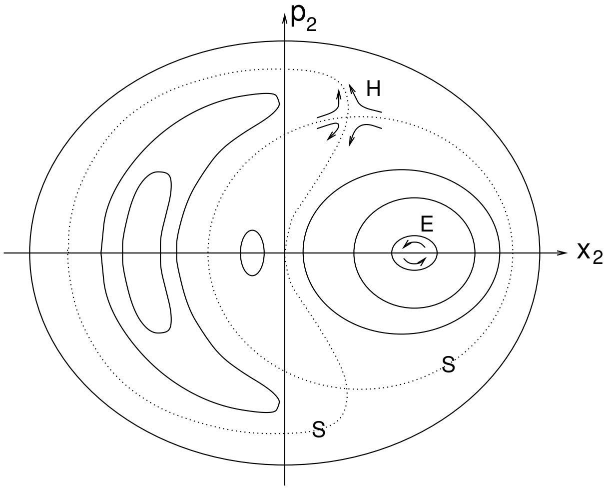
Plot of the phase space trajectories of a pendulum, $p_\varphi = ml^2\dot{\varphi}$ is the conjugate momentum of the coordinate φ. The different lines correspond to different energies $E = \frac{1}{2ml^2}p_\varphi^2 + mgl(1 - \cos\varphi)$. The dashed line is the separatrix.

An integrable or non-chaotic system has as many constants of motion as there are degrees of freedom. Hence there is no chaos in Hamiltonian systems with one degree of freedom since there is always energy conservation. This is no longer the case for two degrees of freedom. For such a system the phase space is four dimensional, therefore it is hard to plot the trajectories directly. Instead one studies the so-called Poincaré map: This map is obtained by a cut through the $2f$ dimensional phase space along a $2f-1$ dimensional hyperplane, e.g., the trajectory is plotted every time it crosses through the plane $x_1 = 0$. For Hamiltonian systems energy conservation can be used to eliminated one more variable, e.g., p_1. For a two-dimensional system ($2f = 4$) the Poincaré map is therefore a $2f - 2 = 2$ dimensional plot $p_2(x_2)$: Choose some energy $E = const.$, integrate the differential equations, and add a point to the plot $p_2(x_2)$ every time the trajectory (x_1, p_1, x_2, p_2) crosses the plain $x_1 = 0$. This Poincaré plot still contains the full information of the current state of the system: p_1 is determined via $E(x_1 = 0, x_2, p_1, p_2) = const.$

As an example let us look at the two dimensional harmonic oscillator with the momenta $p_1 = m_1\dot{x}_1$, $p_2 = m_2\dot{x}_2$, potential $V(x_1, x_2) = \frac{1}{2}\omega_1^2 x_1^2 + \frac{1}{2}\omega_2^2 x_2^2$, and total energy $E = \frac{1}{2m_1}p_1^2 + \frac{1}{2m_2}p_2^2 + V(x_1, x_2)$. This is still an integrable system since both energies $E_i = \frac{1}{2m_i}p_i^2 + \frac{1}{2}\omega_i^2 x_i^2$, $i = 1, 2$ are conserved separately. The solutions of the equations of motion are easily written down: $x_1(t) = x_1^{(0)} \sin(\omega_1 t + \varphi_1)$, $x_2(t) = x_2^{(0)} \sin(\omega_2 t + \varphi_2)$, $p_2(t) = m_2\omega_2 x_2^{(0)} \cos(\omega_2 t + \varphi_2)$. The constants $x_i^{(0)}$, φ_i, $i = 1, 2$, are determined by the initial conditions. The Poincaré plot is obtained by plotting p_2, x_2 for times $t_n = (n\pi - \varphi_1)/\omega_1$, see figure to the right. If ω_2/ω_1 is an irrational number, the corresponding trajectory appears as an ellipse in the Poincaré plot, the size of the ellipse depends on the initial conditions or, equivalently, in which way the total energy is distributed between E_1 and E_2. However, these ellipses have nothing to do with the ellipses that appeared in the plot of the phase-space trajectories of the pendulum shown above: In the latter case the curves show a true evolution of the system: As time proceeds the system moves along one of those ellipses. The Poincaré plot gives a stroboscopic picture instead: *consecutive points are generally not close to each other*. In fact, if the ratio ω_2/ω_1 is rational, say $1/N$, the Poincaré map shows only N disjunct points, since after N points the system will be at its initial point again. Such trajectories are called fixed points of period N of the Poincaré map. A fixed point of period 4 is indicated by crosses in the Poincaré map of the two-dimensional harmonic oscillator. Only for irrational frequency ratios the trajectories will eventually run through



A more general Poincaré map of an integrable system is shown in the figure below. It shows elliptic (or stable) fixed points labeled “E”, hyperbolic (or unstable) fixed points labeled “H”. In the neighborhood of elliptic fixed points we find the same kind of elliptical trajectories that appeared in the Poincaré map of the two-dimensional harmonic oscillator, thus the name “elliptic fixed point”. Trajectories that come close to hyperbolic fixed points do not remain in the neighborhood of the fixed point, instead the system moves away from the fixed point and the trajectory reaches points in phase space that are far way from the



Poincaré map of an integrable system. E indicates an elliptic fixed point, H a hyperbolic fixed point. The dotted lines are separatrices labeled S. The behavior of the map close to two of the fixed points is indicated by arrows

hyperbolic fixed point. The behavior of the system close to the fixed points is indicated by the arrows that also explain why the fixed points are called "elliptic" and "hyperbolic". Hyperbolic fixed points are connected by separatrices, these are trajectories that separated the regimes that belong to different elliptic fixed points.

What happens if a small perturbations is added to an integrable system? There are two possibilities: (1) the system remains integrable; this is the non-generic case, (2) it becomes chaotic. The way the system becomes chaotic, when the strength of the perturbation is increased, is described by the famous KAM theorem (Kolmogorov, Arnold, Moser):
Trajectories with irrational frequency ratios become distorted, but they still exist up to a certain strength of the perturbation. The statement of the theorem will be given her for the special case of $f = 2$ degrees of freedom. It is valid for higher dimensions as well, but the general statement is more complicated (see, e.g., M. Rasetti, *Modern Methods in Equilibrium Statistical Mechanics*, World Scientific, Singapore 1986). Generally, any trajectory of an integrable system with 2 degrees of freedom can be characterized by two frequencies ω_1, ω_2 that describe the periodicity of the two coordinates in complete analogy to the example of the two-dimensional harmonic oscillator mentioned above. In the more general case these frequencies are obtained by a transformation to action and angle variables that will not be discussed here, but that can be found in many books on classical mechanics. The KAM theorem specifies how much the frequency ration ω_1/ω_2 must differ from *all* rational numbers r/s (r, s integers) so that the corresponding trajectory still exists for a certain strength of the perturbation:

$$\left|\frac{\omega_1}{\omega_2} - \frac{r}{s}\right| \geq \frac{c}{s^{2+\delta}} \ , \tag{64}$$

where c and δ are arbitrary constants. The trajectories, for which (64) is true, are called KAM trajectories. It is interesting to see, whether there are at all irrational numbers that do obey the inequality (64). To show that, we add up all intervals of width $c/s^{2+\delta}$ around all rational numbers r/s within the interval $(0, 1)$. For each denominator s there are at most s of these rational numbers. Therefore the total length of all these intervals together is at most

$$c\sum_{s=2}^{\infty}\frac{s}{s^{2+\delta}} < c\int_1^{\infty} s^{-1-\delta} = \frac{c}{\delta}\ .$$

Thus, for small c and large δ this length is smaller than 1, hence there is a remaining Cantor set of measure larger than zero that is covered by irrational numbers for which the statement (64) is true. Small c and large δ require that the strength ϵ of the perturbation is small (clearly, for $\epsilon = 0$ we must have $c = 0$). Thus, for small perturbations there exist many KAM trajectories and the fraction of phase space, where regular motion is observed, is large. These KAM trajectories can be found in the neighborhood of the elliptic fixed points of the unperturbed system, whereas KAM trajectories in the neighborhood of hyperbolic fixed points disappear, in particular a separatrix does no longer exist. Instead the regime in the neighborhood of the separatrix becomes chaotic, i.e., a single trajectory now covers a whole *area* of the Poincaré plot, no longer only a single line as it was the case for integrable systems. As the strength of the perturbation increases, more and more KAM trajectories disappear and the chaotic regime in phase space becomes larger until the last KAM trajectory is destroyed and *one single* chaotic trajectory fills the whole phase space: this is state called fully developed chaos (see, e.g., the Henon-Heiles system for the energy $E = 1/6$).

8.5 Runge-Kutta Method

In this chapter a higher order algorithm for the integration of differential equations will be discussed, namely the fourth-order Runge-Kutta method. As an introduction to the Runge-Kutta methods, however, first the second order Runge-Kutta method will be introduced that is also called midpoint method. The starting point is again the central difference scheme

$$x_{n+1} = x_{n-1} + \widetilde{\Delta t} f(x_n, t_n) + \mathcal{O}((\widetilde{\Delta t})^3) \tag{65}$$

for the differential equation (55). The midpoint method is obtained by approximating x_n using the Euler method,

$$x_n = x_{n-1} + \widetilde{\Delta t} f(x_{n-1}, t_{n-1}) + \mathcal{O}((\widetilde{\Delta t})^2)\ .$$

The final form of the second order Runge-Kutta method is obtained by introducing the new step size $\Delta t = 2\widetilde{\Delta t}$, i.e.,

$$\begin{aligned} k_1 &= \Delta t f(x_n, t_n) && \text{(66a)}\\ k_2 &= \Delta t f(x_n + \tfrac{1}{2}k_1, t_n + \tfrac{1}{2}\Delta t) && \text{(66b)}\\ x_{n+1} &= x_n + k_2 + \mathcal{O}((\Delta t)^3)\ . && \text{(66c)} \end{aligned}$$

Here, the x_n have been redefined as well so that x_n again corresponds to $t_n = t_0 + n\Delta t$. Whereas the Euler method uses the derivative x' at time t_n to extrapolate to x_{n+1}, the algorithm (66) first calculates x at the midpoint of the interval in order to use the derivative at this new point to extrapolate from x_n to x_{n+1}. Consequently this algorithm now needs two function evaluations per time step. A third order Runge-Kutta algorithm (rarely used) is obtained using (66a) and (66b) together with

$$\begin{aligned} \widetilde{k}_3 &= \Delta t f(x_n - k_1 + 2k_2, t_n + \Delta t) \ , && (67a) \\ x_{n+1} &= x_n + \tfrac{1}{6}(k_1 + 4k_2 + \widetilde{k}_3) + \mathcal{O}((\Delta t)^4) \ . && (67b) \end{aligned}$$

This formula can be derived with the help of Simpson's rule (13):

$$\begin{aligned} x_{n+1} &= x_n + \int_{t_n}^{t_{n+1}} dt\, f(x(t), t) \\ &= x_n + \tfrac{\Delta t}{6} \Big[f(x_n, t_n) + 4 f(x_{n+\frac{1}{2}}, t_n + \tfrac{1}{2}\Delta t) + f(x_{n+1}, t_{n+1}) \Big] + \mathcal{O}((\Delta t)^4) \ . \end{aligned}$$

The formula for $\widetilde{k}_3$ can be inferred from the expansion

$$x_{n+1} = x_n - \tfrac{1}{2}\Delta t x'_n + \Delta t x'_{n+\frac{1}{2}} + \mathcal{O}((\Delta t)^3) \ .$$

The fourth-order Runge-Kutta method again uses Eqs. (66a) and (66b), but a different formula for k_3:

$$\begin{aligned} k_3 &= \Delta t f(x_n + \tfrac{1}{2}k_2, t_n + \tfrac{1}{2}\Delta t) \ , && (68a) \\ k_4 &= \Delta t f(x_n + k_3, t_n + \Delta t) \ , && (68b) \\ x_{n+1} &= x_n + \tfrac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) + \mathcal{O}((\Delta t)^5) \ . && (68c) \end{aligned}$$

Eq. (68) requires 4 function evaluations per time step, once at the initial point, twice at trial midpoints, and once at a trial endpoint. Therefore, this method is only superior to the second order method, if it allows for a time step that is at least twice as large as the one that can be used for (66) for the same accuracy. For most problems this turns out to be correct, however, this is not necessarily true in all cases. The fourth order Runge-Kutta method is the last algorithm in this sequence that requires p function evaluations with p being also the order of the algorithm. Higher order Runge-Kutta methods require even more function evaluations and are therefore rarely used.

Adaptive Step-Size Control for the Fourth-Order Runge-Kutta method

It is exactly the implementation of the fourth-order Runge-Kutta algorithm in connection with an adaptive step-size control that makes this method superior to other methods in many cases. The reason is easily understood by looking at the problem of launching a rocket from the earth towards the moon: On the long way from the earth to the moon the rocket flies on an almost straight trajectory, therefore a relatively large time step is acceptable. However, on its way around the moon the curvature of the rocket's path becomes large and a small step size is necessary. It is inconvenient and often impossible to vary the time step from the outside, rather the algorithm itself should determine a time step so that the overall error is kept below a certain bound. To implement an adaptive step-size control in, e.g., the leapfrog algorithm is almost impossible because positions and velocities are never calculated at the same time, hence changing the time step requires a change in the algorithm. There are many different ways to implement an adaptive step-size control in Runge-Kutta-type algorithms.

For example you can use the difference in the results for say x_{n+2} that are obtained by starting at x_n and use (i) two steps of size Δt or (ii) one step of size $2\Delta t$ to keep the error of x_{n+2} smaller than some accuracy ϵ. Here, we will compare the results of the second-order Runge-Kutta algorithm with those of the fourth-order method to estimate the accuracy of the algorithm. A different, more accurate, but also more complicated method is presented in the Numerical Recipes.

In order to compare the two methods it is helpful to rewrite Eq. (68c) slightly to make the similarity to the midpoint method more apparent:

$$\begin{aligned}
2^{\text{nd}}\text{ order:}\quad & x_{n+1}^{(2)} = x_n + k_2 + \mathcal{O}((\Delta t)^3) \\
4^{\text{th}}\text{ order:}\quad & x_{n+1}^{(4)} = x_n + k_2 + \delta + \mathcal{O}((\Delta t)^5) \\
\text{with}\quad & \delta = \tfrac{1}{6}(k_1 - 4k_2 + 2k_3 + k_4)\ .
\end{aligned}$$

The difference $x_{n+1}^{(4)} - x_{n+1}^{(2)} \simeq \delta$ between the two methods can be used to control the errors made in each time step. Let $\kappa = |\delta/k_2| \sim (\Delta t)^2$; for a system of M first-order differential equations set $\kappa = max|\delta^{(m)}/k_2^{(m)}|$ for $m = 1, \ldots, M$ with $k_2^{(m)} \neq 0$. We then require that $\kappa \leq \epsilon$ in order to keep the relative error made in each step* below ϵ. Since κ is proportional to Δt the relative error of the integration routine can be kept below ϵ by implementing the following idea: If the error in a step of size $\widetilde{\Delta t}$ is κ, the error made using $\Delta t = \widetilde{\Delta t}\sqrt{\epsilon/\kappa}$ is approximately ϵ. Hence we arrive at the following algorithm:

(i) Choose an overall accuracy ϵ and initial step size Δt.

(ii) Calculate δ, κ and

$$\Delta t_{\text{new}} = 0.8\Delta t_{\text{old}} \left(\frac{\epsilon}{\kappa}\right)^{1/2}\ . \tag{69}$$

The factor 0.8 is for safety reasons.

(iii) If $\kappa > \epsilon$, discard the step and choose the new step size $\Delta t = \max(\Delta t_{\min}, \Delta t_{\text{new}})$. If Δt_{new} becomes smaller than the prefixed $\Delta t_{\min}$ the program should probably warn about the loss in accuracy.

(iv) If $\kappa < \epsilon$ the step is accepted and the time step $\Delta t = \min(\Delta t_{\max}, \Delta t_{\text{new}})$ is chosen for the *next* step of the iteration.

(v) Iterate (ii), (iii), and (iv).

In principle this algorithm uses the fourth order Runge-Kutta method to estimate the error of the second-order method. This may not be very accurate. In the Numerical Recipes a fifth-order algorithm is used to estimate the error of the fourth-order Runge-Kutta method.

*It is also possible to keep the absolute error below some bound $\epsilon x_{\max}$. In that case set $\kappa = x_{\max}/\Delta t$. Both expressions for κ attempt to keep the global error below ϵ; if only the error per time step is to be kept below ϵ, κ can be multiplied by Δt and the time step is scaled with $(\epsilon/\kappa)^{1/3}$ instead of $\sqrt{\epsilon/\kappa}$.

9. Molecular Dynamics

(Reference: M. P. Allen, D. J. Tildesley: *Computer Simulations of Liquids*, Oxford University Press, Oxford, 1990)

Molecular dynamics typically deals with dynamical problems of *classical* many particle systems, such as gases, fluids, membranes, sand, etc. A molecular dynamics simulation consists of the numerical integration of the equations of motion of all particles simultaneously. To get realistic results the number of particles has to be as large as possible. Naturally one is not interested in the individual trajectories of the particles, but in more global expectation values that could also be measured in a corresponding laboratory experiment. Examples of these quantities of interest are the kinetic energy that is related to the temperature of the system, the total energy, and various correlation functions, such as the pair distribution function function (ρ is the average particle density)

$$g(r) = \frac{1}{\rho^2}\left\langle \sum_{i\neq j} \delta(\mathbf{r}_i)\delta(\mathbf{r}_j - \mathbf{r})\right\rangle = \frac{V}{N^2}\left\langle \sum_{i\neq j} \delta(\mathbf{r} - \mathbf{r}_{ij})\right\rangle \tag{70}$$

or its fourier transform, the structure factor that can be measured in scattering experiments, the velocity autocorrelation function

$$c_{vv}(t) = \langle \mathbf{v}(0)\mathbf{v}(t)\rangle = \frac{1}{N}\sum_{i=1}^{N} \langle \mathbf{v}_i(0)\mathbf{v}_i(t)\rangle \tag{71}$$

that contains information about, e.g., the diffusion coefficient, and many more. These quantities of interest are obtained by summing over all particles and by time averaging as well, e.g., the average $\langle \ldots \rangle$ in (71) contains an additional time average

$$\langle \mathbf{v}_i(0)\mathbf{v}_i(t)\rangle \equiv \frac{\Delta t}{T} \sum_{t'=t_0+\Delta t}^{t_0+T} \langle \mathbf{v}_i(t')\mathbf{v}_i(t+t')\rangle \quad .$$

<u>Examples</u>:

a) 1. MD simulation: B. J. Alder, T. E. Wainright (1957): Simulation of a gas of hard spheres ($\rightarrow$ contact interaction). Result: the velocity autocorrelation function decays algebraically instead of exponentially as it was expected: $c_{vv}(t) \sim t^{-d/2}$.

b) Membranes: Does a crumpling transition exist for two-dimensional objects embedded in three-dimensional space? Answer: If the particles of the membrane form a fluid (fluid membranes), the membrane is always crumpled, whereas for tethered membranes there is no crumpled phase, the membrane is always flat [F. F. Abraham, W. E. Rudge, M. Plischke, Phys. Rev. Lett. **62**, 1757 (1989)], i.e., the fluctuations in the direction of the overall surface normal scale as L^n with $n < 1$. In MD simulations it was found that $n \simeq 0.8$.

c) Granular material (sand): Friction forces are of crucial importance, hence an algorithm has to be used that allows velocity dependent forces. However, energy conservation is no longer a problem, since these are driven systems, e.g., sand flow through vertical pipes or through a hopper under the influence of gravity. Other phenomena include heap formation and size segregation in a container with an oscillating bottom.

Discussion of MD simulations of gases

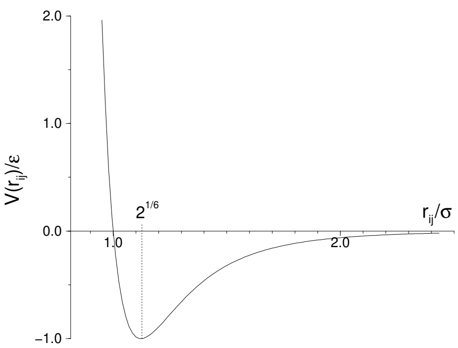
The total energy of a gas of N identical particles is

$$\mathcal{H} = \sum_{i=1}^{N} \tfrac{1}{2} m \mathbf{v}_i^2 + \sum_{\substack{i,j=1 \\ i<j}}^{N} V_{ij} \ . \tag{72}$$

Here, V_{ij} is the pair potential that describes the interaction between two particles. For gases one usually uses the Lennard-Jones or 6-12 potential

$$V_{ij} = 4\epsilon \left[\left(\frac{r_{ij}}{\sigma} \right)^{-12} - \left(\frac{r_{ij}}{\sigma} \right)^{-6} \right] \quad \text{with} \quad r_{ij} = |\mathbf{r}_i - \mathbf{r}_j| \ . \tag{73}$$

The term proportional to r_{ij}^{-6} corresponds to the van der Waals interaction of the gas molecules, whereas the first term on the right-hand side of (73) describes the repulsion of particles at small distances. The typical form of the potential is shown in the figure to the right. The characteristic time scale in the simulation is set by $\tau = [m\sigma^2/\epsilon]^{1/2}$. Typical numbers for, e.g., Argon are: $m \simeq 6.6 \times 10^{-26}$kg, $\sigma \simeq 3.4 \times 10^{-10}$m, $\epsilon \simeq 1.7 \times 10^{-21}$J that give a time scale of $\tau \simeq 2.2 \times 10^{-12}$s. The equations of motion can be cast into a dimensionless form using the transformations $t/\tau \to t$, $\mathbf{r}_i/\sigma \to \mathbf{r}_i$, $\mathbf{v}_i\tau/\sigma \to \mathbf{v}_i$, and $V_{ij}/\epsilon \to V_{ij}$. In this units the equations of motion, i.e., Newton's equations read



$$\frac{d^2\mathbf{r}_i}{dt^2} = -\nabla \sum_{\substack{j=1 \\ j\neq i}}^{N} V_{ij} = \sum_{\substack{j=1 \\ j\neq i}}^{N} \mathbf{F}_{ij} = \mathbf{F}_i \tag{74a}$$

$$\text{with} \quad \mathbf{F}_{ij} = 24 \left(\frac{2}{r_{ij}^{14}} - \frac{1}{r_{ij}^{8}} \right) \mathbf{r}_{ij} \ . \tag{74b}$$

The differential equations (74) describe a microscopic ensemble: the total energy is conserved as well as the number of particles.

Since the computation of the forces is the most time consuming part ($\sim N^2$ in general) in the numerical integration of the equations of motion (74), it is essential to use an algorithm that requires only one force calculation per time step. Furthermore, energy conservation is very important, whereas high accuracy is of minor importance: the system is fully chaotic, the accuracy is lost within a few time steps anyway. Therefore, MD simulations usually use the leapfrog algorithm or algorithms that are equivalent to the leapfrog algorithm. Clearly the Runge-Kutta algorithm is a bad choice for MD simulations. If the forces depend on the velocities, so-called predictor-corrector algorithms are used. One example is the Gear algorithm, a description of which can be found in many books on molecular dynamics, but it will not be discussed in this lecture.

A disadvantage of the leapfrog algorithm (63) is the fact that positions and velocities are calculated at different time steps. However, many quantities of interest, e.g., the total energy, require the knowledge of the positions and velocities at the same time. Therefore the leapfrog algorithm is rewritten in an algebraically equivalent form that calculates velocities and positions at the same times:

$$\mathbf{r}_i(t+\Delta t) = \mathbf{r}_i(t) + \Delta t \mathbf{v}_i(t) + \tfrac{1}{2}(\Delta t)^2 \mathbf{a}_i(t) \ , \tag{75a}$$

$$\mathbf{v}_i(t+\Delta t) = \mathbf{v}_i(t) + \tfrac{1}{2}\Delta t \left[\mathbf{a}_i(t) + \mathbf{a}_i(t+\Delta t)\right] \ . \tag{75b}$$

The algorithm (75) is called the "velocity form of the Verlet algorithm" and is the standard integration method in molecular dynamics. In the dimensionless units mentioned above the accelerations are identical to the forces $\mathbf{a}_i(t) \equiv \mathbf{F}_i(t)$. The Verlet algorithm (75) requires the storage of $\mathbf{r}_i(t)$, $\mathbf{v}_i(t)$, and $\mathbf{a}_i(t)$ for all particles $i = 1, \ldots, N$. These are $9N$ numbers for three spatial dimensions. The algorithm (75) is implemented in the following way:

(i) compute $\mathbf{r}_i(t+\Delta t)$ according to (75a).

(ii) calculate $\mathbf{v}_i(t+\frac{1}{2}\Delta t) = \mathbf{v}_i(t) + \frac{1}{2}\Delta t \mathbf{F}_i(t)$.

(iii) compute the forces $\mathbf{F}_i(t+\Delta t)$; since we do not discuss velocity dependent forces they depend only on $\mathbf{r}_i(t+\Delta t)$.

(iv) calculate $\mathbf{v}_i(t+\Delta t) = \mathbf{v}_i(t+\frac{1}{2}\Delta t) + \frac{1}{2}\Delta t \mathbf{F}_i(t+\Delta t)$.

Remarks:

(a) Use periodic boundary conditions, $x_{i+L}(t) = x_i(t)$, analogously for the other coordinates with L being the linear size of the system, i.e., L^d is the volume of the system under investigation. This minimizes boundary effects.

(b) Because of (a) the range of the potential must be smaller than $L/2$ otherwise particles can interact with themselves. Therefore, a potential cutoff must be introduced: $V(r_{ij}) = 0$ for $r_{ij} > r_c$. For the Lennard-Jones potential the choice $r_c = 2.5\sigma$ is quite common. The cutoff should be smooth, i.e., the force should be continuous at r_c otherwise spurious instabilities can occur in the numerical integration.

(c) Use neighborhood tables (Verlet tables). For each particle create a list of neighbors that are located within a sphere of radius $r_l > r_c$ around that particle. Only particles that are listed in the Verlet table of particle i are needed to compute the force $\mathbf{F}_i$. Thus, the force calculation becomes much faster. However, updating the tables is time consuming. Therefore, the radius r_l is chosen so large that the same tables can be used for several time steps (10-20): $r_l - r_c$ must be larger than the maximum displacement within that time interval. However, if r_l is chosen too large, the force computation makes the algorithm again less efficient. The optimal value for r_l depends on the specific problem. For the Lennard-Jones potential with a cutoff at $r_c = 2.5\sigma$ a value of $r_l = 2.7\sigma$ led to a speed-up by a factor of more than 2 for a system of 500 particles.

(d) If the formulae for the forces are complicated, the forces can be tabulated for some values of r_{ij}. Then cubic spline interpolation is used to obtain the force for values r_{ij} that lie between the tabulated values.

10. Random Numbers

Random numbers are needed for many applications. Here is just a short list of problems that have been discussed in this course that require the computation of random numbers:

(i) Simulation of stochastic processes: brownian motion, Langevin equations.

(ii) Pick elements randomly out of an ensemble: Monte-Carlo Integration, points are chosen at random within the volume of integration.

(iii) Set a random initial configuration: Molecular dynamics simulation, the initial state should not be completely ordered. The initial positions and velocities of the particles should fluctuate around some average values.

(iv) Most importantly: Monte-Carlo simulations, see next chapter.

10.1 What are Random Numbers?

Computers are deterministic machines, therefore any sequence of numbers is deterministic and randomness in the strict sense does not exist. The following may nevertheless serve as a working characterization:

"Definition": Random numbers (better: pseudo random numbers) are deterministic sequences of real numbers that are sufficiently uncorrelated, i.e., the way the numbers are generated must not influence the results of the simulation.

From this definition it is already clear that "good" and "bad" depends very much on the application, for which the random numbers are needed. A random number generator that is good enough for one application, e.g., to generate a random initial configuration for a molecular dynamics simulation, may be extremely bad for another, e.g., a Monte-Carlo simulation. However, there are several criteria a good generator has to obey. Here, only RNG that generate uniformly distributed random numbers will be discussed, because they are most important for applications in physics. Different distributions can be generated using a uniformly distributed random numbers, see Numerical Recipes.

(i) Since the number of bits in a computer word is finite, any random number generator (RNG) has a finite period. This period of the RNG must be much larger than the number of random numbers that is used in the program.

(ii) Correlations must be small, i.e., not only must the random numbers be uniformly distributed in the interval $[0, 1)$ (even the worst random number generators do not fail in this respect), but also pairs of random numbers (x_n, x_{n-s}), $s = 1, 2, 3, \ldots$ should be uniformly distributed within the square $[0, 1) \times [0, 1)$.

(iii) A RNG must be fast: in some Monte-Carlo simulations up to 40% of the cpu-time is spent for the generation of random numbers.

10.2 Random Number Generators

Linear congruential RNGs

Linear congruential random number generators are the most common RNGs. They generate random numbers according to the following formula:

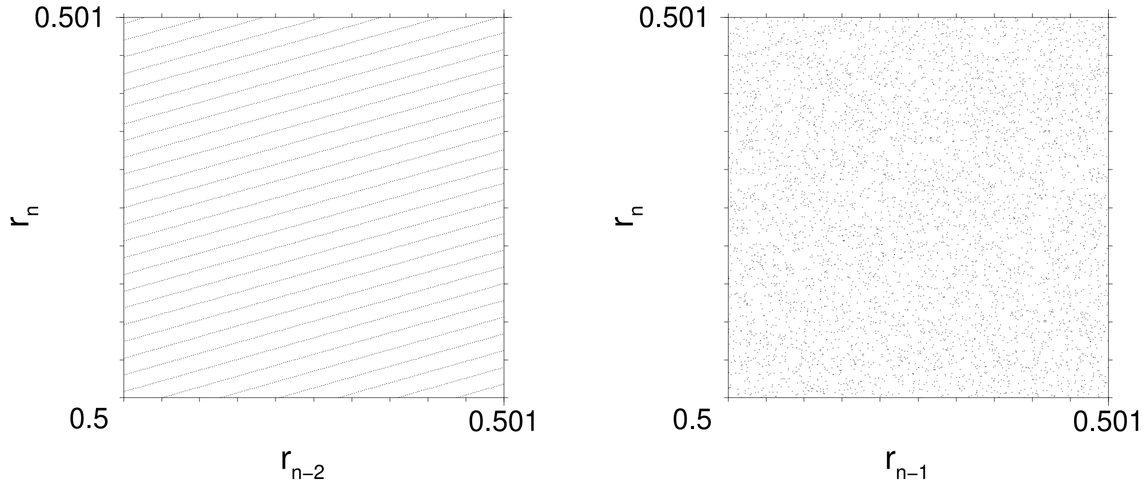
$$\mathrm{LCG}(a,c,m): \qquad I_n = (a\, I_{n-1} + c)_{\mathrm{mod}\ m}\ . \tag{76}$$

I_n, a, c, and m are integers; m is called the modulus, a is the multiplier, and c the increment. Almost all system supplied RNG are linear congruential RNG. Furthermore, the worst RNGs also belong to this group. From this one can already conclude that on should *never* use the system supplied RNG, but for the simplest applications. Since the next random number of the sequence generated from (76) depends only on the previously generated random number, it is immediately clear that the maximum period of the generator $\mathrm{LCG}(a,c,m)$ is m. However, a and c have to be chosen very carefully in order to reach this maximum period. An example for such a generator is $\mathrm{LCG}(69069, 1, 2^{32})$ that has a period of $m = 2^{32}$. This is also the maximum period that can be reached using a linear congruential RNG using integers on a 32 bit computer. Furthermore, using $m = 2^{32}$ has the advantage that it is not necessary to implement the modulo operation in (76), since the usual integer overflow arithmetic discussed in chapter 3.2 is equivalent to an operation modulo 2^{32}. Therefore, the $\mathrm{LCG}(a,c,2^{32})$ can be programmed as $I_n = a\, I_{n-1} + c$ and the conversion to a real number between 0 and 1 is obtained through $r_n = I_n / r_{\max} + 0.5$ with $r_{\max} = 2.0^{32}$.

Why are Linear Congruential RNGs so Bad?

Beside the small period of maximum $m = 2^{32}$, that is easily reached on today's workstations, linear congruential RNGs have large correlations: They show the so-called Marsaglia effect: Points $(r_{n+1}, r_{n+2}, \ldots, r_{n+d})$ are not distributed uniformly within the d-dimensional unit cube, but lie on *at most* $m^{1/d}$ hyperplanes of dimension $d-1$. Thus, for $d = 2$ the square $[0,1) \times [0,1)$ is not filled uniformly, but parallel lines running across the square are obtained. Between those lines no pairs (r_n, r_{n+1}) can be found, see the figure below.

At least the numbers of linear congruential generators should be shuffled: First generate an array of random numbers, $x_i = \mathrm{LCG}(a,c,m)$, $i = 1, \ldots, L$ with, e.g., $L = 97$. Then draw an additional random number y. Convert y to an integer between 1 and L, $n = \mathrm{int}(Ly) + 1$. Use x_n as the result of the RNG, replace x_n by the next number from the generator, $x_n = \mathrm{LCG}(a,c,m)$, and use the same random number to update y as well. This new value of y is then used to compute a new n, etc. This shuffling algorithm does not change the period of the LCG, it only changes the sequence of the random numbers. A better way of destroying correlations in a linear congruential RNG uses a second LCG to shuffle the numbers of the first generator: $x_i = \mathrm{LCG}(a_1, c_1, m_1)$ and $y = \mathrm{LCG}(a_2, c_2, m_2)$. Again, y is used exclusively to determine the element of the array $(x_1, x_2, \ldots, x_L)$ that then is used as the result of the combined generator. The array is filled up using the first generator as before. However, now two random numbers have to be computed per returned random number. Consequently the random number generator will be slower, and in most cases it is more reasonable not to use a linear congruential generator at all, but use one of the better generators that are discussed in the next paragraph.



Correlations in linear congruential random number generators: The left figure shows pairs (r_{n-2}, r_n) of random numbers generated from the LCG$(69069, 1, 2^{32})$ generator. All 2^{32} random numbers were computed. The Marsaglia planes are clearly visible. If pairs (r_{n-1}, r_n) are plotted the picture is even worse: The planes are not filled: The pairs appear at regularly spaced discrete x values [see, K. G. Hamilton, Comput. Phys. Commun. **75**, 105 (1993)]. For comparison the left graph shows the result for the combination generator $\mathrm{F}(2, 1, *)$ and $\mathrm{F}(3, 1, -)$ for 2^{32} generated random numbers. No distinct pattern is visible in this case.

Better Random Number Generators

[Reference: G. A. Marsaglia, in *Computer Science and Statistics: The Interface*, L. Billard (ed.), (Elsevier, Amsterdam, 1985), p. 3]

A large class of random number generators are called lagged Fibonacci generators

$$\mathrm{F}(r, s, \otimes) \; : \qquad I_n = (I_{n-r} \otimes I_{n-s})_{\mathrm{mod}\ m} \qquad (\text{with } r > s) \; , \tag{77}$$

where $\otimes$ is one of the operators $+$, $-$, $*$, or $\oplus$ [bitwise exclusive-or (xor)]. The maximum period of these generators is much larger than 2^{32}, the maximum period that can be obtained for linear congruential generators. It can be shown that for $m = 2^n$, n integer, the period is $(2^r - 1)m/2$ for $+$ and $-$, $(2^r - 1)m/8$ for $*$ on *odd* integers, and $2^r - 1$ for $\oplus$.

One example of such a generator is the Kirkpatrick-Stoll generator [Kirkpatrick, Stoll, J. Comp. Phys. **40**,517 (1981)]

$$I_n = I_{n-250} \oplus I_{n-147} \; . \tag{78}$$

Lagged Fibonacci generators that use the exclusive-or operation $\oplus$ are also called shift register or Tausworthe generators. The period of the generator (78) is extremely large: $2^{250} - 1 \simeq 10^{75}$. To initialize the generator 250 random integers are needed. A linear congruential generator like LCG$(69069, 1, 2^{32})$ can be used to accomplish this task. The Kirkpatrick-Stoll generator is very popular in the physics community. It is very fast and only recently reports have been published that hint at hidden correlations in this generator [W. Selke *et. al.*, JETP Letters, **58**, 665 (1993)]. One has to be careful when implementing lagged Fibonacci generators using $\oplus$: For small lags r the generators $\mathrm{F}(r, s, \oplus)$ are extremely bad, see Marsaglia's article.

Better generators can also be obtained by combining two generators. See Numerical Recipes for a combination generator of two LCGs. The disadvantage of that generator is its very slow speed. Here I give a combination generator of the two Fibonacci generators $\mathrm{F}(2, 1, *)$ and $\mathrm{F}(3, 1, -)$ that has a period of $\simeq 10^{19}$, is very fast, and needs only 5 integers to initialize:

$$\begin{aligned} I_n &= (I_{n-2} * I_{n-1})_{\mathrm{mod}\ 2^{32}} \qquad \text{on odd integers} \\ J_n &= (J_{n-3} - J_{n-1})_{\mathrm{mod}\ 2^{30}-35} \\ K_n &= (I_n - J_n)_{\mathrm{mod}\ 2^{32}}\ . \end{aligned} \tag{79}$$

The returned random integer is K_n that lies between -2^{31} and $2^{31}-1$ and can be transformed to a real number within $[0, 1)$ in the usual way. A faster, however machine dependent method of transforming an integer to a real number between 0 and 1 pastes the random integer to the mantissa of a floating point number and adjusts the exponent part of the floating point number so that the result lies between 0 and 1. For machines that conform to the IEEE convention this is done by copying the random integer to the second word of a double precision number. Since the mantissa of a double precision number is 52 bits long, a 32 bit integer leaves 20 bits empty. Therefore, the mantissa of the double precision number is of the form $1 + i_{31}2^{-21} + i_{30}2^{-22} + \ldots + i_0 2^{-52}$, where i_k are the bits of the random integer. Thus by setting the exponent value of the double precision number to $e = 20$ the double precision number attains the value $2^{20} + r$, with r lying between 0 and 1. Subtracting 2^{20} from this number gives the desired random number r. For an implementation of this method see the programs `randv.C` or `randv.f90`, resp.

11. Monte-Carlo Simulation

11.1 The Ising Model (E. Ising, 1925)

Consider a system of spins $S_i = \pm 1$ on a d-dimensional simple cubic lattice, $i = 1, \ldots, L^d$. This can be regarded as a model for a magnet with a large anisotropy so that all spins point preferably in the $+$ or $-$ z direction. But this model also describes the binary alloy that was investigated in assignment #3: The density ρ_i is defined to be $\rho_i = 1$, if there is an A atom at site i, and $\rho_i = 0$ otherwise. Consequently, the quantity $S_i = 2\rho_i - 1$ takes only values ± 1, thus the binary alloy is described by the Ising model as well. In general any system, where the microscopic variables can take only two values, can be mapped to the Ising model.

The energy of the Ising model for a specific configuration of spins is given by

$$\mathcal{H} = -J \sum_{\langle i,j \rangle} S_i S_j - h \sum_i S_i \ . \tag{80}$$

Here, $\langle i, j \rangle$ denotes a summation over all nearest-neighbor pairs (dL^d pairs in total) and h is an external magnetic field. If $h = 0$ and $J > 0$ a parallel alignment of spins is energetically favorable, thus $J > 0$ describes a ferromagnet. $J < 0$ is a model for an antiferromagnet: If the lattices is subdivided into two sublattices analogous to a checker board lattice so that all neighbors of a spin are located in the other sublattice, then the preferred configuration has the spins of one sublattice pointing into the opposite direction than the spins of the other sublattice. This discussion is correct at zero temperature $T = 0$. For higher temperatures there will be thermal excitations so that for $J > 0$ not all spins point into the same direction. Thus, the questions Monte-Carlo simulations are trying to answer are: Given a finite temperature T and possibly an external field h, what is the state of the system? What is the magnetization, the energy? What is the magnetic susceptibility and the specific heat?

11.2 Thermodynamic Equilibrium, Master Equation, Detailed Balance

Let $\mathcal{C} = (S_1, \ldots, S_N)$, $N = L^d$, be a configuration of spins. In the thermodynamic equilibrium the probability of finding the configuration $\mathcal{C}$ is given by the Boltzmann distribution

$$P_{\text{eq}}(\mathcal{C}) = \frac{1}{Z} e^{-\beta \mathcal{H}(\mathcal{C})} \ , \quad \beta = \frac{1}{k_B T} \tag{81}$$

with $k_B \simeq 1.38 \times 10^{-23}$ J/K being Boltzmann's constant. $Z = \sum_{\{\mathcal{C}\}} e^{-\beta \mathcal{H}(\mathcal{C})}$ is the partition function and the sum $\sum_{\{\mathcal{C}\}} \equiv \sum_{S_1 \pm 1} \sum_{S_2 \pm 1} \cdots \sum_{S_N \pm 1}$ runs over all 2^N configurations. If Z is known , many quantities of interest can be calculated, e.g., expectations values such as the average energy

$$\langle \mathcal{H} \rangle = -\frac{J}{Z} \sum_{\{\mathcal{C}\}} \sum_{\langle i,j \rangle} S_i S_j e^{-\beta \mathcal{H}(\mathcal{C})} = -J k_B T \frac{\partial}{\partial J} \log Z \ .$$

Since 2^N is a huge number Z cannot be calculated directly not even with the best computers. Therefore, different numerical methods must be used to calculate expectation values.

In a Monte-Carlo simulation a sequence of configurations $\mathcal{C}_1 \to \mathcal{C}_2 \to \mathcal{C}_3 \cdots$ is generated that allow an approximate evaluation of the sum over the configurations. The idea is that only those configurations have to be considered that have a large Boltzmann weight $e^{-\beta\mathcal{H}(\mathcal{C})}$. Most of the 2^N configurations in fact will have a very small probability so that their contribution can be safely neglected, e.g., at low temperatures and $J > 0$ most of the spins in a typical configuration will still be parallel aligned, therefore configurations with many antiparallel spin pairs can be neglected. In a Monte-Carlo simulation the transition rates that determine, which configuration $\mathcal{C}'$ can be reached from the current configuration $\mathcal{C}$, are constructed so that only those typical configurations are generated in the sequence of configurations that are important to estimate the quantities of interest.

In order to determine reasonable transition rates, first the dynamical evolution of the system has to be defined. There are two physically different situations:

(i) Glauber dynamics or spin-flip dynamics: In an elementary time step a spin at site i is flipped: $S_i \to -S_i$. For this kind of dynamics the total magnetization is not conserved.

(ii) Kawasaki dynamics or spin-exchange dynamics: In an elementary time step a spin at site i is exchanged with one of its neighbors at site j: $S_i \leftrightarrow S_j$. Consequently, the total magnetization $\sum_i S_i$ remains constant. This type of dynamics is the correct description for the simulation of, e.g., binary alloys, since in that case the concentration of A and B atoms has to remain constant.

These are the microscopic mechanisms that lead to changes from one configuration to another. Still one has to determine reasonable transition rates for these changes. Consider an equation of motion for the time-dependent probability distribution $P(\mathcal{C},t)$. The dynamics must converge to the Boltzmann distribution, i.e., $\lim_{t\to\infty} P(\mathcal{C},t) = P_{\mathrm{eq}}(\mathcal{C})$. If the transition rate from configuration $\mathcal{C}$ to $\mathcal{C}'$ is called $w(\mathcal{C} \to \mathcal{C}')$ the equation of motion for the probability distribution takes the form

$$\frac{\partial}{\partial t} P(\mathcal{C},t) = -\sum_{\{\mathcal{C}'\}} w(\mathcal{C} \to \mathcal{C}')P(\mathcal{C},t) + \sum_{\{\mathcal{C}'\}} w(\mathcal{C}' \to \mathcal{C})P(\mathcal{C}',t) \ . \tag{82}$$

Eq. (82) is called a master equation. The first, negative term on the right-hand side describes transitions that destroy the configuration $\mathcal{C}$ due to transitions to any other configuration $\mathcal{C}'$. This is the lost term. The second, positive term describes transitions that lead from any configuration $\mathcal{C}'$ to the configuration $\mathcal{C}$, this is the gain term. In the limit $t \to \infty$ we must have $\partial P_{\mathrm{eq}}(\mathcal{C})/\partial t = 0$ and therefore

$$\sum_{\{\mathcal{C}'\}} w(\mathcal{C} \to \mathcal{C}')P_{\mathrm{eq}}(\mathcal{C}) \stackrel{!}{=} \sum_{\{\mathcal{C}'\}} w(\mathcal{C}' \to \mathcal{C})P_{\mathrm{eq}}(\mathcal{C}') \ . \tag{83}$$

Eq. (83) is a constraint the transition rates w have to obey, if the generated configurations in the simulation are to relax to the correct equilibrium state. In the simulation of nonequilibrium dynamics the system may relax to a steady-state that may not have anything to do with the equilibrium state. In these cases the constraint (83) does not apply and there are many more choices for the transition rates.

A sufficient, but not necessary, condition for the transition rates w in order to satisfy Eq. (83) is the condition of detailed balance

$$w(\mathcal{C} \to \mathcal{C}')P_{\mathrm{eq}}(\mathcal{C}) = w(\mathcal{C}' \to \mathcal{C})P_{\mathrm{eq}}(\mathcal{C}') \ . \tag{84a}$$

For most configurations we have $w(\mathcal{C} \to \mathcal{C}') = 0 = w(\mathcal{C}' \to \mathcal{C})$. For those configurations $\mathcal{C}$, $\mathcal{C}'$ with $w(\mathcal{C}' \to \mathcal{C}) \neq 0$ the Eqs. (81) and (84a) lead to the requirement

$$\frac{w(\mathcal{C} \to \mathcal{C}')}{w(\mathcal{C}' \to \mathcal{C})} = \exp\left\{-\beta\left[\mathcal{H}(\mathcal{C}') - \mathcal{H}(\mathcal{C})\right]\right\} \ . \tag{84b}$$

Transition rates that obey (84b) will generate the Boltzmann distribution (81). In principle the condition (84b) constrains the transition rates much more than the weaker condition (83). However, for almost all models it is impossible to prove that transition rates that do not obey the detailed balance condition (84b) nevertheless obey the more general constraint (83). Most likely transition rates that do not obey (84b) will in fact not satisfy the constraint (83). Therefore, in Monte-Carlo simulations the transition rates are almost always chosen to obey the stricter constraint of detailed balance (84b).

An additional property the transition rates must have is the property of ergodicity: from any initial configuration it must be possible to reach any other allowed configuration.

11.3 Metropolis Algorithm

The Metropolis algorithm uses the following choice for the transition rates, if the transition $\mathcal{C} \to \mathcal{C}'$ is allowed:

$$w(\mathcal{C} \to \mathcal{C}') = \begin{cases} 1 & \text{if } \mathcal{H}(\mathcal{C}') \leq \mathcal{H}(\mathcal{C}) \\ \exp\left\{-\beta\left[\mathcal{H}(\mathcal{C}') - \mathcal{H}(\mathcal{C})\right]\right\} & \text{if } \mathcal{H}(\mathcal{C}') > \mathcal{H}(\mathcal{C}) \end{cases} . \tag{85}$$

Detailed balance (84b) is trivially obeyed. If $\Delta E = \mathcal{H}(\mathcal{C}') - \mathcal{H}(\mathcal{C})$ denotes the change in the energy due to the transition the metropolis rates take the simple form $w(\mathcal{C} \to \mathcal{C}') = 1$ if $\Delta E < 0$ and $w(\mathcal{C} \to \mathcal{C}') = e^{-\beta\Delta E}$ otherwise. Hence, any transition that lowers the energy of the system is accepted with probability 1, whereas transitions that increase the energy occur with a much lower probability that decreases rapidly with decreasing temperature T.

A different possibility for the transition rates w are the Glauber rates

$$w(\mathcal{C} \to \mathcal{C}') = \frac{1}{1 + e^{\beta\Delta E}} = \frac{1}{2}\left(1 - \tanh\frac{\beta\Delta E}{2}\right) \tag{86}$$

that use the same formula for $\Delta E > 0$ and $\Delta E < 0$. The relaxation towards the equilibrium state for the Glauber rates (86) is not as fast as for the Metropolis rates (85), however, the dynamical evolution is modeled more realistically, e.g., a transition between two states with the same energy occurs with probability 1/2. Thus, if only equilibrium properties shall be studied, one usually uses Metropolis rates, whereas for investigations of dynamical evolutions one may consider to use the Glauber rates.

Depending on the physics of the problem at hand one still has to decide on the kind of dynamics that must be used: Either nonconserved dynamics (Glauber or spin-flip dynamics) that are appropriate for, e.g., magnetic systems, or conserved dynamics (Kawasaki or exchange dynamics) that are appropriate for, e.g., binary alloys. In both cases two consecutively generated configurations $\mathcal{C}$ and $\mathcal{C}'$ differ in the values of only one (Glauber) or two (Kawasaki) spins. For all other configurations the transition rates $w(\mathcal{C} \to \mathcal{C}')$ are zero.

Calculation of Averages

The statistical average of a quantity A is given by

$$\langle A \rangle = \frac{1}{Z} \sum_{\{\mathcal{C}\}} A(\mathcal{C}) e^{-\beta \mathcal{H}(\mathcal{C})} \tag{87}$$

as explained above. For, e.g., the magnetization ($A = M$) we have $M(\mathcal{C}) = \sum_{i=1}^{L^d} S_i$. In the Monte-Carlo simulation a "Markov chain" of configurations is generated, $\mathcal{C}_1 \to \mathcal{C}_2 \to \mathcal{C}_3 \to \cdots$, however, the transitions from one configuration to the next occur with the probability $e^{-\beta \Delta E}$, therefore the configurations that are generated in the simulation do *not* occur with the same probability, but with a probability of $e^{-\beta \mathcal{H}(\mathcal{C})}$. This is the whole idea of *importance sampling* in the Monte-Carlo method. Only those configurations that carry a large weight in the sum in Eq. (87) are generated. Furthermore, since these configurations are generated with a probability of $e^{-\beta \mathcal{H}(\mathcal{C})}$ the calculation of expectation values reduces to the calculation of mean values

$$\langle A \rangle \simeq \frac{1}{n} \sum_{l=1}^{n} A(\mathcal{C}_l) \ , \tag{88}$$

where $\mathcal{C}_l$ is one particular configuration occurring in the Markov chain. Usually not all configurations that are generated in a simulation are taken into account in the summation (88) because of correlations, see below.

Implementation of a Monte-Carlo Simulation

The basic Monte-Carlo step consists of two parts:
(i) propose a new configuration $\mathcal{C}'$.
(ii) accept or reject the transition $\mathcal{C} \to \mathcal{C}'$.

The full transition rate $w_{\mathrm{tot}}(\mathcal{C} \to \mathcal{C}')$ is therefore the product of two parts

$$w_{\mathrm{tot}}(\mathcal{C} \to \mathcal{C}') = w_p(\mathcal{C}'|\mathcal{C})\, w(\mathcal{C} \to \mathcal{C}') \ . \tag{89}$$

The factor $w_p(\mathcal{C}'|\mathcal{C})$ is the probability that the configuration $\mathcal{C}'$ is proposed, if the current configuration is $\mathcal{C}$, and $w(\mathcal{C} \to \mathcal{C}')$ is the probability that the transition is accepted as described above. Thus, the detailed balance condition becomes

$$\frac{w_p(\mathcal{C}'|\mathcal{C})}{w_p(\mathcal{C}|\mathcal{C}')} \frac{w(\mathcal{C} \to \mathcal{C}')}{w(\mathcal{C}' \to \mathcal{C})} = e^{-\beta \Delta E} \ .$$

For Glauber and Kawasaki dynamics $w_p(\mathcal{C}'|\mathcal{C}) = w_p(\mathcal{C}|\mathcal{C}')$, e.g., consider Glauber dynamics and the two configurations that differ only in the value of one spin at site i: $w_p(\mathcal{C}'|\mathcal{C}) = L^{-d}$ since the probability that a spin is proposed for a flip is the same for all spins. Therefore the detailed balance condition reduces to (84b) for these algorithms. The decision to accept the transition $\mathcal{C} \to \mathcal{C}'$ is made by drawing a uniformly distributed random number z out of the interval $[0, 1)$. If $z < e^{-\beta\Delta E}$ the transition is accepted.

The Swendsen-Wang algorithm is a cluster-flip algorithm that proposes a new configuration according to the probability $w_p(\mathcal{C}'|\mathcal{C}) = e^{-\beta\Delta E}$. In that case the acceptance of the transition is trivial, but detailed balance is satisfied nevertheless.

Even for Glauber and Kawasaki dynamics the sequence, in which the spins are flipped/exchanged, must still be determined. There are several possibilities:

(i) Randomly: In each step another random number ζ is drawn. The spin S_i that is proposed for the spin flip is determined by $i = \mathrm{int}(L^d\zeta) + 1$. This is the only correct method, if one is interested in the dynamical evolution of the system.

If only equilibrium properties are to be studied, there are other possibilities that avoid the computation of additional random numbers.

(ii) Sequential: The spins are selected in a deterministic sequence, $i = 1, 2, 3, \ldots, L^d$.

(iii) Sublattice updates (only for Glauber dynamics): Divide the lattice into two sublattices according to the checker board rule: all nearest neighbors of a spin lie in the other sublattice. Therefore, all spins belonging to one sublattice can be updated in parallel. This is the preferred algorithm on vector machines or machines that allow parallel processing. This method also allows for a fast implementation of the periodic boundary conditions (see next paragraph) even on a single-processor workstation.

One also has to decide, which boundary conditions to use: free, periodic, or helical boundary conditions. This choice influences the finite-size effects, i.e., the corrections in the expectation values due to the finite size of the system. Most often periodic boundary conditions are used since they minimize finite-size effects. Free boundary conditions are in almost all cases not a good choice. Helical boundary conditions, e.g., for a two-dimensional lattice use $S_{i,L+1} = S_{i+1,1}$ instead of $S_{i,L+1} = S_{i,1}$, which is the correct choice for periodic boundary conditions, are very fast on the computer, since they allow to store a two-dimensional field $S_{i,j}$ in an one-dimensional array $S_{(i-1)L+j}$ and the boundary conditions on j are implemented automatically. For the i direction periodic boundary conditions are used. However, the size dependence of the finite-size corrections (see, Sec. 11.4) is not completely clear.

<u>Example:</u> Ising model with $J > 0$, $h = 0$, Glauber dynamics with Metropolis rates, periodic boundary conditions, sequential updates.

(i) Choose an initial configuration. It has turned out that the best choice is usually a completely ordered $T = 0$ configuration, even for simulations at high temperatures, where disordered configurations are more important. Using an ordered $T = 0$ configuration, e.g., $S_i = 1$ for all $i = 1, \ldots, L^d$ avoids the formation of several large domains of plus and minus spins. Such configurations correspond to metastable states that are very hard to destroy, since it is necessary to flip a whole domain of spins in the process, which is unlikely to occur especially at low temperatures. Therefore the simulation can get stuck in one part of the configuration space and important contribution from other states are missed so that the calculated expectation values can be severely biased and inaccurate.

(ii) Set the magnetization $M = \sum_i S_i$ and the energy $E = -J\sum_{\langle i,j\rangle} S_i S_j$ to the values that correspond to the initial configuration.

(iii) The basic Monte-Carlo step: For a spin S_i, $i = 1, \ldots, L^d$ calculate the energy change that is obtained if the spin would be flipped,

$$\Delta E = \mathcal{H}(\mathcal{C}') - \mathcal{H}(\mathcal{C}) = 2JS_i \sum_{\langle j \rangle_i} S_j \ . \tag{90}$$

Here the sum runs over all nearest neighbors $\langle j \rangle_i$ of the spin at site i. Determine the Boltzmann factor $e^{-\beta\Delta E}$. Draw a random number z. If $z < e^{-\beta\Delta\mathcal{H}}$ flip the spin: $S_i \to -S_i$ and update the magnetization and the energy E. If $z > e^{-\beta\Delta E}$ leave everything unchanged. Perform this step consecutively for all sites $i = 1, \ldots, L^d$. One run through the whole system, i.e., one attempt per spin, is called a Monte-Carlo step (MCS).

(iv) Starting from the initial configuration perform λ_0 Monte-Carlo steps without calculating averages. It takes a certain time to relax to the equilibrium state. This time in fact depends on the temperature and even diverges if T is close to the critical temperature T_c (critical slowing down, see next section).

(v) Perform $n\lambda$ more MCS. After each λ MCS the averages of the quantities of interest are updated, e.g.,

$$\langle M \rangle = \frac{1}{n} \sum_{l=1}^{n} |M_l| \ ,$$

where M_l is the actual magnetization after $\lambda_0 + l\lambda$ MCS. The number λ of MCS between two updates must be large enough so that the configuration after λ MCS is sufficiently uncorrelated with the configuration that was used for the previous update of the averages. Again, reasonable values for λ depend on the temperature and the size of the system: λ must be larger the closer T is to the critical temperature T_c and the larger the system size L.

Remarks

Most of the computing time is spent in the basic Monte-Carlo step, point (iii) above. Therefore, special care should be taken to program this step as efficient as possible.

(a) For many models the number of values that are obtained for ΔE is finite, hence the transition rates $w(\mathcal{C} \to \mathcal{C}')$ can (and should) be calculated before the start of the simulation and stored in an array, e.g., the energy change ΔE (90) for the two-dimensional Ising model can be written as

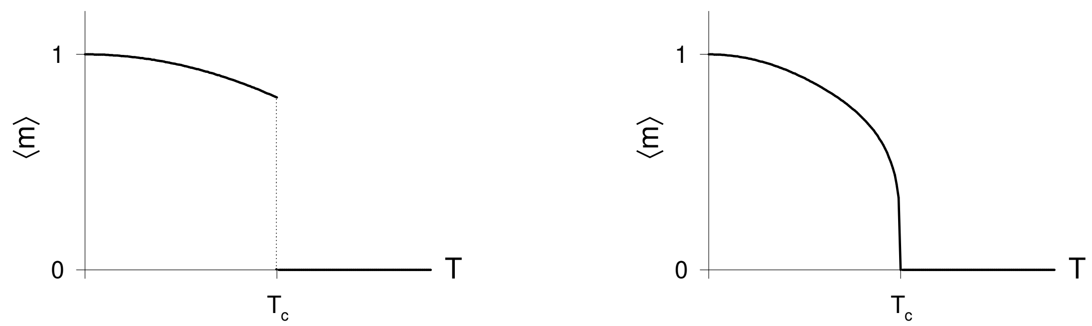
$$\Delta E = 2J\, n_e(\mathcal{C}, i) \qquad \text{with} \qquad n_e(\mathcal{C}, i) \equiv S_i \sum_{\langle j \rangle_i} S_j = -4, -2, 0, 2, 4 \ .$$

Thus, the Boltzmann weights $e^{-\beta\Delta E}$ can be calculated beforehand and stored in an array b_l, $l = -4, \ldots, 4$ so that in the actual simulation only $n_e(\mathcal{C}, i)$ has to be calculated in order to determine the transition rates $b_{n_e(\mathcal{C},i)}$.

(b) The periodic boundary conditions should not be implemented using `if` statements or modulo operations. Instead, one can use buffer layers or use auxiliary arrays that hold the indices of the nearest neighbors of each spin. The first method is implemented in, e.g., two dimensions with $i = (x, y)$ by enlarging the lattice artificially so that x and y run from 0 to $L + 1$. The lattice is subdivided into two sublattices according to the checker board rule and all spins belonging to one sublattice are updated first. Every time before the spins of the other sublattice are updated the boundary spins of the current sublattice $S_{1,y}$, $S_{x,1}$, $S_{L,y}$, and $S_{x,L}$ are copied to $S_{L+1,y}$, $S_{x,L+1}$, $S_{0,y}$, and $S_{0,x}$. The spins $S_{x,y}$ for $x, y = 1, \ldots L$ can then be updated without problem. The second method uses integer fields that hold the coordinates of the neighboring sites, e.g., set $p_x = x + 1$ for $1 \leq x < L$ and $p_L = 1$. Then $S_{x+1,y}$ can be accessed as $S_{p_x,y}$.

11.4 Phase Transitions, Critical Exponents, Finite-Size Scaling

At low temperatures the preferred state of the Ising model is an ordered state: almost all spins are aligned in one direction. The magnetization $M = \sum_{i=1}^{L^d} S_i$ is nonzero. At high temperatures the fluctuations due to thermal excitations become so large that the order is destroyed and the magnetization is zero. In any case the system is in a state that minimizes the free energy $F = E - TS$, thus at small T this minimum is reached by minimizing the energy. At high temperatures the system prefers disordered states since the number of these states is much larger and therefore the entropy S is increased. Between these two regimes there is a sharp phase transition at a certain temperature T_c, so that $|M| > 0$ for $T < T_c$ and $M = 0$ for $T \geq T_c$. The magnetization is called the order parameter of the phase transition. One distinguishes two types of phase transitions: first-order transitions, where the first derivative of the free energy is discontinuous, i.e., the order parameter jumps from a finite value to zero at the transition, and second-order phase transitions, where the order parameter goes to zero continuously, but second derivatives of the free energy such as the susceptibility diverge. Whereas first-order phase transitions depend on the details of the



Temperature dependence of the order parameter at first-order (left) and second-order (right) phase transitions. T_c indicates the critical temperature.

system second-order phase transitions show universal behavior, e.g., the magnetization M, the susceptibility χ, and the specific heat c behave as a function of temperature as

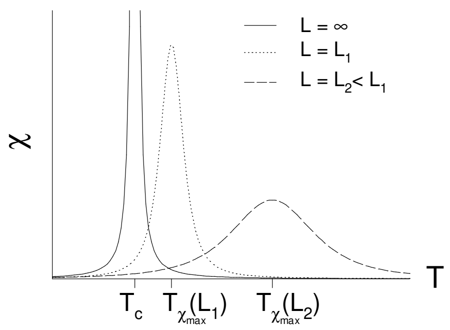
$$\langle M(T)\rangle \sim (T_c - T)^{\beta} \qquad (T < T_c)\ , \tag{91a}$$

$$\chi(T) = \frac{1}{k_B T L^d}\left[\langle M^2\rangle - \langle M\rangle^2\right] \sim |T_c - T|^{-\gamma}\ , \tag{91b}$$

$$c(T) = \frac{1}{k_B T^2}\left[\langle E^2\rangle - \langle E\rangle^2\right] \sim |T_c - T|^{-\alpha}\ . \tag{91c}$$

The Ising model has a second-order phase transition for $h = 0$ as a function of temperature. For the two-dimensional model the critical exponents β, γ, and α and the critical temperature can be determined exactly (Onsager 1944), e.g., $\sinh \frac{2J}{k_B T_c} = 1$. The critical exponents β, γ, and α do not depend on the details of the system but only on the symmetries of the microscopic variables, i.e., they remain the same if, e.g., next-nearest neighbor interactions are included in the Hamiltonian. However, they do change, if the spins are allowed not only to point into the $\pm z$ direction, but are allowed to rotate freely in all directions (Heisenberg model). Thus, by measuring critical exponents it is possible to determine the nature of the microscopic elements of the system under investigation.

The divergence of, e.g., the susceptibility at the critical temperature T_c occurs only in an infinite system, there are no singularities in a finite system, instead the singularities appear rounded for finite system sizes L. Furthermore, the maximum of the susceptibility is shifted away from the critical temperature of the infinite system as illustrated in the figure to the right. However, as the system size is increased the maximum of the susceptibility increases as well, in fact in a characteristic way that depends on the values of the critical exponents. Thus this kind of *finite-size scaling* allows an accurate determination of critical exponents in a Monte-Carlo simulation.



Schematic graph of the susceptibility χ as a function of temperature T for different system sizes L.

To understand finite-size scaling an additional length, the correlation length

$$\xi(T) \sim |T - T_c|^{-\nu} \tag{92}$$

that diverges at the critical temperature with an critical exponent ν must be introduced. The divergences of the susceptibility and of the specific heat have their origin in the divergence of the fluctuations of the magnetization and the energy at the critical point. The correlation length measures the size of a typical fluctuation in the system, for $T > T_c$ one typically finds that the correlation function of the magnetization behaves as $\langle M(\mathbf{r}+\mathbf{x})M(\mathbf{x})\rangle \sim e^{-r/\xi(T)}$. At the critical point the correlation length diverges and this behavior is no longer valid. In a finite system, however, the size of the fluctuations is limited by the system size; this leads to the rounding of the singularities. To study these effects we make the finite-size scaling ansatz, here written down for the susceptibility,

$$\chi(T, L) = L^{y_\chi} Q(L/\xi(T)) \ .$$

In the limit $L \to \infty$ this equation must reproduce the relation (91b), i.e., the L dependent terms must cancel. This requires that $Q(x) \sim x^{-y_\chi}$ for $x \gg 1$ and $y_\chi = \gamma/\nu$ because of (92). Therefore the finite-size scaling ansatz takes the form

$$\chi(T, L) = L^{\gamma/\nu} Q(L/\xi(T)) \ . \tag{93}$$

The scaling function Q describes the change in the functional dependence of $\chi(T)$ from the result for the infinite system size (91b) to a form, where the singularity is more and more rounded as the system size decreases. The maximum $\chi_{\max}(L)$ is determined by the relation

$\partial\chi(T,L)/\partial T|_{T=T_{\chi_{\max}}} = 0$ or, equivalently, $Q'(L/\xi(T_{\chi_{\max}})) = 0$. The latter relation can be reformulated as $\xi(T_{\chi_{\max}}) = x_0 L$, where x_0 is the solution of the equation $Q'(x) = 0$. From this relation the shift of the maximum from the critical temperature

$$|T_c - T_{\chi_{\max}}(L)| \sim L^{-1/\nu} \tag{94}$$

is obtained. For $T = T_{\chi_{\max}}(L)$ the scaling function Q reduces to a constant and we therefore obtain

$$\chi_{\max}(L) \sim L^{\gamma/\nu} \tag{95}$$

and analogously for the specific heat per spin

$$c_{\max}(L)/L^d \sim L^{\alpha/\nu} \ . \tag{96}$$

Similar scaling relations can be derived for many other quantities, e.g., the magnetization, etc. These scaling relations (94), (95), and (96), allow the determination of the critical exponents ν, γ, and α in a Monte-Carlo simulation. In order to obtain high accuracies it is not really necessary to simulate huge system sizes, but to do the simulation for several different system sizes with a spread in the values of L as large as possible.

The order parameter $\langle M \rangle$ for $T < T_c$ cannot be obtained in a Monte-Carlo simulation by just averaging over the values of $M = \sum_{i=1}^{L^d} S_i$, since for $h = 0$ the configuration that has all spins reversed occurs with the same probability. Thus, averaging over all configurations results in an average of zero. In an infinite system such a reversal of all spins does not occur within finite time, since it requires an intermediate state with $M = 0$ that has a probability of zero. To avoid this artificial averaging to zero in a simulation one can obtain the order parameter for $h = 0$, i.e., the spontaneous magnetization, by calculating the average

$$\langle M(T) \rangle = \frac{1}{n}\sum_{l=1}^{n} |M_l| = \frac{1}{n}\sum_{l=1}^{n} \left| \sum_{i=1}^{L^d} S_i \right| \ . \tag{97}$$

Clearly, in the thermodynamic limit $L \to \infty$ this quantity must converge to the correct value of the spontaneous magnetization.

The critical temperature T_c is most accurately determined by studying the fourth order cumulant

$$U(T,L) = 1 - \frac{\langle m^4 \rangle}{3\langle m^2 \rangle^2} \tag{98}$$

with $m = M/L^d$. At low temperatures, $T \ll T_c$, $\langle m^4 \rangle \approx \langle m^2 \rangle^2$ and therefore $U \simeq 2/3$. At high temperatures, $T \gg T_c$, U approaches zero as can be shown be assuming a Gaussian probability distribution for m. In the neighborhood of the critical temperature the scaling ansatz for the magnetization $\langle m(T,L) \rangle = L^{-\beta/\nu} Q_1(L/\xi(t))$ can be used to derive the scaling form for U: $\langle m^4 \rangle = L^{-4\beta/\nu} Q_4(L/\xi(t))$ and $\langle m^2 \rangle = L^{-2\beta/\nu} Q_2(L/\xi(t))$ and consequently

$$\frac{\langle m^4 \rangle}{\langle m^2 \rangle^2} = \tilde{Q}\left(\frac{L}{\xi(T)}\right) \ . \tag{99}$$

For $T = T_c$ the correlation length $\xi(T_c)$ is infinite, hence the right-hand side of Eq. (99) becomes a constant *independent of the system size L*. Thus, a graph of the fourth order cumulant U as a function of temperature for different system sizes L shows a set of curves that intersect at the critical temperature T_c of the *infinite system*.

Another problem in the Monte-Carlo simulation of second-order phase transitions is the phenomenon of critical slowing down: correlations between two states decay on time-scales $t^* \sim \xi^z$, z is called the dynamical critical exponent. At the critical point these times diverge and in a simulation correlations decay only on scales $t^* \sim L^z$ for T close to T_c. This is the reason that the parameters λ_0 and λ introduced in the previous section must be *increased* the larger the system size L becomes. This will lead to a serious increase in the necessary computation time and limits the maximum size L that can be studied in the simulation. The dynamical exponent z depends on the dynamics: $z \approx 2$ for Glauber dynamics, whereas $z \approx 4$ for Kawasaki dynamics. The advantage of the cluster-flip algorithm introduced by Swendsen and Wang is the reduction of the dynamical exponent: $z \approx 0$ and $t^* \sim \log \xi(T)$. Therefore, much larger systems can be studied.

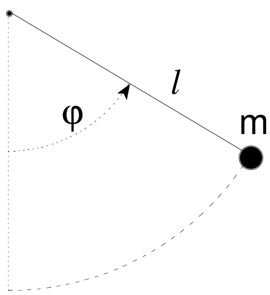
Appendix A: Problems

1. Differentiation

Calculate the derivative of the function $f(x) = \arcsin x$ numerically for $0 \leq x < 1$ using the finite difference formula $f'(x,h) = [f(x+h) - f(x-h)]/(2h)$ for the first derivative of f. Use $h = x_c \epsilon^{1/3}$ with $x_c = 1$. Estimate the error of the numerical result by changing h by factors of 2 and calculate $|f'(x,h) - f'(x,2h)|$. Compare with the exact result. What happens if x is close to 1?

2. Numerical Integration: Pendulum

Use Simpson's rule to calculate the oscillation period $T(\varphi_0)$ of a mathematical pendulum with an amplitude $\varphi(t = 0) = \varphi_0$ and initial angular velocity $\dot{\varphi}(t = 0) = 0$ (see figure to the right). Energy conservation tells you that



$$\tfrac{1}{2} m l^2 \dot{\varphi}^2 + mgl(1 - \cos\varphi) = mgl(1 - \cos\varphi_0) \qquad \text{(A2.1)}$$

(m, l mass and length of the pendulum, g gravitational constant). Therefore, $\dot{\varphi}^2/[\cos\varphi - \cos\varphi_0] = 2g/l$, and

$$T(\varphi_0) = 4 \left(\frac{l}{2g}\right)^{1/2} \int_0^{\varphi_0} \frac{d\varphi}{[\cos\varphi - \cos\varphi_0]^{1/2}} \; .$$

This integral is not very well suited for numerical integration because of the singularity of the integrand at the upper limit of the integral. However, this singularity can be removed using $\cos\varphi = 1 - 2\sin^2(\varphi/2)$ and introducing the new variable ϑ through $\sin(\varphi/2) = \sin(\varphi_0/2)\sin\vartheta$. This substitution transforms the integral into a so-called complete elliptic integral of the first kind:

$$T(\varphi_0) = 4 \left(\frac{l}{g}\right)^{1/2} \int_0^{\pi/2} \frac{d\vartheta}{[1 - k^2 \sin^2\vartheta]^{1/2}} \quad \text{with} \quad k = \sin(\varphi_0/2) \; . \qquad \text{(A2.2)}$$

This integral does no longer have a singularity within the interval of integration and can be calculated using Simpson's rule. For the computation set $\omega_0 = (g/l)^{1/2} = 1$. Create a graph showing T as a function of φ_0. In the same graph also plot the result that is obtained if the small angle approximations $1 - \cos\varphi \approx \varphi^2/2$ and $1 - \cos\varphi_0 \approx \varphi_0^2/2$ are used in equation (A2.1).

For $\varphi_0 \to \pi$ the elliptic integral (A2.2) diverges logarithmically. To illustrate this fact include a plot of the function $f(\varphi_0) = 2\ln[16/(1 - k^2)]$ in the graph as well.

3. Frenkel-Kontorova Model

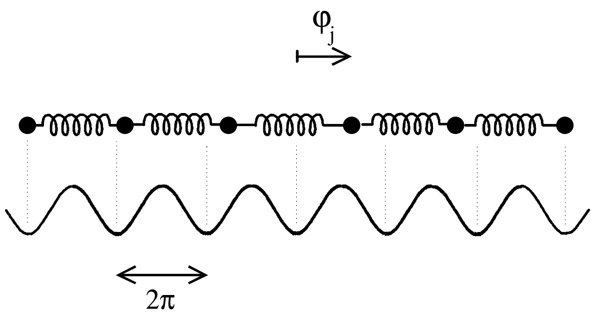
Consider a chain of particles coupled together with harmonic springs in a periodic potential. The total energy of the system is

$$V = \sum_j \left[\tfrac{\kappa}{2}\left(x_{j+1} - x_j - b\right)^2 + \rho\left(1 - \cos kx_j\right) - \tfrac{1}{2}b^2\right] ,$$

where x_j, $j = 1, \ldots, N$ denote the positions of the particles, b is the equilibrium length of the springs, ρ is the strength of the periodic potential, and $k = 2\pi/\lambda$ with λ being the period of the potential. Most of these parameters can be eliminated by a change of coordinates: Set $\varphi_j = kx_j - 2\pi j$, i.e., rescale all lengths with k so that the period of the periodic potential becomes 2π and define the New coordinates φ_j so that they measure the distance from the j-th minimum of the potential. Furthermore, redefine the energy scale so that the factor in front of the contribution of the springs becomes one. The resulting total energy

$$V = \sum_j \left[\tfrac{1}{2}\left(\varphi_{j+1} - \varphi_j - \Delta\right)^2 + u\left(1 - \cos \varphi_j\right) - \tfrac{1}{2}\Delta^2\right] \tag{A3.1}$$

contains only two parameters: u, the redefined strength of the periodic potential, and $\Delta = bk - 2\pi$, the misfit between the equilibrium length of the springs and the period of the potential. The stable configurations of this model correspond to sets of variables



φ_j, $j = 1, \ldots, N$ that minimize the total energy V (A3.1). These are the ground states of the model. If u and Δ are changed, in general the ground state will change as well. The *phase diagram* specifies the ground state of the Frenkel-Kontorova model for each pair of values (u, Δ). Two extreme states can be specified immediately: For $u = 0$ the distances between the particles will be equal to the length of the springs, e.g.,

$$\varphi_{j+1} - \varphi_j = \Delta \ , \ j = 1, \ldots, N-1, \quad \text{and} \quad \begin{cases} \varphi_{N/2} = -\Delta/2 & N \text{ even} \\ \varphi_{\frac{N+1}{2}} = 0 & N \text{ odd} \end{cases} . \tag{A3.2}$$

For very large u the particles will all sit in the minimum of the potential, i.e.,

$$\varphi_j = 0 \ , \ j = 1, \ldots, N . \tag{A3.3}$$

For other values of u and Δ the ground state configurations have to be determined numerically. For a finite number of particles N the boundary conditions must be specified as well. Two cases are of interest:

a) Free boundary conditions: There is only one spring attached to the particles at the end of the chain, i.e., there are only $N-1$ springs. The total energy in this case is

$$V = \sum_{j=1}^{N-1} \left[\tfrac{1}{2} \left(\varphi_{j+1} - \varphi_j - \Delta \right)^2 - \tfrac{1}{2}\Delta^2 \right] + \sum_{j=1}^{N} u \left(1 - \cos\varphi_j \right) \ . \qquad \text{(A3.4)}$$

b) Periodic boundary conditions: The chain is closed at the ends, i.e., there is a spring that connects particle N with particle 1, or in other words the particle at φ_{N+1} is identified with the particle at φ_1. In this case the "winding number" p must be specified as well that determines how much φ_{N+1} differs from φ_1:

$$\varphi_{N+1} = \varphi_1 + 2\pi p \ , \ p \text{ integer.} \qquad \text{(A3.5)}$$

The total energy in this case is

$$\begin{aligned} V = \ & \sum_{j=1}^{N-1} \left[\tfrac{1}{2} \left(\varphi_{j+1} - \varphi_j - \Delta \right)^2 - \tfrac{1}{2}\Delta^2 + u\left(1 - \cos\varphi_j\right) \right] \\ & + \tfrac{1}{2} \left(\varphi_1 + 2\pi p - \varphi_N - \Delta \right)^2 - \tfrac{1}{2}\Delta^2 + u\left(1 - \cos\varphi_N\right) \ . \end{aligned} \qquad \text{(A3.6)}$$

Write a program that determines ground state energies of the Frenkel-Kontorova model for free and/or periodic boundary conditions. The parameters u, Δ, N, and, in the case of periodic boundary conditions, p are input parameters of the program. Use the Newton-Raphson method complemented by a golden section search in cases where the Newton-Raphson step fails to reduce V. As initial configuration use the configuration (A3.2). To compare energies for different N the final result for the total energy (A3.4) or (A3.6) should be given as energy per particle V/N. Plot a typical solution φ_j as a function of j preferably for large N. The routines `tridiag` and `tridiagpb` can be used (see below) to solve the sets of linear equations at each step of the Newton-Raphson method.

Description of the Routines tridiag and tridiagpb

These two routines solve for an array x of length N the tridiagonal set of equations

$$a_i x_{i-1} + b_i x_i + c_i x_{i+1} = r_i \ , \quad i = 1, \ldots, N \ . \qquad (*)$$

In `tridiag` $a_1 = 0 = c_N$, in `tridiagpb` periodic boundary conditions are used: $x_0 = x_N$ and $x_{N+1} = x_1$. The routines are called in the following way:

(i) FORTRAN:

```
double precision a(n),b(n),c(n),r(n),x(n),w(n),anorm
integer n,info
...
call tridiag(a,b,c,r,x,n,anorm,info,w)
```

or

```
call tridiagpb(a,b,c,r,x,n,anorm,info,w)
...
```

a, b, c, r, anorm, n have to be set before calling the routines. a, b, c, r are arrays of length n that specify the matrix elements and the right-hand side of the equation according to eq. ($*$). anorm must be set to a value that is of the typical size of the matrix elements, e.g., anorm $= max(|a_i|, |b_i|, |c_i|,\ i = 1, \ldots, N)$. n must specify the length of the arrays a, b, c, r. If after the call the variable info=1, x will contain the solution of the system of equations ($*$). If info=2, this indicates that your matrix is singular and no solution can be obtained. The array w is used as workspace by the routine. The compiler must be told that the routines can be found in the library /usr/lib/libcomphys.a:

```
f77 myprog.f -lcomphys
```

(ii) C:

```
void tridiag_(double *,double *,double *,double *,double *,int *,
              double *,int *,double *);
void tridiagpb_(double *,double *,double *,double *,double *,int *,
                double *,int *,double *);

double *a,*b,*c,*r,*x,*w,anorm;
int n,info;
...
a=(double *)malloc(n*sizeof(double));
b=(double *)malloc(n*sizeof(double));
c=(double *)malloc(n*sizeof(double));
r=(double *)malloc(n*sizeof(double));
x=(double *)malloc(n*sizeof(double));
w=(double *)malloc(n*sizeof(double));
...
tridiag_(a,b,c,r,x,&n,&anorm,&info,w)
```

or

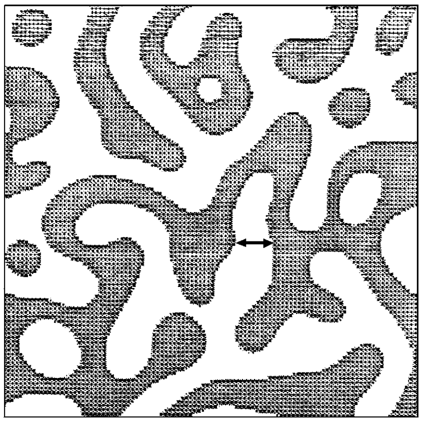
```
tridiagpb_(a,b,c,r,x,&n,&anorm,&info,w)
...
```

The meaning of the variables is the same is for FORTRAN. The compiler call must call the FORTRAN libraries:

```
cc myprog.c -lF77 -lU77 -lcomphys
```

Note that tridiagpb will change the arrays a, b, c, r in the calculation. Thus, if you still need the arrays after the call to tridiagpb you must make a copy of these arrays before the call.

4. Density-Density Correlation Function



Consider an alloy that contains two kinds of particles, say A and B, with the following properties: At high temperatures the particles A and B mix perfectly, the alloy is said to be in a disordered state. At low temperatures A particles attract each other as the B particles do, whereas A and B particles repel each other. Therefore A and B particles tend to separate and regions with only A particles and regions with only B particles are formed. If this alloy is cooled down quickly from high temperatures to low temperatures first small domains of A and B particles are formed. These domains then grow in size: the alloy becomes more and more ordered. A typical configuration is shown in the figure to the right. The quantity of interest in this situation is the average domain size R indicated with an arrow in the figure. To describe the situation the density $\rho(\mathbf{r})$ is introduced with $\rho(\mathbf{r}) = 1$ if there is an A particle at the site $\mathbf{r}$ and $\rho(\mathbf{r}) = 0$ otherwise. It is customary, however to study the quantity $M(\mathbf{r}) = 2\rho(\mathbf{r}) - 1$ instead, i.e., $M(\mathbf{r}) = 1$ at A sites and $M(\mathbf{r}) = -1$ at B sites. The density-density correlation function is defined as $C(r) = V^{-1} \sum_{\mathbf{r}'} M(\mathbf{r} + \mathbf{r}')M(\mathbf{r}')$, where $\mathbf{r}$ and $\mathbf{r}'$ run over all lattice sites within the measurement volume V. The function C depends only on the distance $r = |\mathbf{r}|$ between the two points where M is measured.

A typical measurement of the correlation function C can be found in the file `~siegert/comphys/c.data` on the Physics workstations. The file contains three columns: r_i, C_i, and σ_i, $i = 1, \ldots, N$. The first column contains the points r at which the measurements of C were taken that are listed in the second column. The third column contains an estimate $\sigma_i = 0.02$ for the standard deviations of these measurements. The first zero of the correlation function $C(r)$ can be taken as a measure for the average domain size R. The correlation function has an oscillatory form $C(r) \simeq c_0 \cos(qr)d(r)$ with a function d that decays exponentially, $d(r) \sim \exp(-br)$, but is unknown otherwise. To determine this domain size and other properties of the correlation function write a program that uses the Levenberg-Marquardt algorithm to fit the measured data C_i to the following fitting functions:

$$
\begin{aligned}
&(1) \qquad C(r) = c_0 \cos(qr + \varphi) e^{-br} \\
&(2) \qquad C(r) = c_0 \cos(qr + \varphi)(1 + br) e^{-br} \\
&(3) \qquad C(r) = c_0 \cos(qr + \varphi) \frac{e^{-br}}{1 + (br)^2} \, .
\end{aligned}
$$

In order to estimate systematic errors first fit the data by setting $\varphi = 0$ and only fitting the three parameters c_0, q, and b. In a second set of least square fits perform a 4 parameter fit by allowing φ to vary as well. In any case your program should determine the optimal choices for the parameters, the corresponding value of χ^2, the standard deviations of the parameters, and the correlation coefficients r_{kl}. Plot the data together with the fitting function that yields the smallest χ^2 as a function of r. Plot the residuals $R_i = C_i - C(r; \mathbf{a})$ preferably in the same graph but on a different scale.

The Levenberg-Marquardt algorithm requires the solution of a system of linear equations at each step. This can be accomplished using the routine `dgesv` of the public domain library package LAPACK that has been installed on the workstations.
Usage for C programmers: (The variable `n` corresponds to the number of parameters in the fitting function)
Prototype:

```
void dgesv_(int *,int *,double *,int *,int *,double *,int *,int *);
```

Function call:

```
double **a,**b;
int n,nrhs,info,*ipiv,lda,ldb;
...
lda=n;
ldb=n;
dgesv_(&n,&nrhs,&a[0][0],&lda,&ipiv[0],&b[0][0],&ldb,&info);
```

For a description of the different parameters type `man dgesv`. To allocate the storage for the matrices `a` and `b` you can use the routine `matrix` that you find in the file `~siegert/comphys/matrix.c`. The routine `dgesv` can be used to solve a set of linear equations (`nrhs=1`) and to find the inverse of a matrix as well. In the latter case set `b` to the unity matrix and set `nrhs=n`.

To create an executable of your program `myprog.c` use the following two step process:

```
cc -c myprog.c
f77 -o myprog myprog.o -lm -llapack -lblas
```

5. Henon-Heiles System

Use the leapfrog algorithm to compute the trajectories for the Lagrange functions

$$\mathcal{L} = \frac{1}{2}\left(p_1^2 + p_2^2\right) - V(q_1, q_2) \qquad \text{with} \qquad p_i = \dot{q}_i \ , \quad i = 1, 2 \tag{A5.1}$$

where the following two cases for the potential V are to be investigated:

$$\begin{aligned} V_T(q_1, q_2) &= \tfrac{1}{24}\left(e^{2q_1 - 2q_2\sqrt{3}} + e^{2q_1 + 2q_2\sqrt{3}} + e^{-4q_1}\right) - \tfrac{1}{8} && \text{(A5.2)} \\ V_H(q_1, q_2) &= \tfrac{1}{2}\left(q_1^2 + q_2^2\right) + q_1 q_2^2 - \tfrac{1}{3} q_1^3 \ . && \text{(A5.3)} \end{aligned}$$

Eq. (A5.2) is the potential of the so-called Toda lattice, whereas eq. (A5.3) specifies the potential of the Henon-Heiles system that is obtained by expanding V_T up to cubic terms in q_1 and q_2. Both systems can also be regarded as a perturbed two-dimensional harmonic oscillator. The calculated trajectories are to be plotted in the form of Poincaré maps $p_1(q_1)$: Every time the trajectory crosses the plane $q_2 = 0$ a point is added to the Poincaré map. These points can be calculated in the following way: If q_2 changes sign between two steps of the iteration $q_{2,n}$, $q_{2,n+2}$, estimate the step size that would lead to $q_2 = 0$ using the Euler method: $\Delta t^* \approx -q_{2,n}/p_{2,n+1}$. Use this step size Δt^* to determine the other quantities q_1, p_1, p_2 at the point with $q_2 = 0$ using the Euler method as well. Use these values only to determine the points of the Poincaré map. The integration of the differential equations should continue with the leapfrog algorithm as usual.

The program should print the values q_1, p_1, p_2 together with the total energy of the system for all points of the trajectories with $q_2 = 0$. In the case of the Toda lattice compute additionally the quantity

$$A = 8p_2(p_2^2 - 3p_1^2) + (p_2 + p_1\sqrt{3})e^{2q_1 - 2q_2\sqrt{3}} + (p_2 - p_1\sqrt{3})e^{2q_1 + 2q_2\sqrt{3}} - 2p_2 e^{-4q_1} \tag{A5.4}$$

at these points (can you show analytically that this is a constant of motion?).

Generate the Poincaré maps for the energies $E = 1/8$ and $E = 1/6$ for both potentials. As initial conditions for the integration of the differential equations always choose $q_2 = 0$, because that already gives the first point for the Poincaré map. Several combinations of initial values for q_1, p_1 should be tried to generate the complete Poincaré map. The initial value for p_2 is always determined by energy conservation. The Poincaré map for the Henon-Heiles system with energy $E = 1/6$, however, can be generated from *one* trajectory ($\Rightarrow$ fully developed chaos).

6. Flight to the Moon

Compute the trajectory of a rocket that is launched towards the moon from the surface of the earth at $\mathbf{r} = (x, y) = (r_E, 0)$ with an initial velocity $\mathbf{v} = (v_x, v_y) = (v_0, 0)$. The trajectory of the rocket is influenced by the gravitational forces of the earth and the moon, all other forces shall be neglected. The trajectory of the moon is assumed to be circular and lies in the $x - y$ plane, whereas the earth is assumed to be at rest. At time $t = 0$, when the rocket is started, the moon is located at $D(\cos\alpha, -\sin\alpha)$ and rotates counterclockwise around the earth. Use the following parameters: mass of the earth $M_E = 5.98 \times 10^{24}$kg, radius of the earth $r_E = 6.38 \times 10^6$m, mass of the moon $M_M = 7.35 \times 10^{22}$kg, radius of the moon $r_M = 1.74 \times 10^6$m, orbital period of the moon $T_M = 27.3$d, radius of the moon's orbit $D = 3.84 \times 10^8$m, gravitational constant $\gamma = 6.67 \times 10^{-11}\mathrm{Nm}^2/\mathrm{kg}^2$. The initial parameters v_0 and α are input parameters of the program. The aim is to choose values v_0 and α so that the rocket flies around the moon as closely as possible and hits the earth on the way back. Reasonable first guesses for v_0 and α can be obtained by setting the mass of the moon to zero: In that case the velocity that is necessary to shoot the rocket exactly to the trajectory of the moon is

$$v_D = \left[2\gamma M_E \left(\frac{1}{r_e} - \frac{1}{D}\right)\right]^{1/2} .$$

The corresponding time to reach the moon is

$$T_D = \sqrt{\frac{D}{2\gamma M_E}} \left[r_E \sqrt{\frac{D}{r_E} - 1} + D \arctan \sqrt{\frac{D}{r_E} - 1} \right]$$

(why?). Use the fourth-order Runge-Kutta algorithm with adaptive step-size control to integrate the equations of motion. Plot the trajectory $y(x)$. In that plot the position of the moon for the time when the distance rocket-moon is minimal should be indicated. The program should return this minimal distance and the minimal distance rocket-earth on the way back as well.

7. Two-Dimensional Ising Model

Perform a Monte-Carlo simulation of the two-dimensional Ising model on a quadratic lattice with periodic boundary conditions for $J > 0$ and $h = 0$, i.e., a ferromagnet without magnetic field. Use Glauber dynamics (spin-flips), Metropolis rates, and sequential updates of the spins. The energy and the magnetization of the system are

$$E = -J \sum_{\langle i,j \rangle} S_i S_j \qquad \text{and} \qquad M = \sum_{i=1}^{L^d} S_i \ .$$

Here $\langle i,j \rangle$ denotes a sum over all nearest neighbor pairs. The program has to calculate the following quantities:

(a) the average magnetization per spin $\langle m \rangle = L^{-d} \langle M \rangle$ with $d = 2$. The magnetization itself is calculated as $\langle M \rangle = \frac{1}{n} \sum_{l=1}^{n} |M|$, where the sum runs over the configurations $\mathcal{C}_l$ that are used to calculate averages, see point (v) below;

(b) the susceptibility $\chi = \beta L^{-d} [\langle M^2 \rangle - \langle M \rangle^2] = \beta L^d [\langle m^2 \rangle - \langle m \rangle^2]$;

(c) the energy per particle $\langle e \rangle = L^{-d} \langle E \rangle$ with $\langle E \rangle = \frac{1}{n} \sum_{l=1}^{n} E$;

(d) the specific heat per particle $c = k_B \beta^2 L^{-d} [\langle E^2 \rangle - \langle E \rangle^2] = k_B \beta^2 L^d [\langle e^2 \rangle - \langle e \rangle^2]$;

(e) the fourth order cumulant $U = 1 - \langle m^4 \rangle / (3 \langle m^2 \rangle^2)$;

(f) the standard deviation of χ in order to estimate the error of the value of χ.

The following procedure should be incorporated:

(i) Initialize the spins of the system in a completely ordered state, i.e., $S_i = 1$ for all $i = 1, \ldots, L^d$.

(ii) Set the initial values for the magnetization M and the energy E corresponding to the initial configuration.

(iii) The basic step: For a particular spin S_i calculate the energy change that is obtained, if the spin would be flipped, $\Delta E = 2JS_i \sum_{\langle j \rangle_i} S_j$, where the sum runs over all nearest neighbor sites of the site i. Determine the Boltzmann factor $e^{-\beta \Delta E}$. Draw a random number z. If $z < e^{-\beta \Delta E}$, flip the spin and update the magnetization M and energy E, otherwise leave everything as it is. Continue with the next spin using a sequential updating method. One run through the whole system from $i = 1$ to $i = L^d$ is called a Monte-Carlo step (MCS).

(iv) Starting from the initial configuration perform λ_0 MCS without calculating averages.

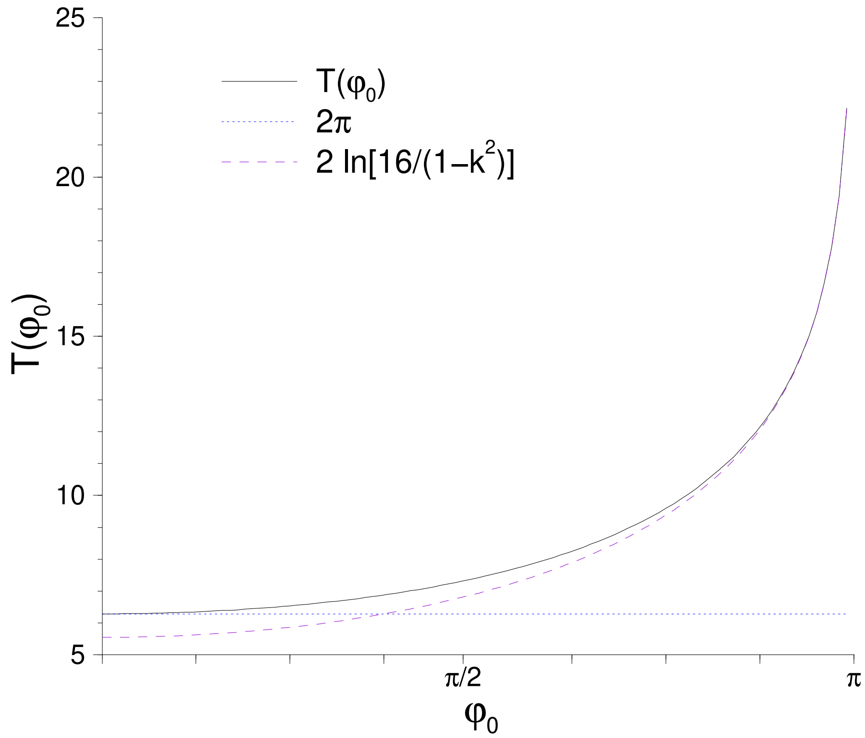
(v) Perform $n\lambda$ more MCS. After each λ MCS update the averages, e.g., $\langle M \rangle = \frac{1}{n} \sum_{l=1}^{n} |M|$. Thus, only the current configurations after each λ MCS are used to calculate averages.

Input parameters of the program are: the temperature $k_B T / J$, the linear size of the system L, the initial number λ_0 of MCS, the number λ of MCS between updates, and the number n of updates to be taken. The simulation shall be carried out for several system sizes L and temperatures T. Plot $\chi(T, L)$, $c(T, L)$ and $U(T, L)$ as functions of T (data for several system sizes in the same graph). Determine (graphically) the maximum $\chi_{\max}(L) = \chi(T_{\chi,\max}, L)$ of the susceptibility with respect to temperature for each system size, analogously determine $c_{\max}(L)$. Determine the critical temperature T_c from the intersection of the graphs of $U(T, L)$ for different system sizes. Plot $\chi_{\max}(L)$, $c_{\max}(L)$, and $|T_{\chi,\max}(L) - T_c|/T_c$ as functions of system size L in a log-log plot and extract the critical exponents γ, α, and ν. In order to determine α you also may want to plot $c_{\max}(L)$ on a semilogarithmic scale.

In particular the part of the program that implements one Monte-Carlo step should be as efficient in terms of computation time as possible, e.g., `if` statements or modulo operations aimed at implementing the periodic boundary conditions should be avoided. The Boltzmann weights $e^{-\beta\Delta E}$ should be calculated at the beginning of the program for all possible values of ΔE and stored in an array. For the generation of random numbers use the program `~siegert/comphys/randv.c` that generates an array `r` of random numbers. This routine is much faster, if several random numbers are generated in one call. An example of an implementation of the routine can be found in the file `~siegert/comphys/test_rng.c`.

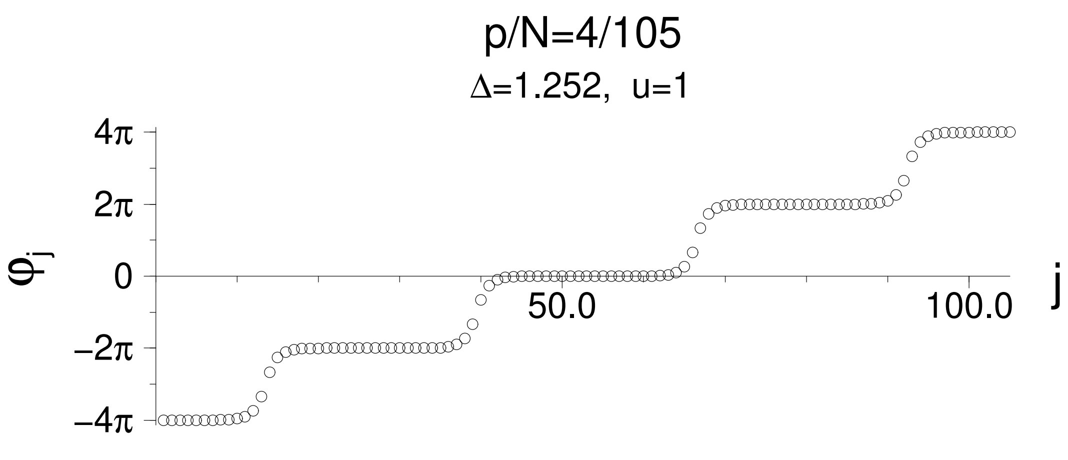
Appendix B: Solutions to the Problems, Graphs

2. Numerical Integration: Pendulum



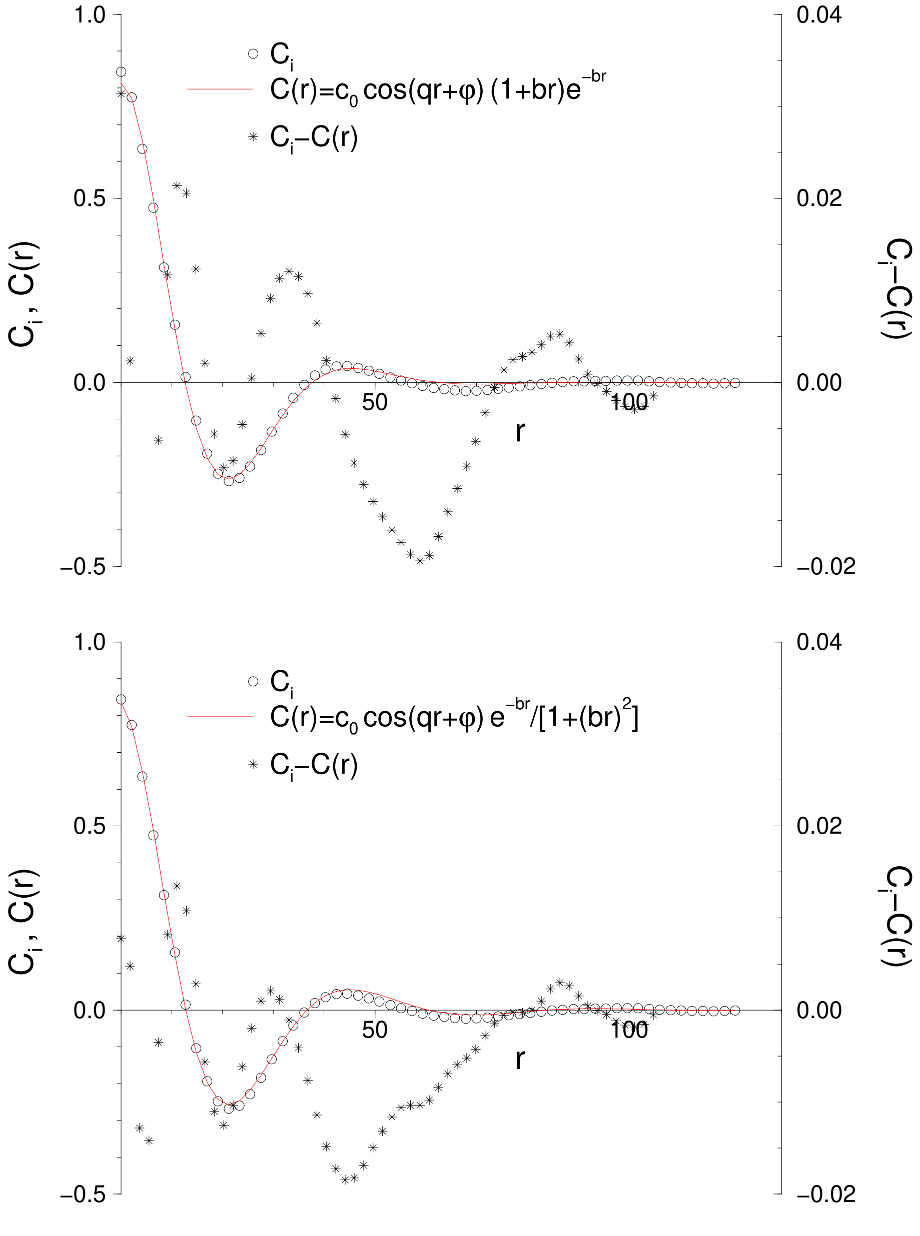
Oscillation period of a pendulum as a function of the initial angle φ_0. The to limiting cases $T(\varphi_0 \to 0) = 2\pi$ and $T(\varphi_0 \to \pi) = 2\ln[16/(1-k^2)]$, $k = \sin(\varphi_0/2)$, are also indicated.

3. Frenkel-Kontorova Model



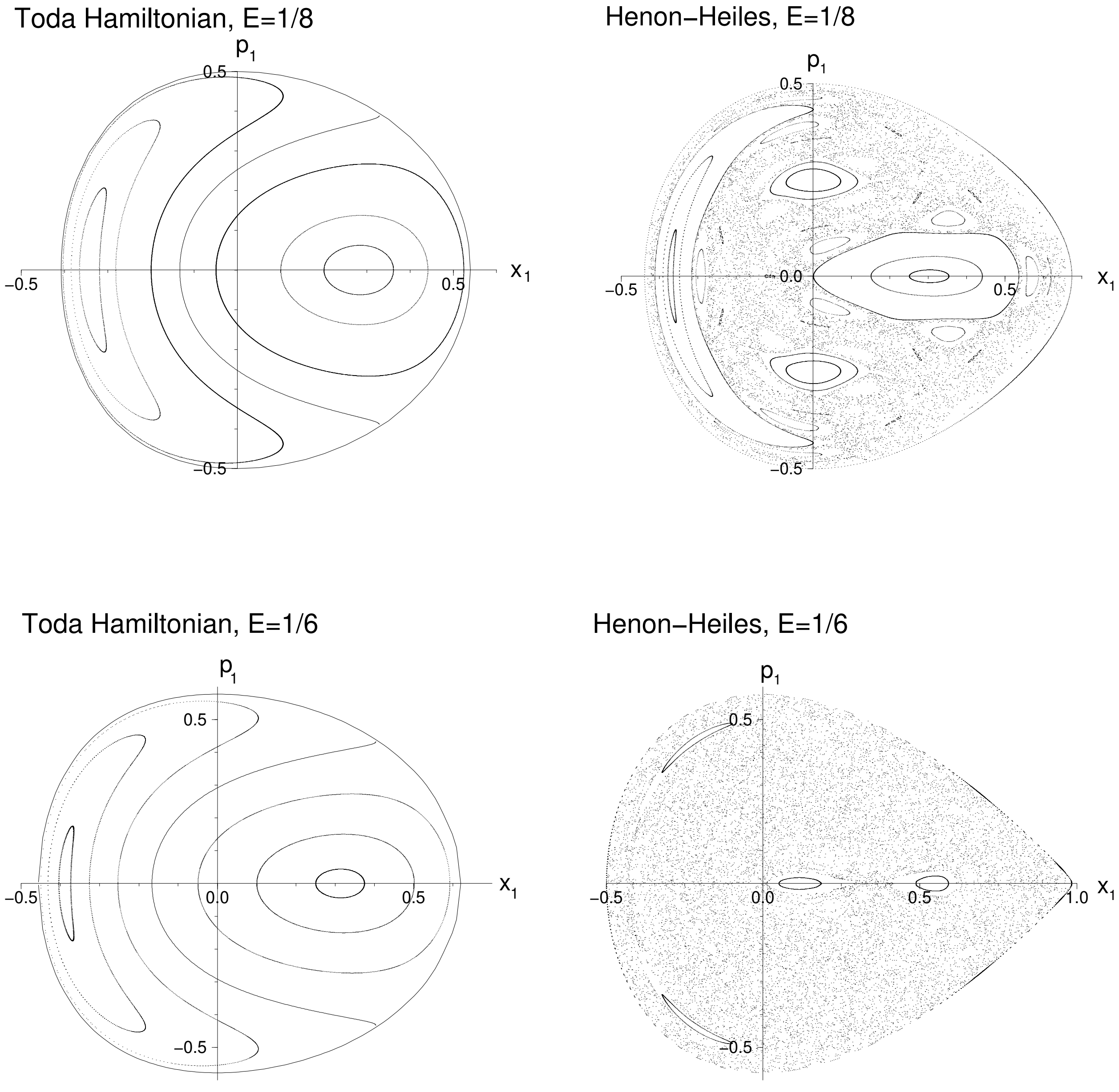
Typical ground-state configuration of the Frenkel-Kontorova model with periodic boundary conditions. The parameter p sets the number of defects (or solitons) in the solutions, here $p = 4$ for $N = 105$ particles.

4. Density-Density Correlation Function



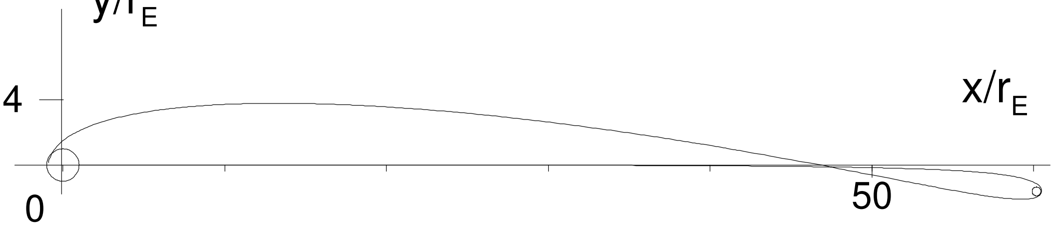
Least-squares fit of the density-density correlation function; the original data C_i and the fitting function $C(r)$ are plotted (left axis) as well as the residuals (right axis). $\chi^2/(N-M) = 0.339$ for the upper fit, $\chi^2/(N-M) = 0.193$ for the lower fit with $N = 58$, $M = 4$.

5. Henon-Heiles System



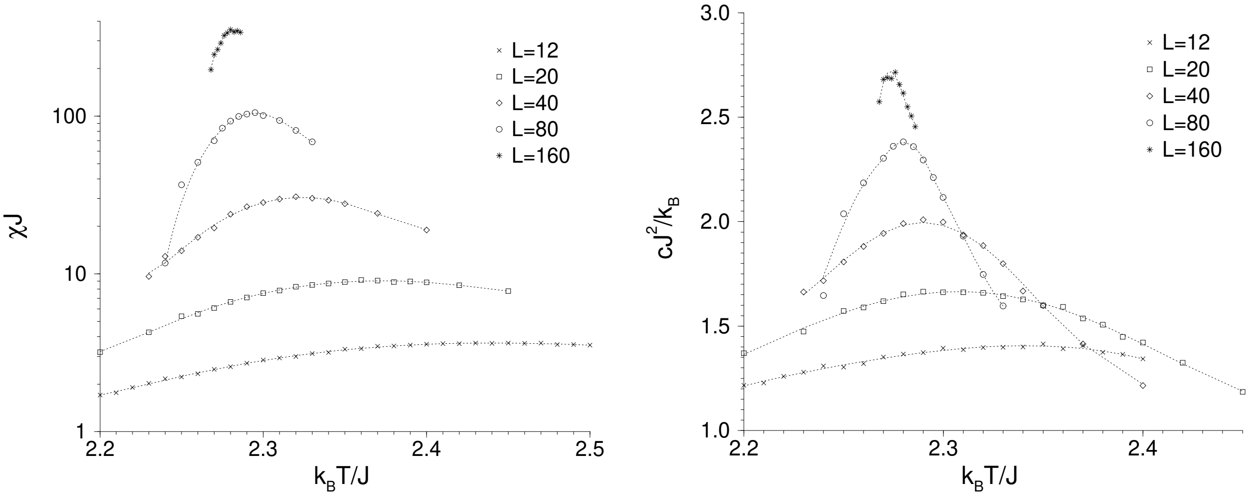
Poincaré maps for the Toda Hamiltonian and the Henon-Heiles system for energies 1/8 and 1/6

6. Flight to the Moon

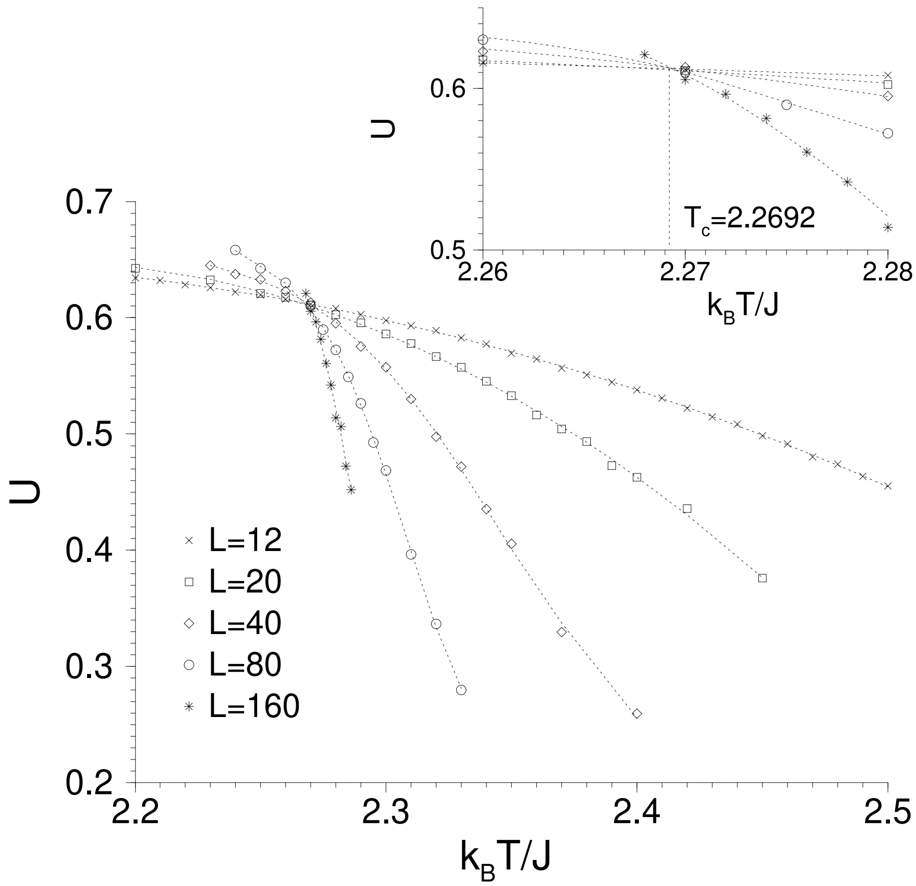


Trajectory of a rocket launched towards the moon with an initial velocity of $v_0 = 1.0015 v_D$, the angle that determines the position of the moon at the time of the start is $\alpha = 0.734$. The minimum distance of the trajectory from the center of the moon is 1.2 moon radii, the corresponding position of the moon is indicated in the graph as well as the size of the earth.

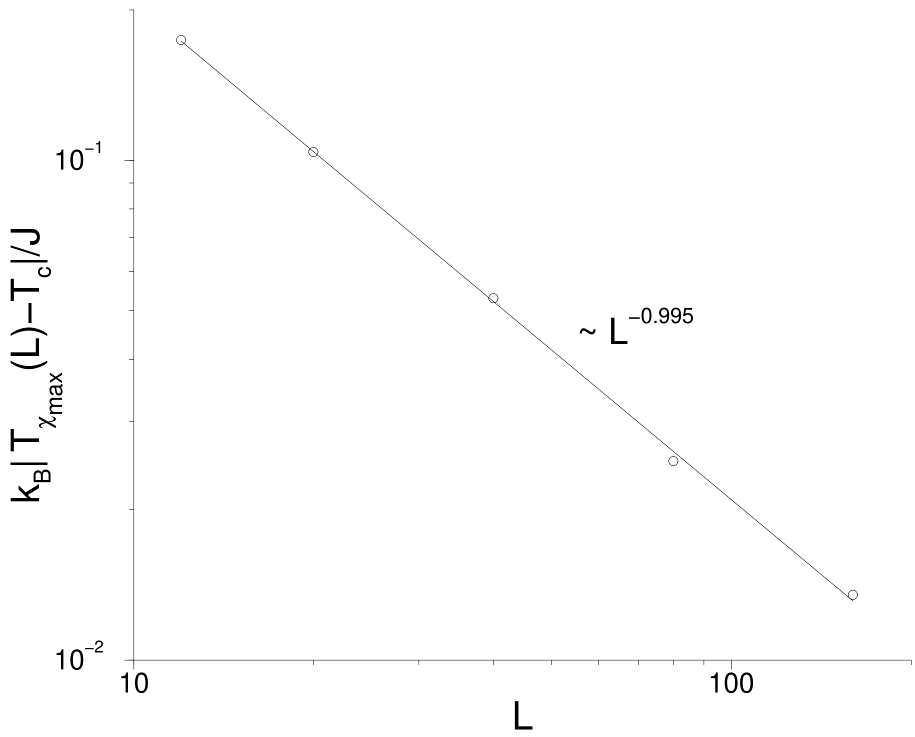
7. Two-Dimensional Ising Model



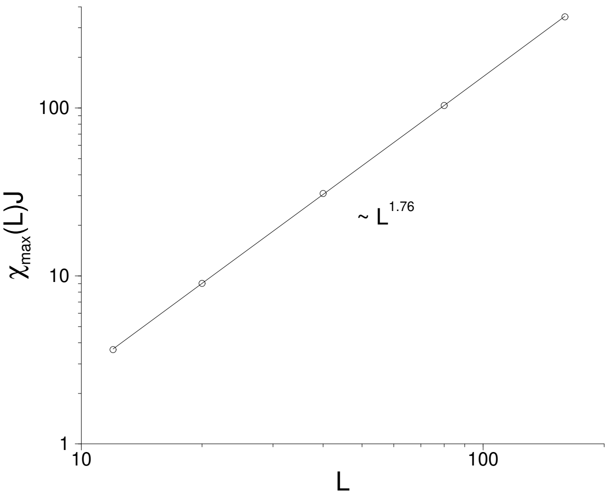
Susceptibility (left) and specific heat (right) as a function of temperature for several system sizes for the two-dimensional Ising model. The data for the susceptibility have been plotted on a logarithmic scale because of the large spread in the data. The lines are drawn as a guide to the eye.

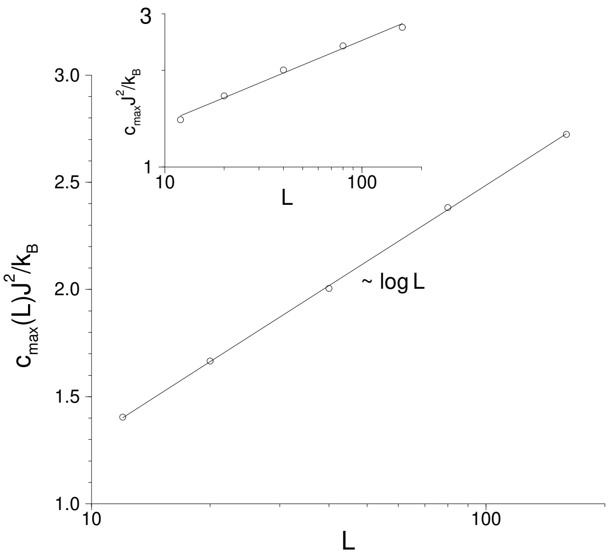


Fourth cumulant $U = 1 - \langle m^4 \rangle / (3\langle m^2 \rangle^2)$ as a function of temperature for different system sizes L. The lines are drawn as a guide to the eye. The inset shows an enlargement of the interval around the critical temperature. The intersection of the curves for different system sizes is obtained as $T_c \simeq 2.269 \pm 0.0004$ from these data.



Shift in the peak position of the susceptibility as obtained form the susceptibility data shown on the previous page as a function of system size. The straight line corresponds to a power-law fit $|T_{\chi_{\max}}(L) - T_c| \sim L^{-1/\nu}$ with $\nu \simeq 1.0 \pm 0.01$. The value $T_c \simeq 2.269$ was used as obtained from the cumulant U.





Maximum of the susceptibility (left) and of the specific heat (right) as a function of system size. The data for the susceptibility have been fitted to a power law $\chi_{\max}(L) \sim L^{\gamma/\nu}$ with $\gamma/\nu \simeq 1.76 \pm 0.01$. The data for the specific heat have been plotted on a semilogarithmic scale and the straight line corresponds to a behavior $c_{\max}(L) \sim \log L$ indicating an exponent $\alpha \simeq 0$. The same data plotted on a log-log scale (see inset) do not fall on a straight line.